Tesina Tecniche AudioVisive
Realizzazione di un Termometro Digitale Intelligente

Luca Vinciguerra

26 luglio 2017

Indice

0.1 Introduzione 2

1 Progettazione e Realizzazione **3**
1.1 Cos'è Arduino 3
1.2 Texas Instruments LM35 4
1.3 Criteri di Realizzazione del Prototipo 5
1.4 Cos'è Matlab 7
1.5 Introduzione a GUIDE 7
1.6 Criteri di Realizzazione dell'Interfaccia Grafica 9

2 Codice Sorgente **11**
2.1 Gestione di Arduino Uno tramite Matlab 11
2.2 Gestione dei dati acquisiti 12

3 Possibili Sviluppi **17**

A Codice Matlab **18**

0.1 Introduzione

Nell'era dell'Internet of Things dove l'interconnessione tra dispositivi elettronici pone le basi per un futuro dove l'uomo sarà sempre più assistito dalle macchine,questo lavoro si prefigge come obiettivo la trasformazione di un comune oggetto, quale un termometro, in un dispositivo digitale intelligente.

Nello specifico, l'intenzione è quella di progettare e sviluppare un termometro digitale che possa comunicare con altri dispositivi elettronici quali laptop o smartphone, fornendo ad essi in tempo reale misure di temperatura e statistiche sulle stesse.

Per non produrre una semplice traduzione dell'oggetto analogico in una sua versione digitale, per quanto semplice possa essere considerata la sua funzione, si è scelto di ottenere un sistema di acquisizione dati che sia il più possibile user-friendly non soltanto nell'utilizzabilità ma anche nelle funzioni che esso propone. Quindi si è deciso di scindere la parte hardware da quella software per garantire un alto livello di libertà di utilizzo e ci si è concentrati nel fornire all'utente un giusto set di strumenti utilizzabili attraverso un'interfaccia uomo macchina ottimizzata nella veste grafica e nella disposizione dei sui elementi.

Sotto questi vincoli, il dispositivo si compone di un sensore di temperatura Texas Instruments LM35 connesso, insieme ad un LED che si accende alla rilevazione di temperature indesiderate, alla scheda di prototipazione Arduino Uno. Questa è scelta per il buon connubio tra dimensioni e prestazioni e viene programmata tramite il software sviluppato in Matlab che risiede su un laptop.

Nel Capitolo 1 verrà presentata la fase di progettazione e realizzazione del dispositivo e del software, con particolare dettaglio per i criteri di ideazione basati sul raggiungimento di un sistema immediato e facilmente gestibile dall'utente.

Nel Capitolo 2 verrà spiegato il codice sorgente, in particolare la scelta dell'ambiente Matlab e gli algoritmi di cui esso di compone.

Nel Capitolo 3 verranno affrontati i possibili miglioramenti del prototipo realizzato in funzione delle più recenti e future tecnologie di comunicazione.

Capitolo 1

Progettazione e Realizzazione

1.1 Cos'è Arduino

Il progetto Arduino prende forma nel 2005 a Ivrea nell'Interaction Design Institute, un istituto di formazione post-dottorale fondato da Olivetti e Telecom Italia, in cui il team composto da Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis decisero di ideare un dispositivo hardware di controllo programmabile che rappresentasse un'economica alternativa alle più gettonate e costose piattaforme di prototipazione. Tale progetto fu inizialmente sviluppato per una distribuzione interna all' Interaction Design Instituite per poi passare alla commercializzazione su larga scala. Oggi Arduino si definisce uno strumento di ausilio alla creatività di professionisti, hobbisti e artisti offrendo una vasta gamma di schede di varie dimensioni e prestazioni e di un'infinità di sensori il tutto gestibile attraverso un immediato ambiente di programmazione detto Wiring, derivante dal C e C++. Nello specifico, ogni scheda Arduino è composta da un microcontrollore caratterizzato da una quantità eterogenea di PIN, tramite i quali è possibile la comunicazione con i diversi sensori in commercio. I Pin si suddivido in digitali ed analogici che permetto l'invio di segnali, generati con la Pulse Width Modulation, e l'acquisizione di sorgenti esterne. La programmazione della scheda avviene in comunicazione seriale attraverso la porta USB presente in ognuna di esse.

