UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA

Progettazione di controllo di un minifrigo basato su cella di Peltier e controllore PID

Relatore: Prof. Meneghini Matteo

Laureando: Molin Paolo

ANNO ACCADEMICO 2022 – 2023

Data di laurea 27/09/2023

Sommario

1 Introduzione

Con le invenzioni collegate all'utilizzo dell'energia elettrica l'umanità ha effettuato enormi passi in avanti verso il miglioramento della qualità della vita.

In particolare, sono state due le invenzioni che hanno permesso tale miglioramento:

- L'utilizzo dell'energia elettrica per illuminare i luoghi di vita delle persone durante le ore notturne o quando la luce del sole non è sufficientemente forte;
- L'utilizzo dell'energia elettrica per una migliore e più efficiente conservazione dei cibi.

Alla luce di questi ragionamenti, mi sono chiesto cosa avrei potuto realizzare nei tempi e modi a disposizione e ho optato per la realizzazione di uno strumento che permetta la conservazione dei cibi: un minifrigo.

Gli strumenti in circolazione per la conservazione dei cibi riprendono quanto sviluppato da Ferdinand Carré[1] nel 1859 ed utilizzano un sistema di condensazione dell'ammoniaca. Tale sistema sarebbe stato pericoloso da progettare per le sostanze adoperate, per cui ho optato per un componente termoelettrico che, se attivato, produce una diversa temperatura tra le sue due facce: la cella di Peltier.

Per la realizzazione del frigo ho bisogno, oltre alla cella di Peltier, di:

- un alimentatore in grado di fornire potenza a sufficienza a tutti i carichi collegati;
- un controllore che effettui la parte di controllo PID;
- un dispositivo che permetta di pilotare e grandi correnti con piccole tensioni;
- un sensore di temperatura che possa lavorare nel range di temperature di un frigorifero domestico;
- una ventola per il raffreddamento della cella per evitare un surriscaldamento eccessivo;
- uno schermo per permettere all'utente di conoscere la temperatura interna e impostata;
- una struttura isolata termicamente dall'esterno.

2 Arduino

Per gestire il controllo ho deciso di usare una scheda Arduino, in quanto: ho già lavorato con questo tipo di schede, sono molto versatili e il linguaggio di programmazione utilizzato è simile a C. Arduino è stata ideata e sviluppata nel 2005 da alcuni membri dell'Interaction Design Institute di Ivrea come strumento per la prototipazione rapida, per scopi didattici e professionali[2]. Schemi circuitali e software di sviluppo sono open-source, per questo motivo è molto utilizzato nella didattica educativa. Le schede Arduino sono dotate di ingressi e uscite analogiche e digitali, permettendo di interfacciare le schede con vari sensori ed attuatori.

Fig. 2.1 - Logo Arduino[3]

2.1 Arduino UNO R3

Arduino ha realizzato molte schede, tra queste c'è Arduino UNO, una scheda che possiede ingressi analogici e digitali, uscite PWM a 8 bit, comunicazione SPI, I^2C e seriale.

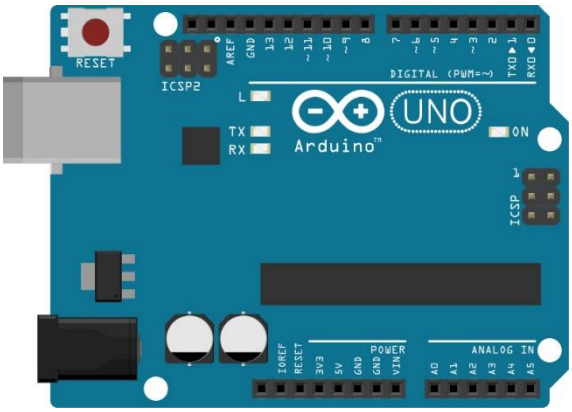

Fig. 2.2 - Scheda Arduino UNO (Fritzing)[4]

Oltre alle caratteristiche sopracitate, la scheda presa in esame presenta le seguenti caratteristiche:

Microcontrollore	ATmega328P
Tensione di funzionamento	5 [V]
Tensione in ingresso	7 – 12 [V]
I/O digitali	14 (di cui 6 con output PWM)
Ingressi analogici	6
Risoluzione ADC	10 bit
Corrente continua fornita dai pin I/O	40 [mA]

Corrente continua per 3.3 [*V*]	50 [*mA*]
Flash Memory	32 kB
SRAM	2 kB
EEPROM	2 kB
Velocità di Clock	16 [*MHz*]

Tab. 2.1 - Tabella caratteristiche Arduino UNO[5]

3 Cella di Peltier

Come sistema di raffreddamento dell'ambiente ho usato un modulo di Peltier, un componente che, se alimentato con tensione fissa, comporta il riscaldamento di una faccia, mentre l'altra si raffredda. Se si abbassa la temperatura del lato caldo, allora il lato freddo abbasserà la sua temperatura di conseguenza, cercando di mantenere una differenza di temperatura costante.

3.1 Principio di funzionamento

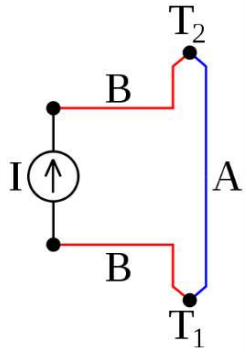

Fig. 3.1 - Schema di un dispositivo Peltier[6]

Il funzionamento della cella di Peltier si basa sull'effetto di Peltier[7]: quando una corrente I viene fatta scorrere nel circuito in Fig. 3.1, dove A e B sono dei conduttori connessi da delle giunzioni T_1 e T_2, una quantità di calore è assorbita dalla giunzione T_1 ed emessa dalla giunzione T_2. La quantità di calore assorbita da T_1 per unità di tempo è:

$$Q = \Pi_{AB} I = (\Pi_B - \Pi_A) I$$

Dove Π_{AB} è il coefficiente di Peltier della termocoppia, mentre Π_A e Π_B sono i coefficienti dei singoli materiali.

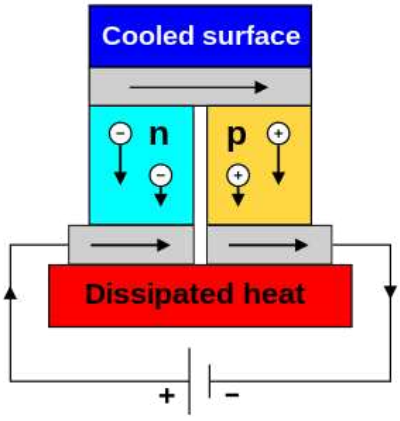

Fig. 3.2 - Schema di funzionamento di una cella di Peltier[8]

In particolare, semiconduttori di tipo P hanno solitamente un coefficiente positivo, mentre quelli N negativo. Il fenomeno utilizzato è l'effetto del ritorno all'equilibrio degli elettroni che assorbono energia da un contatto e la cedono all'altro.

Una comune cella Peltier è formata da due materiali semiconduttori drogati di tipo N e di tipo P, collegati tra loro da una lamella di rame. Se si applica al tipo N una tensione positiva e al tipo P una tensione negativa, la lamella superiore si raffredda mentre quella inferiore si riscalda. Invertendo la tensione lo spostamento di energia termica viene invertito. La sottrazione di calore è favorita dalla creazione di opportuni ponti termici (adesivi o termoconduttivi) che permettono al meglio la conduzione. Il calore sottratto è trasferito sul lato caldo, assieme al calore di funzionamento comportando un abbassamento dell'efficienza, dal lato caldo il calore deve essere invece traferito all'ambiente esterno.[9]

3.2 TEC1-12710

La maggior parte delle celle di Peltier presenta una scritta, tipicamente posta sul lato freddo, che ne riassume le caratteristiche principali. L'identificatore è solitamente nella forma TEX#-NNNAA, dove TE sta per Thermo Electric (converter), X è una lettera che descrive le dimensioni della cella (C = Standard, S = Small), # rappresenta il numero di stadi e le due serie di numeri dopo il trattino sono, rispettivamente il numero NNN di elementi e la corrente nominale AA in Ampère. La cella presa in esame TEC1-12710 è una cella di dimensioni standard, a uno stadio, composta da 127 elementi ed una corrente nominale di 10 $[A]$. Alimentandola a 12 $[V]$ avrà una potenza nominale assorbita di 120 $[W]$.

Di seguito vengono riportati dati presi dal datasheet fornito dal venditore:

Hot side temperature:	$\mathbf{25°C}$	$\mathbf{50°C}$
V_{MAX} $[V]$	15.2	17.4
I_{MAX} $[A]$	10.5	10.5
Q_{MAX} $[W](\Delta T = 0)$	85	96
ΔT_{MAX} $[°C](Q_C = 0)$	66	75
I_{in} $[A]$ (Opt)	8	$\cong 7$
Q_C $[W]$ (Opt)	$\cong 44$	$\cong 42$

Tab. 3.1 - Dati della cella TEC1-12710 [10]

I dati con la notazione (Opt), presenti nella Tab. 3.1, indicano il punto di operatività ipotetico ($\Delta T = 30°C$ e $V_{CC} = 12$ $[V]$), che sono stati ricavati (indicati in rosso) dai grafici riportati in Fig. 3.3.