1.2 Texas Instruments LM35

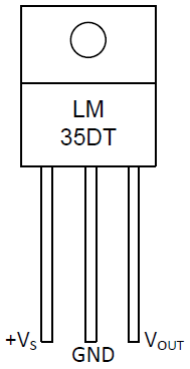

Figura 1.1: Schema sensore di temperatura Texas Instruments LM35

Il sensore di temperatura Texas Instruments LM35 è un circuito integrato il cui voltaggio in uscita è linearmente proporzionale alla temperatura in gradi centigradi. Non richiede nessuna calibrazione esterna o necessita di trimmer, poichè già calibrato in gradi Celsius ed ha un range di funzionamento che va da $-55°C$ a $150°C$. Il sensore ha 3 terminali, il cui schema è riportato in figura, uno per l'alimentazione che va da 4 V a 20 V, un terminale di massa e l'ultimo è per l'uscita della tensione proporzionale alla temperatura rilevata.Quest'ultima è pari a 10 mV per ogni grado centigrado con un errore di ± 5 mV, ovvero $\pm 0.5°C$.

La sua bassa impedenza in uscita, l'output lineare e la precisa calibrazione lo rendono un dispositivo semplice all'uso.

1.3 Criteri di Realizzazione del Prototipo

Come già annunciato nell'introduzione di questo lavoro, il dispositivo viene concepito come un sistema di acquisizione di dati di temperatura dell'ambiente in cui esso è alloggiato, sulla base dei seguenti criteri:

- facilità di posizionamento
- interfaccia uomo macchina user-friendly

Per ottenere un alto grado di libertà nell'utilizzo del dispositivo si è scelto di scindere la componente hardware da quella software. In questo modo è possibile un facile posizionamento della parte hardware dedicata all'acquisizione di valori di temperatura, anche in ambienti di piccole dimensioni.

In questo capitolo si pone il focus sulla progettazione e realizzazione della sola base hardware, mentre la parte software verrà trattata nel paragrafo1.6.

Il prototipo realizzato si compone, quindi, della scheda Arduino Uno a cui sono connessi il sensore di temperatura LM35 ed il Led di stato di colore rosso. Il tutto è alloggiato in box in plexiglas di dimensione 5x11x6cm (L,H,P). La disposizione degli elementi e le dimensioni della scatola sono determinate considerando la necessità di

- dimensione contenute del dispositivo per un facile piazzamento nell'ambiente;
- struttura solida resistente alla possibilità di urti accidentali;
- LED di stato ben visibile per il monitoraggio visivo di temperature indesiderate;

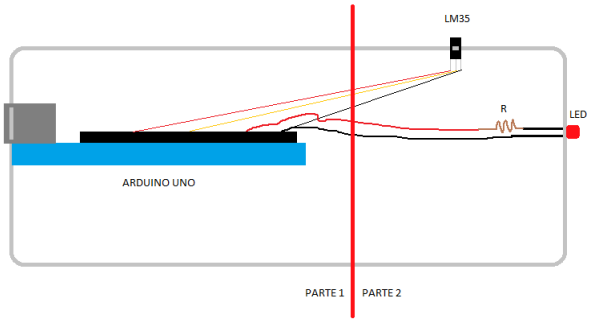

Figura 1.2: Schema progettuale del dispositivo hardware

Dalla Figura1.2 si osserva come il prototipo realizzato possa essere suddiviso in due parti fondamentali: la prima dove risiede la scheda Arduino Uno e

la seconda dove sono alloggiati il sensore di temperatura sulla superficie superiore e il Led di stato sulla superficie laterale corta. In questo modo eventuali surriscaldamenti della scheda di prototipazione non compromettono i valori di temperatura rilevati dal sensore.

Le connessioni circuitali tra i diversi componenti hardware sono riportati in Figura1.3. Il sensore di temperatura LM35 viene alimentato con 5 Volt provenienti dalla scheda, mentre la lettura dei valori di tensione proporzionali alla temperatura sono effettuati connettendo il terminale di V_{out} del sensore al pin analogico $A0$, che effettua un'acquisizione con velocità di circa $9000\frac{sample}{sec}$. Infine il terminale di massa è collegato al pin GND.

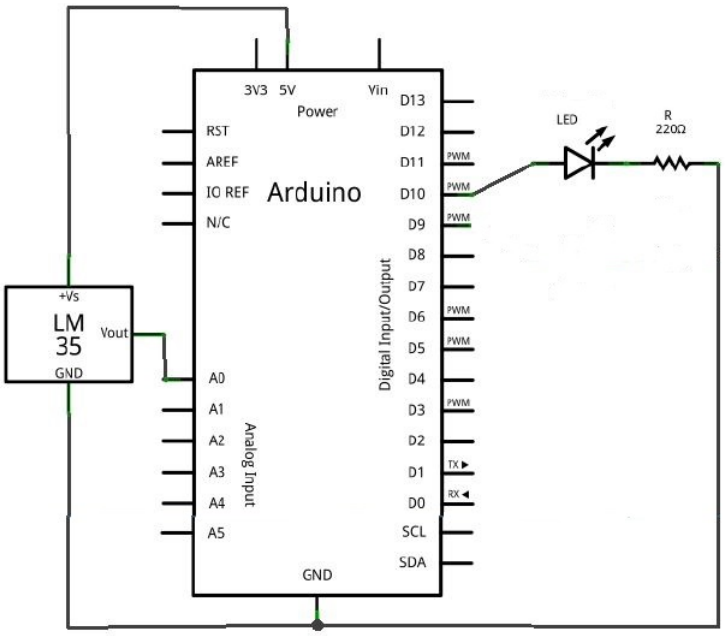

Figura 1.3: Schema circuitale del prototipo

Poichè la funzione del Led di stato è quella di avvertire l'utente della presenza di uno o più valori soprasoglia, questo è connesso tra il pin digitale D10 e mass. Infatti alla rilevazione di valori indesiderati il diodo led viene alimentato con una tensione constante generata con la Pulse Width Modulation tramite il pin digitale scelto. Nello specifico, tra l'anodo del diodo e il pin D10 è interposto un resistore di valore 220Ω, mentre il catodo è connesso ad un secondo punto di massa presente nella scheda Arduino Uno.

Per semplicità di sviluppo si sceglie di utilizzare come dispositivo esterno, ospitante il software sviluppato in Matlab, un laptop connesso alla base hardware tramite connessione seriale con un cavo USB.

1.4 Cos'è Matlab

Matlab è un ambiente di programmazione creato nel linguaggio C da MathWorks per il calcolo numerico e l'analisi statistica. Consente inoltre di manipolare matrici (Matlab è l'abbreviazione di Matrix Laboratory), implementare algoritmi, visualizzare funzioni e dati, creare interfacce grafiche e ha la possibilità di interfacciarsi con altri programmi. L'ideatore è Cleve Moler, presidente del dipartimento di scienze informatiche dell'Università del Nuovo Messico, che negli anni settanta ne iniziò la realizzazione per permettere ai suoi studenti l'accesso a LINPACK e ad EISPACK senza che essi dovessero conoscere il Fortran. Successivamente, questo progetto fu notato dall'ingegnere Jack Little che ne comprese subito il potenziale commerciale, unendosi a Moler fondarono MathWorks e realizzarono quello che noi oggi conosciamo come Matlab. Oggi l'ambiente di programmazione Matlab fornisce una serie di applicazioni denominate Toolbox, le quali forniscono all'utente un'interfaccia grafica per la visualizzazione di dati. Alcuni di questi Toolbox permettono l'elaborazione dei segnali, offrendo la visualizzazione immediata del risultato, oppure la simulazione del funzionamento di motori meccanici fino alla calcolo di statistiche economiche. Oggi Matlab è diventato lo strumento di ricerca più apprezzato e utilizzato nell'ambiente scientifico grazie alla sua efficienza e facilità di programmazione.