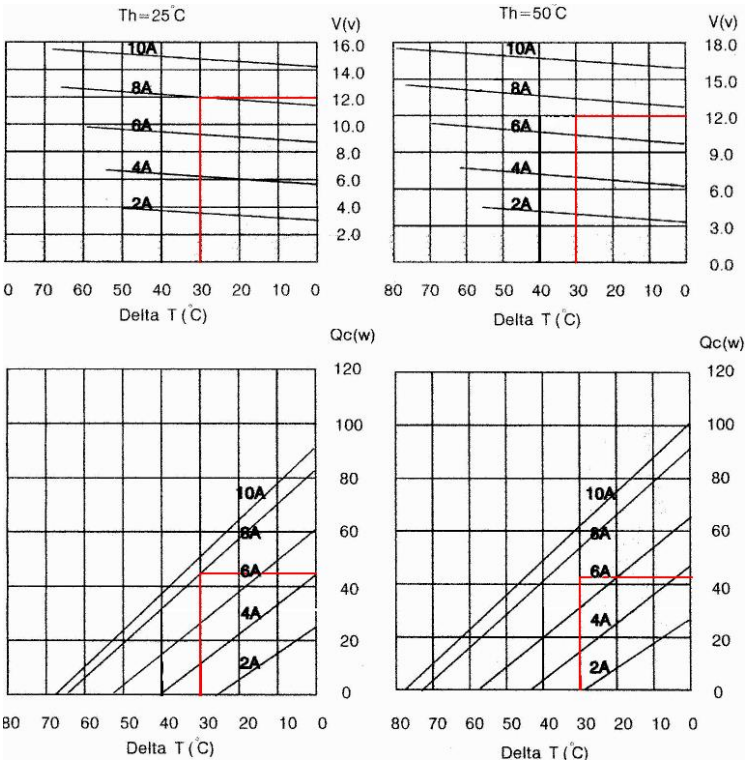

Fig. 3.3 - Grafici riportanti le caratteristiche Tensione e Calore assorbito al variare di ΔT della cella TEC1-12710[11]

4 MOSFET di potenza

Arduino non può sostenere le tensioni e correnti richieste dal modulo di Peltier, per cui è necessario dover pilotare la cella con un transistor che riesca a gestire le grandezze elettriche in gioco.

Sul mercato sono presenti moduli per il controllo di velocità di un motore, i quali però non sono in grado di sostenere grandi correnti per lungo tempo, ho deciso quindi di utilizzare un MOSFET di potenza e comandarlo con Arduino tramite PWM (Pulse Width Modulation).

4.1 IRL530N

Come MOSFET utilizzato per comandare la cella di Peltier, ho scelto un transistor MOS a canale N in modo sia più semplice pilotare la tensione di gate-source: IRL530N. Riporto i parametri di interesse presenti nel datasheet fornito dai costruttori:

$V_{DS_{MAX}}$	$100\ [V]$
$V_{GS_{MAX}}$	$10\ [V]$
$V_t(V_{GS} = V_{DS}, I_D = 150\ [\mu A])$	$1 \div 2\ [V]$
$I_D\ (V_{GS} = 5\ [V])$	$15\ [V]$

Tab. 4.1 - Dati d'interesse del MOSFET IRL530, ricavati dai grafici presenti nel datasheet del componente[12]

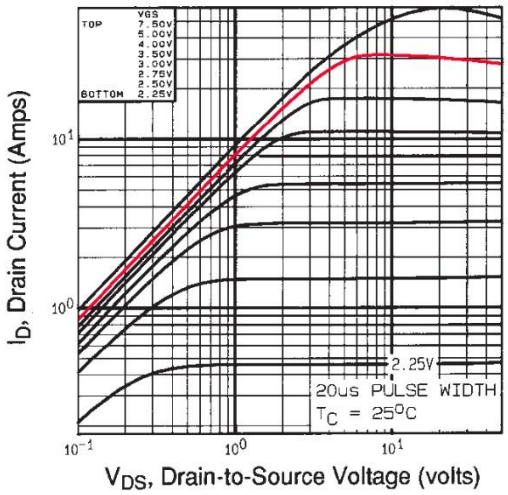

Fig. 4.1 - Transcaratteristica tensione di drain-source e corrente di drain[13]

Nel grafico riportato in Fig. 4.1 ho evidenziato in rosso l'andamento della corrente di drain all'aumentare della tensione V_{DS} con $V_{GS} = 5\ [V]$, gli andamenti sottostanti sono per valori di tensione gate-source inferiori.

4.2 Pulse Width Modulation

Il Pulse Width Modulation (PWM - tradotto Modulazione a Larghezza di Impulso) consiste nel modificare la larghezza di un'onda quadra, mantenendo però costante la frequenza di modulazione.

Definiamo t_{on} l'intervallo temporale, all'interno di un periodo T, in cui il segnale rimane alto, mentre con t_{off} l'intervallo di tempo in cui il segnale rimane basso, si ha che $T = t_{on} + t_{off}$. Nel caso di Arduino, si ha che la frequenza di modulazione è di $500\ [Hz]$, quindi un periodo $T = 2\ [ms]$, con una tensione a livello alto $V_H = 5\ [V]$ e a livello basso $V_L = 0\ [V]$. La tensione media su periodo di modulazione risulta quindi:

$$\overline{v_O}(t) = \frac{1}{T}\int_{[T]} v_O(t)dt = \frac{1}{T}\int_0^T v_O(t)dt = \frac{1}{T}\int_0^{t_{on}} v_O(t)dt + \frac{1}{T}\int_{t_{on}}^{t_{on}+t_{off}} v_O(t)dt =$$

$$= \frac{1}{T}\int_0^{t_{on}} V_H dt + \frac{1}{T}\int_{t_{on}}^{t_{on}+t_{off}} V_L dt = V_H\frac{t_{on}}{T} + V_L\frac{t_{off}}{T} = \delta V_H$$

Definendo duty-cicle con δ come il rapporto tra t_{on} e il periodo T, si nota che la tensione media è direttamente proporzionale al livello alto, con un fattore di proporzionalità δ, $\overline{v_O}(t)$ è quindi direttamente proporzionale al periodo in cui il segnale rimane alto.

Il modulatore di Arduino UNO R3, oltre ad avere una frequenza di modulazione di $500\ [Hz]$, ha, inoltre, una risoluzione del duty-cicle di 8 bit. Una risoluzione simile permette di avere 256 valori di δ, avendo quindi un passo tra un valore di duty-cicle e l'altro di circa $7.8\ [\mu s]$.

5 Sensore di temperatura

Per conosce la temperatura interna, ed avere quindi retroazione, ho bisogno di un sensore di temperatura. Sul mercato sono disponibili molti trasduttori di questo tipo, con coefficiente di temperatura diversi e con range di funzionamento diversi; quello scelto presenta un range adeguato alle temperature di esercizio di un frigorifero comune ed un coefficiente di temperatura abbastanza alto.

5.1 AD22100A

Il trasduttore scelto è AD22100A di Analog Devices per due motivi: un coefficiente di temperatura più elevato rispetto ad altri componenti e per il suo range di funzionamento. Di seguito vengono riportati alcuni dati forniti dal costruttore:

	min	MAX
Temperatura di funzionamento	$-40°C$	$+85°C$
Tensione di Alimentazione V_{CC}	4 $[V]$	6.5 $[V]$
Coefficiente di temperatura	22.5 $[mV/°C]$	
Funzione di Trasferimento V_{CC}: tensione di alimentazione	$V_{out} = \left(\frac{V_{CC}}{5\ [V]}\right)\left(V_{0°C} + 22.5\ \left[\frac{mV}{°C}\right] \cdot T[°C]\right)$, $V_{0°C} = 1.375\ [V]$	
$V_{out}(T_{opt}), V_{CC} = 5[V]$	$(-15°C)$: 1.0375 $[V]$	$(+40°C)$: 2.275 $[V]$

Tab. 5.1 - Dati d'interesse relativi al sensore AD22100A forniti dal datasheet[14]

5.2 Adattamento segnale

Dato il ristretto range di funzionamento del sistema, il trasduttore scelto fornisce una maggiore variazione di potenziale rispetto alle alternative di mercato, in quanto l'output di tensione varia di 1.2375 $[V]$, garantendo quindi una buona risoluzione. Gli altri componenti presenti su mercato hanno un coefficiente di temperatura pari a 10 $[mV/°C]$ comportando una variazione di tensione in uscita di 0.65 $[V]$ o 0.6 $[V]$. Anche usando gli altri trasduttori con una tensione analogica di riferimento interna ad Arduino UNO pari a 1.1 $[V]$, solo poco più del 50% dei valori leggibili dalla porta analogica vengono utilizzati.

Con il componente scelto, e abbassando il segnale in uscita, è possibile ridurre la percentuale di valori non letti. Sarebbe quindi ottimale abbassare la tensione di 1 $[V]$.

Per abbassare la tensione ho usato un amplificatore operazionale in configurazione differenziale: sottraendo 0.9 $[V]$ dal segnale in uscita dal sensore e utilizzando un guadagno unitario, si ha che il segnale in uscita dall'operazionale è compreso $v_o \in [0.1375, 1.575]$ $[V]$.

Arduino UNO permette di impostare una tensione di riferimento per l'ADC tramite il pin AREF, sul quale imporrò una tensione $V_{ref} = 1.7\ [V]$.

5.2.1 Tensione di riferimento

Per applicare V_{ref} su AREF ho usato un partitore di tensione tra la resistenza interna di AREF in parallelo al trimmer Trm_{Ref_2} e la resistenza R_{Ref_1}, come mostrato in Fig. 5.1.

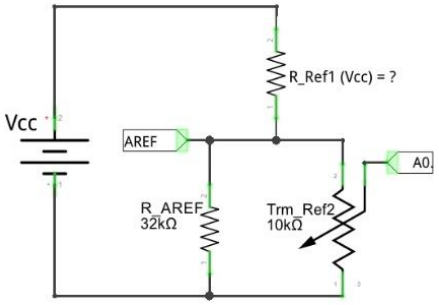

Fig. 5.1 - Circuito equivalente AREF (Fritzing)

Ho scelto di utilizzare un trimmer in parallelo ad AREF, poiché così è possibile regolare la temperatura voluta all'interno del frigo in modo più semplice, in quanto la tensione massima leggibile dal trimmer è pari alla tensione di riferimento, mentre la minima è $0\ [V]$.

Per scegliere la resistenza con cui fare il partitore in modo che abbia $1.7\ [V]$ di riferimento ho dovuto quindi considerare il parallelo tra R_{AREF} ed il trimmer (come riportato in Fig. 5.1):

$$R_{Ref_{eq}} = R_{AREF} || Trm_{Ref_2} = 7.62\ [k\Omega]$$

Da cui si ricava R_{Ref_1}:

$$R_{Ref_1} = R_{Ref_{eq}} \frac{V_{CC} - V_{ref}}{V_{ref}} \Bigg|_{V_{CC}=5[V]}^{V_{CC}=3.3\ [V]} \begin{matrix} \nearrow = 7.17\ [k\Omega] \\ \searrow = 14.79\ [k\Omega] \end{matrix}$$

Non disponendo di valori corrispondenti per R_{Ref_1} ho utilizzato un trimmer in modo da essere più preciso con il valore voluto e quindi avere un potenziale su AREF pari a $1.7\ [V]$ effettivi.