1.5 Introduzione a GUIDE

GUIDE è un ambiente di sviluppo grafico per la creazione di interfacce utente. Nello specifico tramite GUIDE è possibile la creazione di un ambiente grafico nel quale vivono gli algoritmi concepiti dal programmatore. Gli stessi Toolbox di Matlab posso essere creati tramite GUIDE. Per accedere all'ambiente di programmazione grafico basta digitare *guide* nella Command Window dell'ambiente desktop di Matlab. Verrà aperto così una finestra costituita da un ambiente di lavoro e un pannello di item. Questi sono bottoni, pannelli o banner già graficamente pre-programmati. In questo modo trascinando l'item scelto nell'area di lavoro verrà automaticamente generata una funzione

con il nome del progetto e per ogni oggetto selezionato verranno create 3 funzioni:

- CreateFunction;
- Callback
- KeyPressFunction;

Nella prima è possibile definire le azioni che verranno eseguite all creazione dell'oggetto. Nella seconda devono essere scritte le azioni che verranno eseguite ogni volta che viene utilizzato l'oggeto. Mentre nell'ultima, le azioni in essa scritte verranno eseguite alla pressione tramite click del mouse dell'oggetto.

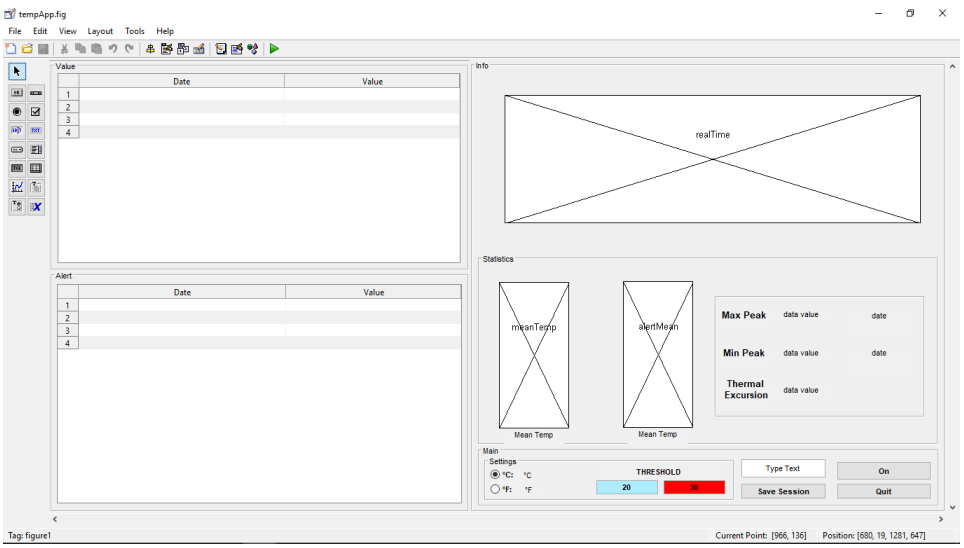

Figura 1.4: GUIDE: interfaccia grafica per la realizzazioni di applicazioni Matlab

1.6 Criteri di Realizzazione dell'Interfaccia Grafica

L'interfaccia grafica è stata progettata sulla base dei seguenti criteri:

- veste grafica user-friendly, per un'efficace e intuitiva gestione dei tool offerti;
- lettura immediata dei risultati numerici;
- semplicità di comprensione dei risultati in forma grafica;

Sulla base di questi, l'interfaccia si suddivide in due parti: quella di sinistra dove sono disposte le tabelle di recording dei valori di temperatura più significativi, mentre la parte di destra è composta da strumenti di visualizzazione grafica dei risultati.