5.2.2 Amplificazione segnale

Ho utilizzato un amplificatore in configurazione differenziale per diminuire la tensione in uscita dal sensore di circa $900\ [mV]$:

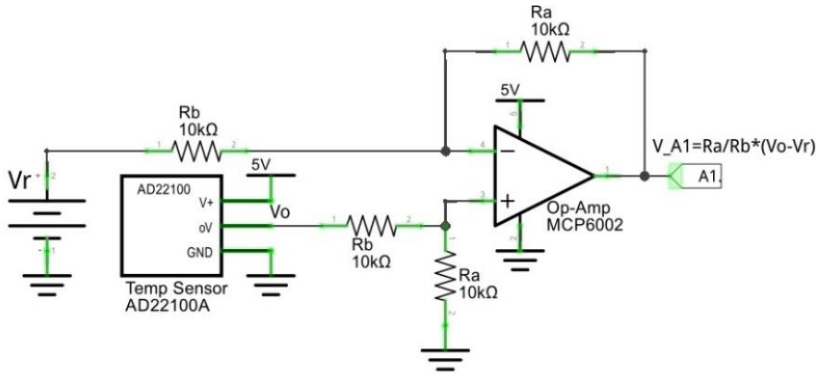

Fig. 5.2 - Circuito di condizionamento del sensore AD22100A (Fritzing)

Scegliendo $R_b = R_a$ e applicando la sovrapposizione degli effetti, spegnendo V_r:

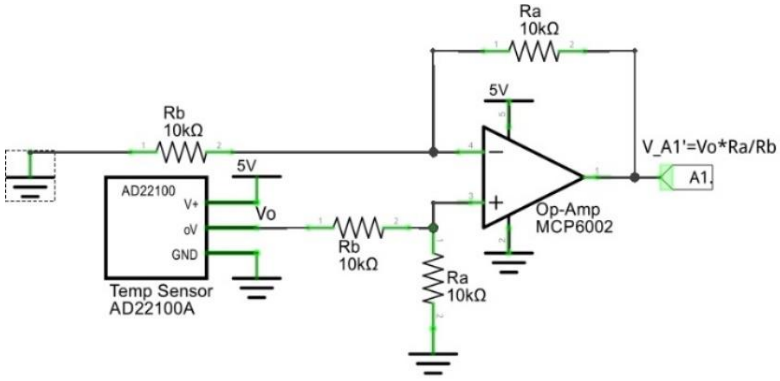

Fig. 5.3 - Sovrapposizione degli effetti: attivo solo V_o (Fritzing)

La tensione all'ingresso non invertente dell'OPAMP è il partitore di tensione di V_o rispetto a R_a, mentre la tensione all'ingresso invertente è il partitore di tensione rispetto a R_b di V_o, infine, gli ingressi sono in massa virtuale, quindi hanno lo stesso potenziale:

$$\begin{cases} V_- = V'_{in_{A1}} \dfrac{R_b}{R_a + R_b} \\ V_+ = V_o \dfrac{R_a}{R_a + R_b} \\ V_- = V_+ \end{cases} \Longrightarrow V'_{in_{A1}} = V_o \frac{R_a}{R_a + R_b} \frac{R_a + R_b}{R_b} = V_o \frac{R_a}{R_b} = V_o$$

Mentre spegnendo il sensore e lasciando acceso V_r:

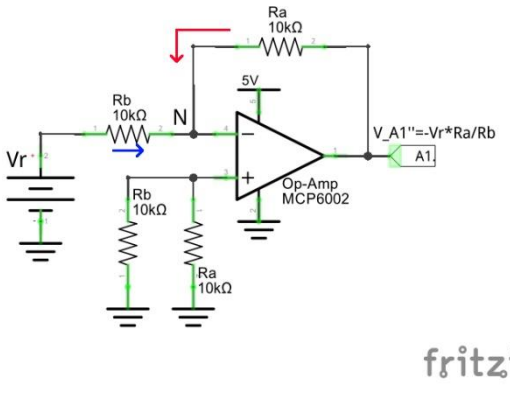

Fig. 5.4 - Sovrapposizione degli effetti: attivo solo V_r (Fritzing)

Al nodo N entrano due correnti: una passante per R_b (in blu) e l'altra passante per R_a (in rosso), inoltre, non essendoci correnti che scorrono nelle resistenze collegate all'ingresso non invertente il potenziale di questo è $0\ [V]$, allora la tensione all'ingresso invertente è anch'essa nulla, il circuito diventa quindi un amplificatore in configurazione invertente:

$$\begin{cases} I_a = \dfrac{V''_{in_{A1}} - V_-}{R_a} = \dfrac{V''_{in_{A1}}}{R_a} \\ I_b = \dfrac{V_r - V_-}{R_b} = \dfrac{V_r}{R_b} \\ I_a + I_b = 0\ [A] \\ V_- = V_+ = 0\ [V] \end{cases} \Longrightarrow V''_{in_{A1}} = -V_r \frac{R_a}{R_b} = -V_r$$

Sommando i contributi della sovrapposizione degli effetti ottengo una tensione sull'ingresso analogico A1:

$$V_{in_{A1}} = V'_{in_{A1}} + V''_{in_{A1}} = \frac{R_a}{R_b}(V_o - V_r) = V_o - V_r$$

Come amplificatore operazionale ho scelto MCP6002 in quanto è un operazionale Rail-to-Rail. Un amplificatore Rail-to-Rail è un amplificatore che permette di avere quasi le stesse tensioni di alimentazione. Inoltre, l'amplificatore scelto presenta una tensione di offset molto piccola (compresa tra $-7[mV]$ e $+7\ [mV]$). Riporto in Tab. 5.2 alcuni dati di interesse presi dalla documentazione fornita dal costruttore:

	min	MAX
Tensione di alimentazione positiva: V_{DD}	$1.8\ [V]$	$5.5\ [V]$
Tensione di Offset in ingresso: V_{os}	$-7\ [mV]$	$+7[mV]$
Corrente di Bias in ingresso: I_B	$-1\ [pA]$	$+1\ [pA]$
Swing di tensione in uscita:	$V_{SS} + 25\ [mV]$	$V_{DD} - 25\ [mV]$
Slew Rate	$0.6\ \left[\frac{V}{\mu s}\right]$	

Tab. 5.2 - Dati di interesse relativi all'amplificatore operazionale MCP6002 forniti dal datasheet[15]

Per verificare l'affidabilità del componente scelto analizzo come varia la caratteristica della tensione su A1 considerando anche le non idealità dell'Op-Amp, ignorando però gli effetti delle correnti di Bias in quanto molto piccole. Considerando la tensione di offset, e applicando la sovrapposizione degli effetti spegnendo V_r e V_o, si ottiene un altro contributo sommato ai due precedenti:

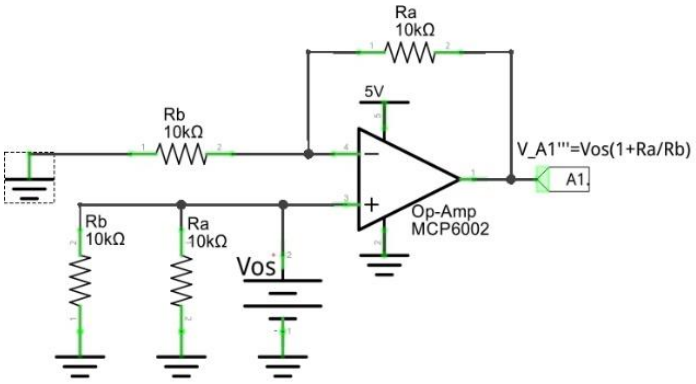

Fig. 5.5 - Sovrapposizione degli effetti: attivo solo V_{os} (Fritzing)

Come si può notare il circuito assume una configurazione non invertente rispetto a V_{os} :

$$V'''_{in_{A1}} = V_{os}\left(1 + \frac{R_a}{R_b}\right) = 2V_{os}$$

Sommando quest'ultimo contributo alla formulazione precedente:

$$V_{in_{A1}} = V'_{in_{A1}} + V''_{in_{A1}} + V'''_{in_{A1}} = \frac{R_a}{R_b}(V_o - V_r) + V_{os}\left(1 + \frac{R_a}{R_b}\right) = V_o - V_r + 2V_{os}$$

Che può comportare una variazione del segnale di uscita di $\pm 14\ [mV]$, che si traducono in una variazione di $\pm 0.6\bar{2}°C$ sulla misura della temperatura. Per verificare quale fosse la tensione di offset effettiva del componente in mio possesso ho configurato ogni lato come un amplificatore non invertente, mostrato in Fig. 5.6 (indicando i valori effettivi delle resistenze utilizzate).

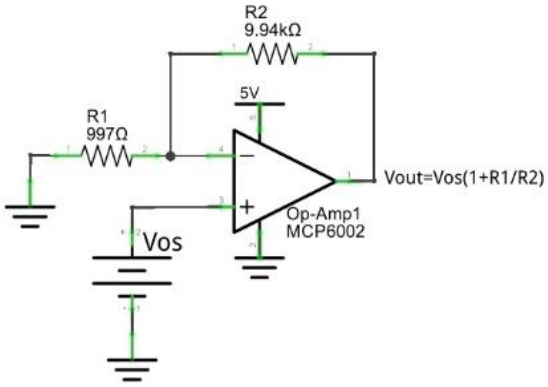

Fig. 5.6 - Amplificatore non invertente di V_{os} (Fritzing)

Il componente ha due amplificatori, uno per lato, e quindi ho due valori di V_{os}:

$$V_{out_A} = 7.2\ [mV] = V_{os_A}\left(1 + \frac{R_1}{R_2}\right) \Leftrightarrow V_{os_A} = V_{out_A}\frac{R_2}{R_1 + R_2} = 656.34\ [\mu V]$$

$$V_{out_B} = 17.9\ [mV] = V_{os_B}\left(1 + \frac{R_1}{R_2}\right) \Leftrightarrow V_{os_B} = V_{out_B}\frac{R_2}{R_1 + R_2} = 1.632\ [mV]$$

Nel circuito di adattamento del sensore, V_{os} comporta una variazione del segnale in ingresso dell'ADC di circa $+1.3\ [mV]$ (equivalente a $+0.06°C$) per il lato A e circa $3.3\ [mV]$ (equivalente a $0.15\ °C$) per il lato B. Usando il lato A posso trascurare l'effetto della tensione di offset.

Arduino UNO R3 non permette di generare una tensione a scelta, per realizzare V_r uso un partitore di tensione su due resistenze:

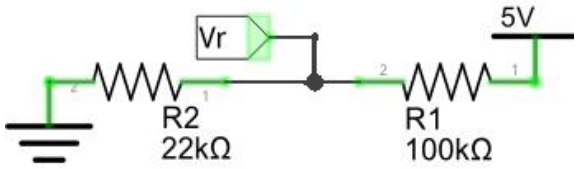

Fig. 5.7 - Partitore di tensione tra R_1 e R_2 per generare V_r (Fritzing)

$$V_r = V_{CC}\frac{R_2}{R_1 + R_2}$$

Conoscendo V_r, V_{CC}, e utilizzando un valore commerciale di R_1 devo da avvicinarmi il più possibile ad un valore commerciale di R_2. Usando $R_1 = 100\ [k\Omega]$ ottengo $R_2 = 21.95\ [k\Omega]$, il valore commerciale più vicino per R_2 è $22\ [k\Omega]$.

Come riportato nella scheda tecnica del sensore, la formula per ricavare la tensione in uscita dal sensore conoscendo la temperatura è:

$$V_{out} = \left(\frac{V_{CC}}{5\ [V]}\right)\left(V_{0°C} + 22.5\ \left[\frac{mV}{°C}\right] \cdot T[°C]\right)$$

Poiché i $5\ [V]$ forniti da Arduino non sono effettivamente tali, la temperatura misurata dal sensore sarà:

$$T[°C] = \frac{V_{out}\left(\frac{5[V]}{V_{CC}}\right) - V_{0°C}}{22.5\ \left[\frac{mV}{°C}\right]}$$

Dove $V_{CC} = 4.97\ [V]$. La tensione, però, letta dall'ingresso analogico A1 (vale a dire in uscita dall'amplificatore differenziale) è:

$$V_{in_{A1}} = V_{out} - V_r \Leftrightarrow V_{out} = V_{in_{A1}} + V_r$$

Quindi conoscendo la tensione letta da Arduino e la tensione in uscita dal partitore è possibile calcolare la temperatura tramite:

$$T[°C] = \frac{\left(\frac{5[V]}{V_{CC}}\right)(V_{in_{A1}} + V_r) - V_{0°C}}{22.5\ \left[\frac{mV}{°C}\right]}$$

6 Ventola

Sfruttando il comportamento della cella di Peltier, è possibile abbassare la temperatura del lato freddo della cella abbassando la temperatura del lato caldo, poiché la differenza di temperatura tra i due lati è mantenuta costante. Per poter raffreddare non basta utilizzare un dissipatore, ho bisogno di rimuovere l'aria calda e farne entrare di fredda; per fare ciò ho usato una ventola, la quale aspira aria dalle prese ai lati, che passa nel dissipatore uscendo da sopra.

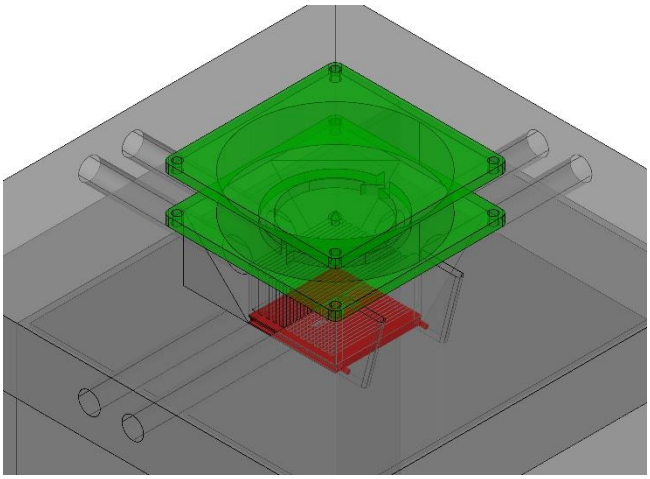

Fig. 6.1 - Render parte alta del frigo, ritraente: ventola in verde, cella in rosso, dissipatore in grigio scuro e polistirolo in grigio chiaro

La ventola scelta per questo scopo ha quattro pin, due per l'alimentazione ($+12\ [V]$ e $0\ [V]$), un ingresso PWM per regolarne la velocità e l'ultimo è l'uscita del tachimetro interno. Lo scopo iniziale era quello di comandare la velocità della ventola in retroazione e, a seconda della velocità e della temperatura interna, imporre la velocità adeguata alla situazione.

6.1 Adattamento segnale tachimetro

Il segnale del tachimetro non è direttamente interfacciabile con Arduino, in quanto la maggior parte del segnale è negativo, non ha la forma di un'onda quadra e la parte positiva del segnale raggiunge appena i $70\ [mV]$. Inoltre, a bassa velocità il segnale presenta un disturbo non trascurabile.

Fig. 6.2 - Uscita tachimetro ad alta velocità

Fig. 6.3 - Uscita tachimetro a bassa velocità

Usando uno stadio amplificatore non invertente con guadagno $A = 1 + \frac{22\ [k\Omega]}{1\ [k\Omega]} = 23$, il picco di segnale raggiunge una grandezza massima di circa 1.5 $[V]$, grandezza adeguata a essere comparata.

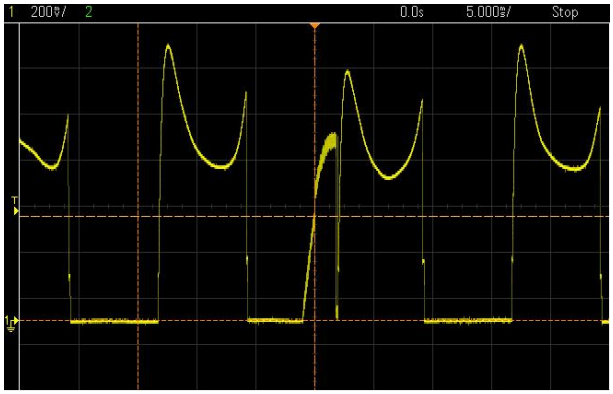

Fig. 6.4 - Segnale tachimetro amplificato ad alta velocità

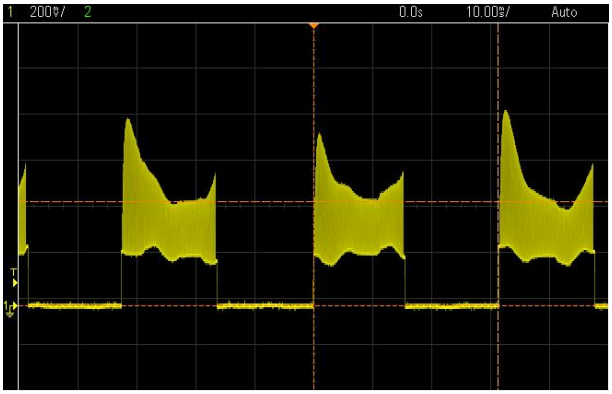

Fig. 6.5 - Segnale tachimetro amplificato a bassa velocità

Lo stadio amplificatore però amplifica anche i disturbi del segnale a bassa velocità. Per ridurre tali disturbi ho usato un filtro passa-basso del primo ordine RC, con frequenza di taglio almeno di un ordine di grandezza superiore alla frequenza massima. Il tachimetro genera due impulsi per giro, in quanto presenta due sonde di Hall, dal costruttore so che la velocità massima della ventola è di 2000 $[rpm]$, quindi ho una frequenza degli impulsi pari a $66.\overline{6}\ [Hz]$: usando un condensatore da 10 $[nF]$ ed una resistenza da 4.7 $[k\Omega]$ ottengo una frequenza di taglio $F_{cut} = 3.4\ [kHz]$.

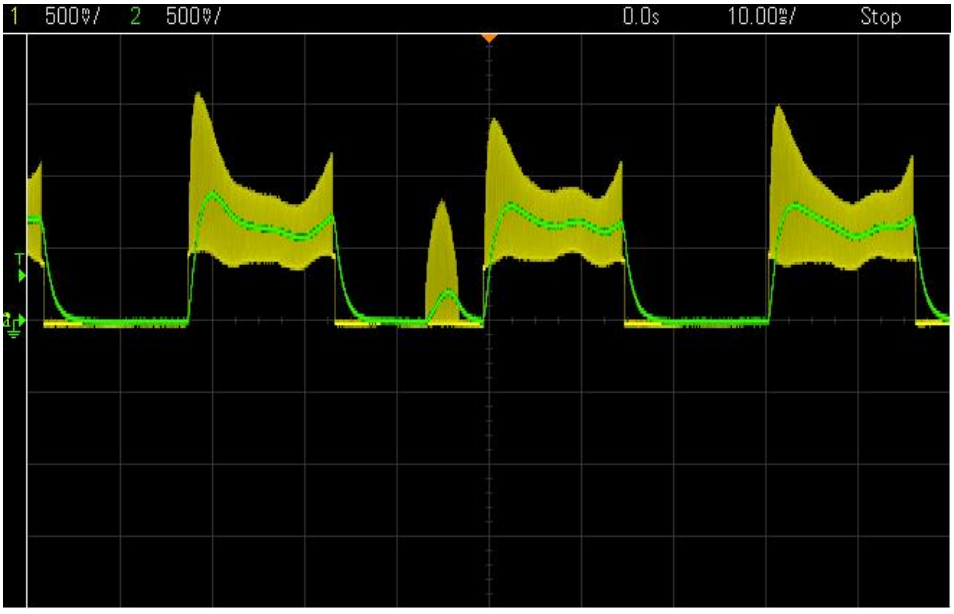

Fig. 6.6 - Segnale tachimetro a bassa velocità, amplificato e filtrato

Nella Fig. 6.6 sono riportate le forme d'onda in ingresso al filtro (in giallo) e in uscita di questo (in verde). Come si può notare il segnale ha una forma meglio definita a bassa velocità, però l'ampiezza massima che raggiunge è appena di 750 $[mV]$: il segnale non è ancora abbastanza grande per poter essere letto dal pin digitale di Arduino. Utilizzo allora un altro operazionale, stavolta però come comparatore, in modo che quando la tensione è sopra la soglia di confronto il segnale sia alto, sia

invece basso nel caso opposto. Come soglia di confronto (V_{comp}) ho utilizzato il partitore tra due resistenze aventi un ordine di grandezza come differenza:

$$V_{comp} = V_{CC} \cdot \frac{R_5}{R_4 + R_5}, \qquad V_{CC} = 5[V], \qquad R_5 = 4.7\ [k\Omega], \qquad R_4 = 10R_5$$

In questo modo si ottiene $V_{comp} = 454.5\ [mV]$, soglia più che sufficiente per poter comparare il segnale. Ottenendo in uscita una forma d'onda quadra.