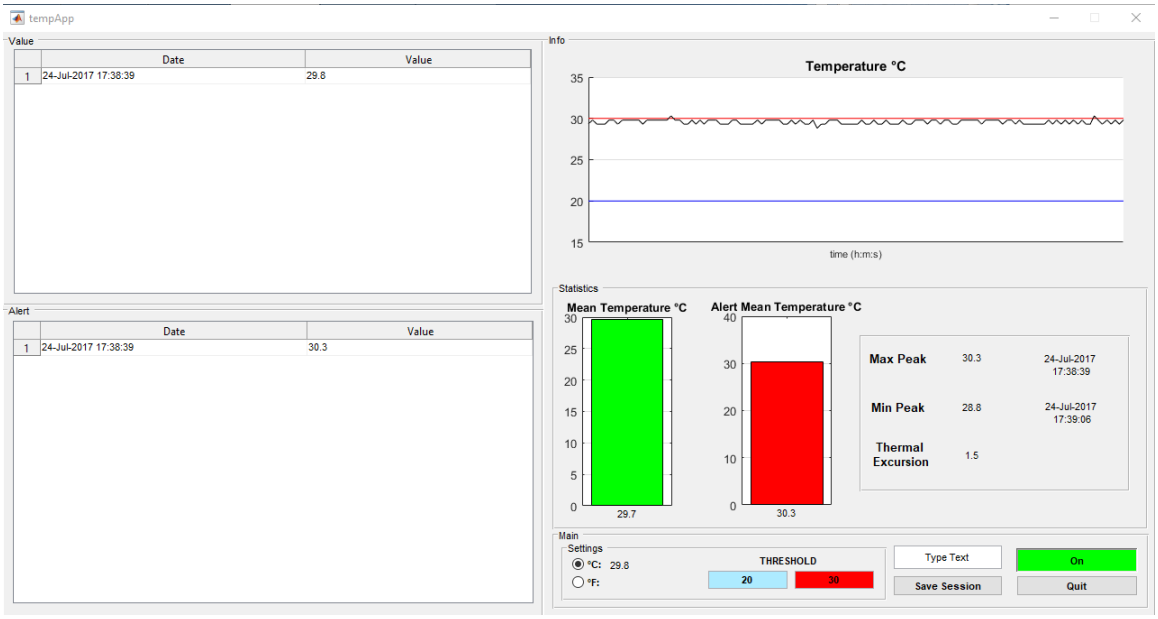

Figura 1.5: Interfaccia grafica realizzata in Matlab

Nello specifico, le tabelle di recording sono due. Nella prima, in alto a sinistra, vengono memorizzati valori di temperatura che hanno un gap di $\pm 1°C$, mentre nella seconda vengono riportate le temperature illecite rispetto alle soglie prestabilite. In entrambe, alla registrazione del valore numerico di temperatura è associata una datazione della rilevazione di tali valori.

La sezione di destra è possibile sezionarla in tre divisioni: la prima dove è presente un grafico Temperatura-Tempo che riporta i valori di temperatura

acquisiti in real time; la secondo,la parte centrale, dove sono presenti i tool di informazioni statistiche, quali i due banner che monitorano la temperatura media dall'inizio dell'acquisizione e la temperatura media degli alert rilevati, oltre alle informazioni sul valore di picco massimo, di picco minimo e la massima escursione termica rilevata. La terza sezione,in basso, è un pannello di setup in cui è l'utente può decidere l'unità di misura della temperatura, i valori di temperatura di soglia minima e massima e salvare la sessione di acquisizione tramite il pulsante 'Save Sassion'. Infine, sono presenti i pulsati di 'Start/Stop' per l'avvio e la sospensione dell'acquisizione dei dati ed il pulsante 'Quit' per la terminazione dell'interfaccia grafica.

Capitolo 2

Codice Sorgente

In questo capitolo verranno spiegate alcune delle funzioni principali di cui si compone il software, in particolare la funzione di comunicazione tra la scheda Arduino Uno e l'interfaccia grafica e la gestione dei dati di temperatura acquisiti.

2.1 Gestione di Arduino Uno tramite Matlab

Per la programmazione delle schede Arduino tramite il linguaggio di programmazione Matlab, Mathwors mette a disposizione il pacchetto *ArduinoSupportPackage* che fornisce un insieme di metodi tramite i quali è possibile la comunicazione con la scheda di prototipazione, la gestione dei pin analogici e digitali ad alcuni metodi più complessi per il controllo di servomotori.

```
48 function tempApp_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
50 % hObject    handle to figure
51 % eventdata  reserved - to be defined in a future version of MATLAB
52 % handles    structure with handles and user data (see GUIDATA)
53 % varargin   command line arguments to tempApp (see VARARGIN)
54 %------------------------
55
56 PIN= 'A0';
57 handles.PIN=PIN;
58 LED='D11';
59 handles.LED=LED;
60 board=arduino('COM3','Uno'); %inizializzazione arduino uno
61 handles.board = board;
62
```

```
63 % Choose default command line output for tempApp
64 handles.output = hObject;
65
66 % Update handles structure
67 guidata(hObject, handles);
```

La porzione di codice riportata riguarda la funzione di creazione dell'intera interfaccia grafica. Non appena i moduli dell'interfaccia vengono creati, vengono inizializzate alcune variabili globali che verranno poi utilizzate in seguito da altre funzioni.