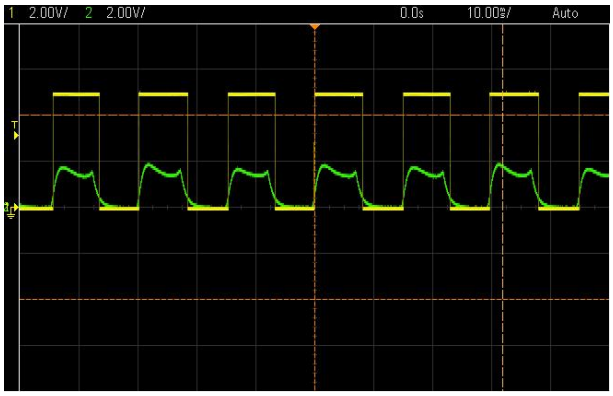

Fig. 6.7 - Ingresso e uscita del comparatore ad alta velocità

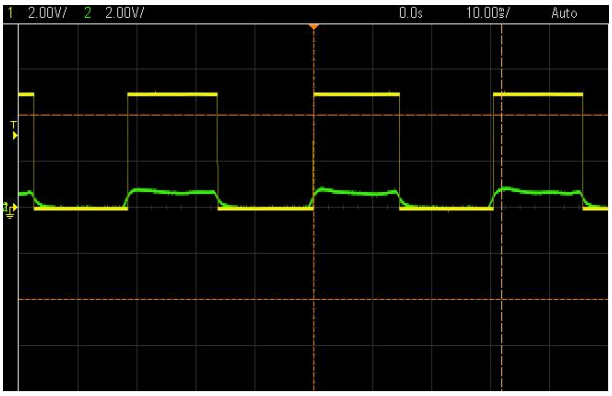

Fig. 6.8 - Ingresso e uscita del comparatore a bassa velocità

Nella Fig. 6.7 e Fig. 6.8 sono rappresentate le forme d'onda in uscita dal comparatore (in giallo) e il segnale in uscita dal filtro (in verde). Il segnale risulta un'onda quadra priva di disturbi, con ampiezza di 4.925 $[V]$, che supera il valore minimo per poter essere riconosciuto da Arduino come valore logico "alto".

Il circuito finale che adatta il segnale del tachimetro della ventola risulta quindi essere:

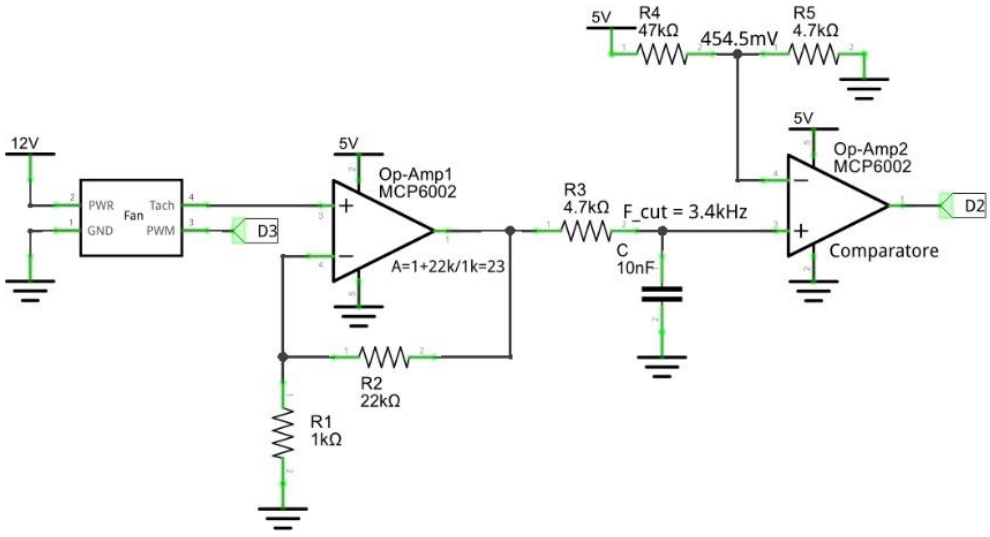

Fig. 6.9 - Circuito adattamento segnale ventola (Fritzing)

6.2 Interferenze tra segnali

Un problema di cui non avevo tenuto conto è l'interferenza di segnale tra la modulazione a larghezza d'impulso di Arduino e il segnale del tachimetro.

Fig. 6.10 - Segnale uscita circuito adattamento

La Fig. 6.10 ritrae il segnale in uscita dall'adattamento del tachimetro (pin digitale 2 di Arduino): come si può notare, il segnale non presenta fronti di salita o di discesa marcati, a differenza di prima. Questo perché il segnale in ingresso all'adattamento è ben diverso:

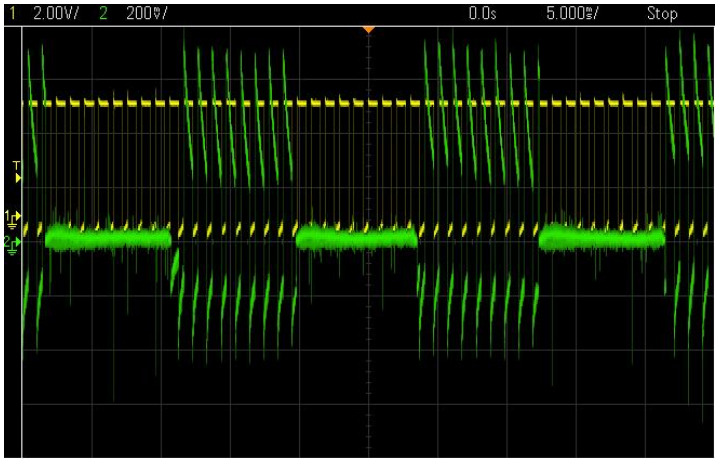

Fig. 6.11 - Segnale PWM e uscita tachimetro

In verde abbiamo il segnale in uscita dal tachimetro della ventola (prima dell'adattamento del segnale), in giallo abbiamo invece l'uscita analogica di Arduino (uscita PWM) con un duty-cicle di circa 70%.

Questa interferenza si ha solo per duty-cycle diversi da 0% (segnale logico "basso") o 100% (segnale logico "alto"). Ciò complica l'adattamento e la lettura del segnale. Ho quindi scelto di non retroazionare la velocità della ventola, in quanto va ad appesantire il programma, inoltre, risulta più semplice usare la temperatura interna come variabile di riferimento per la regolazione della ventola.

7 Controllo PID

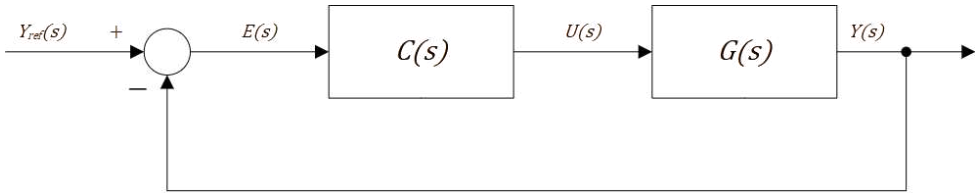

Fig. 7.1 - Semplice schema di retroazione

In un controllo in retroazione l'azione di controllo ($C(s)$) può essere progettata ad hoc se si conosce la funzione di trasferimento del sistema da controllare ($G(s)$). Nel mio caso non conosco $G(s)$, per cui utilizzo un controllo PID. Il quale permette di impostare i parametri tramite "trial & error".

7.1 Principio di funzionamento

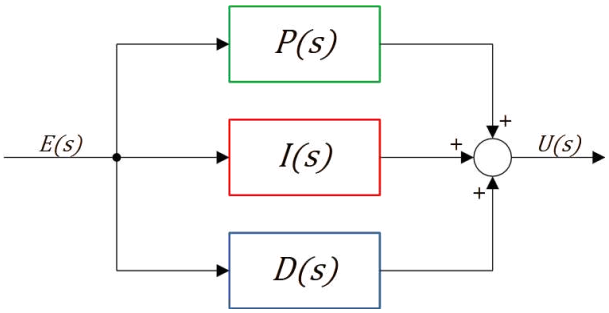

Fig. 7.2 - Blocco di controllo PID

Il controllo PID è composto da tre azioni che intervengono sull'errore. Un'azione proporzionale, una integrativa ed una derivativa che poi sono sommate.

Il microcontrollore però non può tenere conto di ogni istante del valore dell'errore o dell'uscita, per cui si è costretti a ricorrere alla discretizzazione. Per rendere più semplice il processo e quindi più semplice a livello di programma utilizzerò la discretizzazione di Eulero, procedimento che permette di convertire un filtro analogico in uno digitale, mettendo in relazione la trasformata Zeta con la trasformata di Laplace, tramite la seguente relazione:

$$s = \frac{1 - z^{-1}}{T} = 1 - z^{-1}, \qquad T \coloneqq \text{periodo di campionamento} = 1[\text{s}]$$

7.1.1 Azione proporzionale

L'azione proporzionale è il controllo più semplice visto che tale azione incrementa l'errore di una costante k_P. Con $P(s) = k_P$, si ha che l'azione proporzionale $U_P(s) = E(s) \cdot k_P$, che nel dominio del tempo diventa $u_P(t) = k_P \cdot e(t)$. Tale azione però non è sufficiente a portare l'uscita a regime: aumentando k_P si possono ridurre i tempi di salita ma si diminuisce la stabilità del sistema.

Discretizzando il filtro si ha:

$$U_P(s) = E(s) \cdot k_P \xRightarrow{s=1-z^{-1}} U_P(z) = E(z) \cdot k_P \xrightarrow{\mathcal{Z}^{-1}} u_P(k) = k_P \cdot e(k)$$

Implementare questo filtro nel programma risulta piuttosto semplice, di fatto basta moltiplicare l'errore per una costante k_P (definita come macro): (`uP = kP*err`).

7.1.2 Azione integrativa

L'azione integrativa, invece, presenta una funzione nel dominio di Laplace del tipo $I(s) = k_I/s$, comportando $U_I(s) = k_I \cdot E(s)/s$, che nel dominio del tempo si traduce con $u_I(s) = k_I \int_0^t e(\tau)d\tau$. Quest'azione diminuisce ancora di più i tempi di salita, annullando l'errore a regime, ma diminuisce la stabilità del sistema poiché comporta sovraelongazioni dell'uscita.

Discretizzando il filtro si ha:

$$U_I(s) = E(s) \cdot \frac{k_I}{s} \xRightarrow{s=1-z^{-1}} U_I(z) = E(z) \cdot \frac{k_I}{1-z^{-1}} \Leftrightarrow U_I(z) = E(z) \cdot k_I + U_I(z)z^{-1}$$

$$\mathbb{Z}^{-1}[U_I(z)] = u_I(k) = k_I \cdot e(k) + u_I(k-1)$$

L'implementazione del filtro integrativo richiede l'utilizzo di una variabile di appoggio che tiene conto dell'azione precedente dell'integrazione (`uI = kI*err + uI`).

7.1.3 Azione derivativa

Quest'azione serve a stabilizzare il sistema: presenta una funzione nel dominio di Laplace del tipo $D(s) = k_D \cdot s$, comportando un'azione $U_D(s) = k_D \cdot s\, E(s)$ che nel tempo diventa una derivata $u_D(t) = k_D \frac{de(t)}{dt}$. Come detto, quest'azione permette di ridurre le sovraelongazioni, aumentando anche la stabilità del sistema, influisce però sul tempo di risposta.

Discretizzando il filtro si ha:

$$U_D(s) = k_D \cdot sE(s) \xRightarrow{s=1-z^{-1}} U_D(z) = E(z)k_D(1-z^{-1}) \xrightarrow{\mathcal{Z}^{-1}} u_D(k) = k_D\big(e(k) - e(k-1)\big)$$

Anche l'implementazione di questo filtro richiede l'utilizzo di una variabile di appoggio che tiene conto dell'errore precedente (`uD = kD*(err - errOld)`).

8 Schermo OLED

Per conoscere la temperatura interna del frigo senza ricorrere alla comunicazione seriale con il PC, utilizzo un display OLED di piccole dimensioni (1.3 pollici, circa 3.3 cm) in modo che non occupi troppo spazio. Il display scelto utilizza il protocollo di comunicazione I^2C, protocollo vantaggioso perché presenta pochi collegamenti, è più veloce di un eventuale protocollo SPI e non richiede di impostare una frequenza di comunicazione a differenza della comunicazione seriale. Il costruttore dello schermo ha messo disponibili delle librerie per Arduino che ne permettono un utilizzo più semplice ed efficacie.

8.1 Comunicazione I^2C

La comunicazione I^2C è una comunicazione sincrona, in quanto i dati vengono trasmessi a frequenza di clock su un Bus condiviso da tutte le periferiche. Il Bus è composto da due linee, una che trasmette i dati (SDA) e l'altra è per il segnale di clock (SCL), quest'ultima è generata dal controllore e controlla quando i dati vengono spediti o letti.

Un messaggio è composto da pacchetti a 8 bit, dopo ogni pacchetto trasmesso vi è il bit di riconoscimento (Acknowledgement), il quale viene trasmesso dall'unità a cui è indirizzata la trasmissione, se tale bit non viene trasmesso il pacchetto dovrà essere rispedito. Per iniziare una comunicazione tra controllore e dispositivo si deve avere una condizione START (SCL alto e SDA basso); per terminare la comunicazione invece si ha invece una condizione di STOP (SCL alto e SDA basso). Il primo pacchetto a 8 bit contiene sempre l'indirizzo il dispositivo con cui il controllore vuole comunicare, composto da 7 bit (permettendo di comunicare fino a 128 dispositivi), e la condizione di scrittura dal controllore ("0") o di lettura ("1"); dal secondo pacchetto in poi vengono trasmessi i dati.

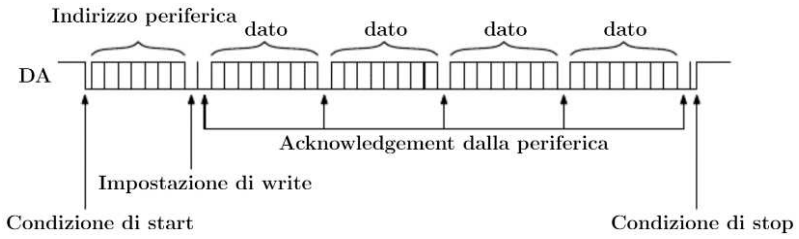

Fig. 8.1 - Scrittura di un messaggio dal microcontrollore verso una periferica nel bus[16]

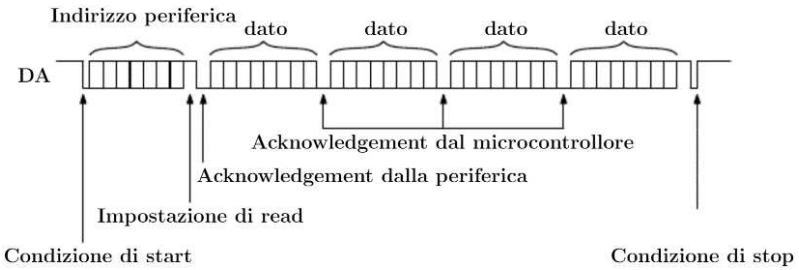

Fig. 8.2 - Lettura di dati da una periferica nel bus da parte del microcontrollore[17]

8.2 Librerie per display

Lo schermo da me utilizzato è compatibile con la libreria open-source (`Adafruit_SH110X.h`) creata da Adafruit[18]. Tale libreria è valida per i display monocromatici OLED basati sui driver SH110X. Questi display utilizzano il protocollo di comunicazione I^2C o SPI.

Questa libreria permette di creare oggetti per associarli all'indirizzo del display (se usa I^2C), o al pin digitale (se usa SPI), in modo da rendere più semplice, all'utente, la comunicazione tra Arduino e display. Di questa libreria solo 2 comandi sono vengono usati nel codice:

- Inizializzazione del display: quando inizializzo il display, devo passare larghezza e altezza del display, l'indirizzo della classe `Wire` e il pin di reset dello schermo.

  ```
  Adafruit_SH1106G display = Adafruit_SH1106G(SCREEN_WIDTH, SCREEN_HEIGHT,
  &Wire, OLED_RESET)
  ```

- Inizializzazione della comunicazione: questo comando va utilizzato all'interno del setup, specificando l'indirizzo del display (nel mio caso `0x3c`) e `true`, che va a resettare il display.

  ```
  display.begin(DEV_ID, true)
  ```

Inoltre, è necessario aver installato anche la libreria Adafruit GFX (`Adafruit_GFX.h`)[19], libreria grafica principale per tutti i display Adafruit, o che utilizzano i suoi driver, fornendo un insieme comune di primitive grafiche (punti, linee, testo, ecc.). Deve essere accoppiato ad una libreria specifica dell'hardware per ogni dispositivo di visualizzazione Adafruit (per gestire le funzioni a livello inferiore).

Infine, per la comunicazione I^2C è necessario installare la libreria `Wire.h`, che permette di usare tale protocollo di comunicazione.

9 Progetto

Per la realizzazione del frigo ho disegnato prima il circuito del sistema, poi ho proceduto alla realizzazione di un modello 3D della struttura per poter comprenderne le grandezze ed eventuali posizioni dei componenti. Una volta realizzati i modelli 3D, ho proceduto al montaggio del progetto, usando il polistirolo come isolante termico e compensato come struttura esterna dove poter fissare i vari componenti. Terminato il modello 3D e costruito il frigo, ho interfacciato i componenti ad Arduino ed effettuato i collegamenti all'alimentatore; infine, ho realizzato il firmware da caricare in Arduino UNO verificando il corretto funzionamento del frigorifero.

9.1 Progettazione 3D

Per la realizzazione del modello 3D ho usato AutoCAD, software con il quale ho familiarità:

1. Ho determinato prima di tutto il volume interno del frigorifero, in modo che potesse contenere 4 lattine da 500 $[ml]$. Considerando degli spazi tra le lattine, le pareti interne e la parete superiore, il volume interno è risultato di circa 4 $[l]$ $(14{,}5\ [cm] \times 14{,}5\ [cm] \times 19[cm])$. Come isolante termico ho scelto il polistirolo, in quanto leggero, ottimo isolante e facile da intagliare.

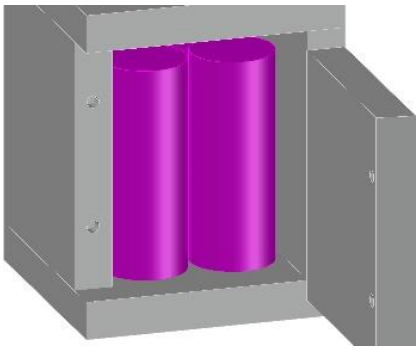

Fig. 9.1 - Ambiente interno, isolato dal polistirolo (bianco), con 4 lattine da 500 cl (magenta), con un lato aperto

2. Dopo aver realizzato l'interno, ho proceduto col posizionare l'elemento raffreddante sul lato superiore, intagliando il posto per la cella con relativo dissipatore e prese d'aria, come si può vedere dalla Fig. 6.1 del capitolo 6 (riportata nuovamente in Fig. 9.2).

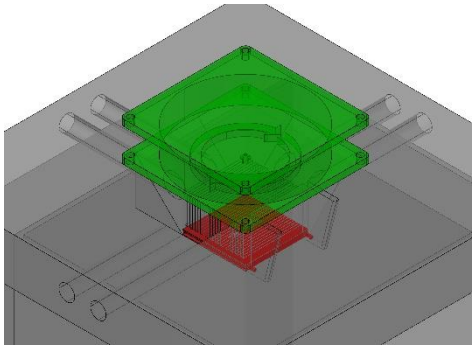

Fig. 9.2 - Prese d'aria nel polistirolo (grigio chiaro), ventola (verde), cella di Peltier (rosso) e dissipatore (grigio scuro)

3. Come struttura esterna più solida ho usato del compensato, che presenta una resistenza agli stress meccanici migliore rispetto al polistirolo, permettendomi, inoltre, di fissare con più sicurezza i componenti, come scheda Arduino, alimentatore, MOSFET e ventola.

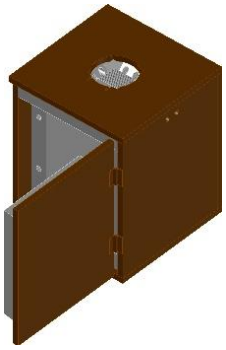

Fig. 9.3 - Struttura esterna di compensato (marrone) che circonda il polistirolo (bianco)

4. Il sensore di temperatura è stato posizionato sul medesimo lato della scheda Arduino, internamente però, dove è presente una breadboard comprendente lo schermo OLED e i circuiti di condizionamento del segnale del sensore.

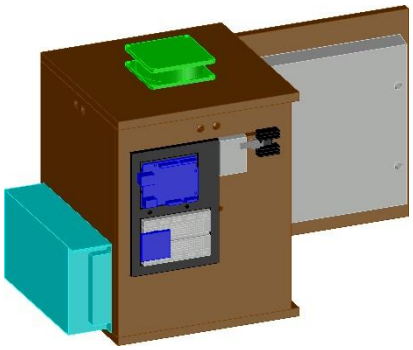

Fig. 9.4 - Componenti fissati al compensato: MOSFET (grigio scuro), Arduino UNO (blu), display (blu), breadboard (bianco), dissipatore MOSFET (nero), Alimentatore (ciano)

9.2 Circuito

In Fig. 9.5 viene riportato il disegno finale del circuito del sistema: a sinistra di Arduino UNO troviamo i componenti alimentati a 12 $[V]$, tra cui: ventola, MOSFET e cella di Peltier; a destra del controllore abbiamo invece i collegamenti di ciò che è già stato presentato, come: il partitore di tensione tra resistenza e trimmer, per avere 1.7 $[V]$ su AREF, l'amplificatore operazionale MCP6002 in configurazione differenziale, che effettua la sottrazione tra la tensione in uscita dal sensore AD22100A, la tensione in ingresso ad A3 ed infine il display OLED.

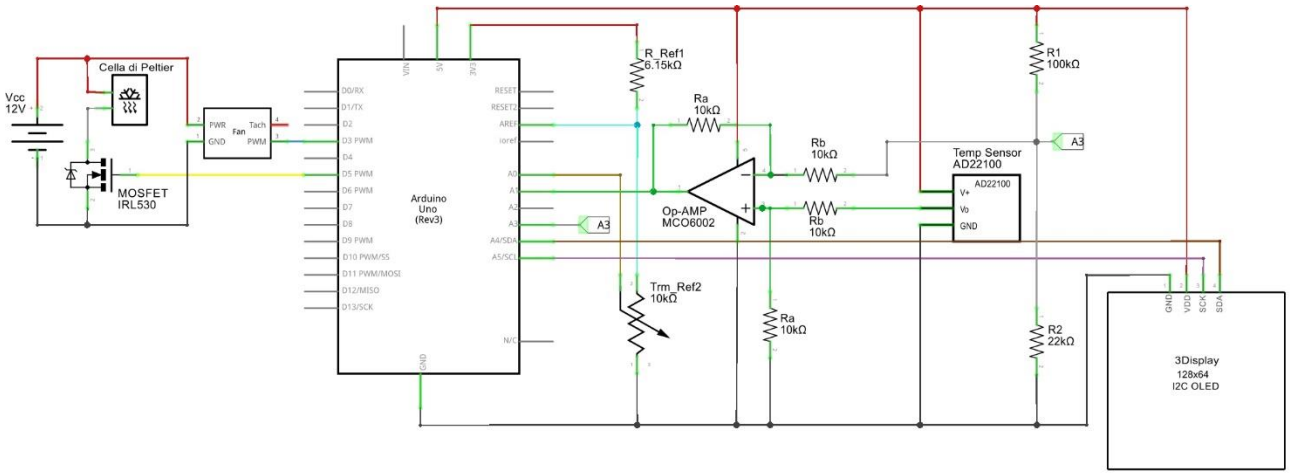

Fig. 9.5 - Circuito del sistema (Fritzing)

9.3 Funzionamento

In Fig. 9.6 viene riportato lo schema a blocchi del sistema, come già detto, tale sistema è retroazionato negativamente. I blocchi che compongono la catena chiusa sono rappresentazioni dei componenti utilizzati:

- Il controllore effettua il calcolo dell'errore ($E(s)$) e del controllo ($PID(s)$), tramite codice, che invia un segnale ($U(s)$) all'attuatore;
- Attuatore: è il sistema che deve essere controllato e che cambia la temperatura interna del frigo, composto dal MOSFET, che permette lo scorrere della corrente ($I(s)$), e cella di Peltier che abbassa la temperatura dell'ambiente interno ($T(s)$) rilevata dal sensore;
- Sensore: composto dal trasduttore di temperatura, che converte la temperatura ($T(s)$) in un segnale di tensione ($V(s)$), e dal condizionamento, che adattata la tensione e la converte nella temperatura misurata ($T_{mis}(s)$).

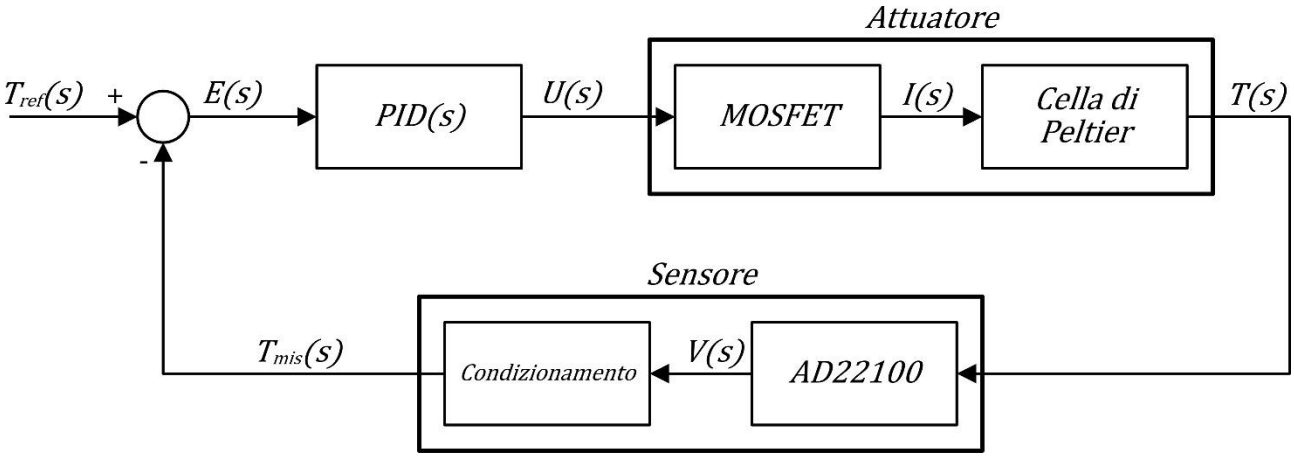

Fig. 9.6 - Schema a blocchi del sistema

9.4 Programma

Il firmware di per sé non è troppo complesso, buona parte del codice, riportato in Tab. 9.1, contiene principalmente inizializzazione di variabili e scrittura a display. Evidenzio le parti più importanti:

- Calcolo della temperatura (da riga 87 a 93): calcolo prima V_r, poi $V_{in_{A1}}$ e la temperatura usando la formula riportata a fine paragrafo 5.2.2; ho usato un filtro passa-basso di tipo IIR sulla variabile `Temp` per ridurre l'effetto di grandi variazioni del segnale dovute a disturbi.
- Mappatura della temperatura di riferimento (da riga 81 a 84 e 98): la funzione `mapTemp` si comporta come una funzione matematica lineare, dove il valore dall'ingresso analogico collegato al trimmer Trm_{Ref_2} rappresenta x e restituisce $f(x) = (T_{MAX} - T_{min})\frac{x}{1023} + T_{min}$.
- Controllo PID (da riga 100 a 108): si può notare che l'errore non è calcolato come riportato nello schema a blocchi ($Err = T_{ref} - Temp$), bensì è di segno opposto perché il riferimento è sempre più basso rispetto alla temperatura interna iniziale, avendo un'azione che può solo essere positiva, questa resterebbe sempre a 0.

```
1  #include <Wire.h>              //Importo la libreria per la comunicazione I2C
2  #include <Adafruit_GFX.h>      //Importo la libreria per le grafiche a display
3  #include <Adafruit_SH110X.h>   //Importo la libreria del display
4
5  //Definisco il display
6  #define  DEV_ID 0x3c         //Indirizzo del display
7  #define  SCREEN_WIDTH   128 //Larghezza display, in pixel
8  #define  SCREEN_HEIGHT  64  //Altezza display, in pixel
9  #define  OLED_RESET -1      //QT-PY / XIAO
10 #define  BLACK  0           //0 = LED off
11 #define  WHITE  1           //1 = LED on
12 #define  INVERT   2         //2 = Inverti stato LED
13 //Inizializzo il display
14 Adafruit_SH1106G display = Adafruit_SH1106G(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
15 OLED_RESET);
16
17 //Definisco i Parametri per calcolare la temperatura
18 #define  portTref A0    //Porta per leggere la temperatura impostata
19 #define  portTemp A1    //Porta del trasduttore di temperatura
20 #define  portVr   A3    //Porta per leggere la tensione Vr
21 #define  V_5v   4.98    //Tensione effettiva in uscita dai 5 [V]
22 #define  Tcoef  0.0225  //Coefficiente di temperatura 22.5 [mV/°C]
23 #define  IRR    0.5     //Costante filtro IIR
24 #define  V_0C   1.375   //Tensione in uscita dal trasduttore a 0°C
25 #define  Vref   1.7     //Tensione applicata ad AREF
26
27 #define  T_min  5       //Temperatura interna minima;
28 #define  T_MAX  15      //Temperatura interna MASSIMA;
29
30 #define  pelPWM 5       //Pin PWM cella Peltier
31 #define  fanPWM 3       //Pin PWM ventola
32
33 float Vr = 0.9;     //Tensione Vr (ideale)
```

```
34 float Vin = 0.0;     //Tensione in ingresso all'ADC
35 float Temp = 0.0;    //Temperatura misurata dal trasduttore
36 float T_old = 0.0;   //Valore di temperatura precedentemente misurato
37 float T_ref = 0.0;   //Valore di temperatura impostata tramite trimmer
38
39 float err = 0.0;     //Errore := Tref - Temp
40 float errOld = 0.0; //Errore Precedente
41 float uP = 0.0;      //Azione Proporzionale
42 float uD = 0.0;      //Azione Derivativa
43 float uI = 0.0;      //Azione Integrativa
44 int PID = 0;         //Azione PID
45 #define kP  7.5     //Costante Proporzionale
46 #define kI  0.1     //Costante Integrativa
47 #define kD  0.1     //Costante Derivativa
48
49 void setup()
50 { display.begin(DEV_ID, true);  //Inizio la comunicazione con il display
51   display.clearDisplay();       //Pulisce lo schermo
52   display.display();            //Esegue i comandi a display
53
54   analogReference(EXTERNAL);    //Imposto 1.7 [V] come tensione di riferimento
55
56   //Scrivo a display una sola volta "Temperatura interna", "Temperatura imposta"
57     e “Azione PID"
58   display.clearDisplay();
59   display.setTextSize(1);
60   display.setTextColor(WHITE);
61   display.setCursor(4,0);
62   display.print("Temperatura interna:");
63   display.setCursor(4,20);
64   display.print("Temperatura imposta:");
65   display.setCursor(31,40);
66   display.print("Azione PID:");
67   display.display();
68
69   pinMode(pelPWM, OUTPUT);
70   analogWrite(pelPWM, 0);
71   pinMode(fanPWM, OUTPUT);
72   analogWrite(fanPWM, 100);
73
74   Vr = analogRead(portVr)*Vref/1023;    //Imposto il valore di Vr effettivo
75   Vin = analogRead(portTemp)*Vref/1023; //Calcolo il valore di tensione
76   //Uso la formula inversa del trasduttore per trovare la temperatura
77   T_old = ((Vin + Vr)*5/V_5v - V_0C)/Tcoef;
78 }
79
80 //Funzione che mappa il valore letto dall’ADC ad un range di temperature
81 float mapTemp(int x)
82 { float y = x;
83   return (T_MAX - T_min)*y/1023 + T_min;
84 }
85
86 void loop()
87 { Vr = analogRead(portVr)*Vref/1023;    //Imposto il valore di Vr effettivo
88   Vin = analogRead(portTemp)*Vref/1023; //Calcolo il valore di tensione
89   //Uso la formula inversa del trasduttore per trovare la temperatura
90   Temp = ((Vin + Vr)*5/V_5v - V_0C)/Tcoef;
91
```

```
  //Filtro IIR, per filtrare le grandi variazioni di segnale dovute a disturbi
  Temp = IRR*Temp + (1-IRR)*T_old;
  //Associo l'attuale valore della temperatura al valore precedente
  T_old = Temp;

  //Mappo il valore letto dall'ADC ad un range di temperatura
  T_ref = mapTemp(analogRead(portTref));

  err = Temp - T_ref;    //Calcolo l'errore
  uP = kP*err;           //Calcolo l'azione Proporzionale
  uI = kI*err + uI;      //Calcolo l'azione Integrativa
  uD = kD*(err - errOld);//Calcolo l'azione Derivativa
  PID = uP + uI + uD;    //Sommo le azioni P.I.D.
  errOld = err;          //Salvo l'attuale valore di errore per il prossimo ciclo

  if(PID>255) PID = 255; //Verifico che il controllo PID non vada oltre 2^8-1
  if(PID<0)   PID = 0;   //Verifico che il controllo PID non scenda oltre 0

  //Imposto i duty-cicle dei PWM collegati al gate del MOSFET e alla ventola
  analogWrite(pelPWM, PID);
  analogWrite(fanPWM, PID);

//Per cancellare i dati precedenti disegno dei rettangoli neri al loro posto
  display.fillRect(0, 10, SCREEN_WIDTH, 8, BLACK);
  display.fillRect(0, 30, SCREEN_WIDTH, 8, BLACK);
  display.fillRect(0, 50, SCREEN_WIDTH, 8, BLACK);
  display.setTextSize(1);
  //Stampo a Display il valore della temperatura interna con 1 cifra decimali
  display.setCursor(16,10);
  display.print(String(String(Temp,1)+(char)247+"C"));
  //Stampo a Display il valore della temperatura imposta con 1 cifra decimali
  display.setCursor(16,30);
  display.print(String(String(T_ref,1)+(char)247+"C"));
  //Stampo a Display il valore dell'azione PID calcolato
  display.setCursor(16,50);
  display.print(PID);
  display.display();
  delay(800);
}

Lo sketch usa 17182 byte (53%) dello spazio disponibile per i programmi.
Le variabili globali usano 499 byte (24%) di memoria dinamica, lasciando altri
1549 byte liberi per le variabili locali.
```

Tab. 9.1 - Programma caricato in Arduino

10 Conclusioni

Di seguito riporto le prove effettuate per verificare il corretto funzionamento del programma e del sistema:

- Appena avviato la temperatura interna dal frigo è pari a quella ambientale (26°C); dopo pochi minuti il controllo PID arriva già al suo massimo (255), ma la temperatura non è ancora scesa abbastanza. Noto che a regime la temperatura interna non si è avvicinata molto a quella di riferimento (12,3 °C), di fatto è scesa di appena 3.5°C (22.5°C).
- Per verificare che il controllo funzioni ho messo del ghiaccio all'interno del frigo: inizialmente la temperatura ha una rapida discesa ed il controllo PID, dopo alcuni secondi, è poco più del 55% (circa 145), questo perché la velocità di discesa è molto alta e l'azione derivativa prevale sul controllo; quando la temperatura scende con meno rapidità il controllo aumenta, la parte integrativa prevale quindi sul controllo aumentandone l'effetto. Dopo quasi un quarto d'ora la temperatura interna raggiunge il riferimento (12,3°C), ma l'azione di controllo è ancora al suo massimo, rimanendo tale per qualche minuto ancora, questo, probabilmente, è a causa dell'azione integrativa molto elevata. Dopo ancora qualche minuto, l'azione PID comincia a diminuire, inoltre, il controllo non presenta ulteriori oscillazioni, in quanto il duty-cicle si stabilizza attorno all'90%, imponendo una tensione in gate di circa $4\ [V]$.

Concludendo, posso affermare che il controllo PID funzioni correttamente, presentando però un difetto: ha un'importante sovraelongazione, principalmente dovuta al tempo impiegato dal sistema per raggiungere la temperatura. Durante il test sono passati circa 15 minuti affinché la temperatura raggiungesse quella impostata, il programma ha una frequenza di campionamento di circa 1 secondo, che si traduce in circa 900 cicli di software. Una possibile miglioria al potrebbe essere una verifica di superamento delle varie azioni, imporre che tutte le azioni appartengano all'intervallo $[-255, +255]$.

È inoltre evidente come il sistema non sia ben isolato, in quanto, la temperatura interna, una volta raggiunto il riferimento, scenda. Un eventuale sviluppo futuro potrebbe essere l'utilizzo di un migliore isolante per le pareti interne del frigo.

Un altro problema riscontrato è la dispersione dell'aria fredda nell'ambiente interno: provando il frigo, senza l'ausilio del ghiaccio, ho notato che la parte inferiore era, seppur di poco, più calda rispetto a quella superiore (più vicina alla cella). Posizionando dei dissipatori sulla parete superiore e usando una ventola puntata sui dissipatori interni, la temperatura interna è scesa, anche se non di molto.

In fase di test per verificare quale configurazione della ventola fosse la migliore (se spingere dentro aria dall'alto o tirarne dentro dai fori ai lati), ho notato che lo strato di alluminio interno diventava più caldo quando la ventola spingeva l'aria dall'alto sul dissipatore; ciò può essere dovuto ad una sbagliata impostazione della cella, in quanto l'aria spinta passava attraverso il dissipatore raffreddandolo e quindi l'aria, ora calda, passava sull'alluminio scaldandolo. Un'ulteriore miglioria potrebbe essere elevare, con materiale termoconduttivo, la cella in modo da "distanziarla" dallo strato di alluminio interno, impedendone così il surriscaldamento.

11 Riferimenti

[1] Fonte testo: “Cronologia delle scoperte scientifiche” di Isaac Asimov
[2] Fonte testo: https://it.wikipedia.org/wiki/Arduino_(hardware)
[3] Fonte immagine: https://upload.wikimedia.org/wikipedia/commons/thumb/8/87/Arduino_Logo.svg/720px-Arduino_Logo.svg.png
[4] Fonte immagine: Fritzing
[5] Fonte tabella: datasheet Arduino UNO R3
[6] Fonte immagine: https://upload.wikimedia.org/wikipedia/commons/thumb/3/3b/Thermoelectric_Cooler_Diagram.svg/800px-Thermoelectric_Cooler_Diagram.svg.png
[7] Fonte testo: https://it.wikipedia.org/wiki/Effetto_Peltier
[8] Fonte immagine: https://upload.wikimedia.org/wikipedia/commons/thumb/3/3b/Thermoelectric_Cooler_Diagram.svg/800px-Thermoelectric_Cooler_Diagram.svg.png
[9] Fonte testo: https://it.wikipedia.org/wiki/Cella_di_Peltier
[10] Fonte tabella: datasheet TEC1-12710 di Compass DHM Projects
[11] Fonte immagini: datasheet TEC1-12710 di Compass DHM Projects
[12] Fonte tabella: datasheet IRL530N di Vishay
[13] Fonte immagine: datasheet IRL530N di Vishay
[14] Fonte tabella: datasheet AD22100ATZ di Analog Divices
[15] Fonte tabella: datasheet MCP6002 di Microchip
[16] Fonte immagine: “Introduzione alle applicazioni industriali di Microcontrollori e DSP” di Simone Buso
[17] Fonte immagine: “Introduzione alle applicazioni industriali di Microcontrollori e DSP” di Simone Buso
[18] Libreria installabile tramite Arduino IDE Library Manager o scaricabile attraverso GitHub: https://github.com/adafruit/Adafruit_SH110X
[19] Libreria installabile tramite Arduino IDE Library Manager o scaricabile attraverso GitHub: https://github.com/adafruit/Adafruit-GFX-Library