Nello specifico, alla riga 56 viene inizializzato il pin analogico $A0$ per l'acquisizione dei dati provenienti dalla scheda Arduino. Diversamente, alla riga 58 si imposta come pin analogico per l'accensione del Led, il pin digitale $D11$. Già da qui è possibile comprendere la facilità di programmazione che si presenta intuitiva ed immediata.

Alla riga 60 avviene l'inizializzazione della comunicazione con la scheda Arduino. Il metodo *arduino*() richiede in input la porta seriale su cui è collegata la scheda ed il modello utilizzato.

```
nomevaribile = arduino(PortaSeriale,ModelloScheda);
```

Per poter rendere visibili a tutte le funzioni le variabili inizializzate, in GUIDE bisogna creare un handle, ovvero un riferimento alla variabile desiderata.

La sintassi è la seguente:

```
handles.nomevariabile = variabile;
```

2.2 Gestione dei dati acquisiti

Il codice riportato fa riferimento alla Callback del pulsante 'On' di avvio della sessione di acquisizione dei dati.

Alla riga 224 è impostato un *while* per il loop che permette l'acquisizione continua dei valori di tensione forniti dalla scheda Arduino Uno. La condizione di uscita dal *while* è il valore associato al pulsante 'On', ovvero finchè il valore del pulsante in questione sarà diverso da zero allora avverrà l'acquisizione dei dati. Quindi alla pressione del pulsante 'On' il suo valore associato cambia dalla condizione di default uguale a 0 in 1 e viceversa.

Alla riga 228 avviene la vera e propria memorizzazione dei valori di tensione forniti da Arduino Uno tramite il metodo $readVoltage()$ che ha come input la variabile associata alla scheda Arduino e il pin da cui vengono letti i valori forniti dal sensore di temperatura.

```
nomevaribile = readVoltage(NomeScheda,NomePin);
```

Poichè i valori trasmessi dal sensore Texas Instruments LM35 sono di natura analogica, allora il pin di lettura sarà anch'esso analogico. Infatti, il riferimento $handles.PIN$ è associato al pin $A0$.

La conversione dei valori di tensione in valori di temperatura è effettuata nella riga 230. Per una facile comprensione di tali formule è necessario specificare come avviene la trasmissione dei dati dalla scheda Arduino al laptop utilizzato.

Come già detto i valori di tensione provenienti dal sensore di temperatura sono valori di ampiezze continue nel tempo, quindi sono segnali analogici. Quando la scheda Arduino riceve tali valori nel rispettivo pin analogico, prima che questi vengano memorizzati nei buffer della scheda, subiscono una conversione al discreto. In altri termini, i valori di ampiezza delle tensioni fornite dal sensore LM35 vengono quantizzati e convertiti a valori discreti dal modulo Analog to Digital Converter presente in Arduino Uno.

Nello specifico alla tensione di $0V$ è associato il valore 0, mentre alla tensione massima di alimentazione $5V$ è associato il valore 1023, questo perchè l'ADC presente nella scheda Arduino Uno è a 10 bit. Quindi, tali valori discreti vengono trasmessi al laptop che li converte in valori di temperatura. Poichè un aumento di $1°C$ corrisponde ad un aumento della tensione di $10mV$, allora la formula di conversione da tensione a gradi centigradi è la seguente

$$Temperatura(°C) = ValoreTensione(mV)\frac{5000(mV)}{1023 * 10(mV)} \tag{2.1}$$

Il metodo $readVoltage()$ conosce già il fattore di conversione $\frac{5000(mV)}{1023}$, cosicché la formula di conversione diventa

$$Temperatura(°C) = \frac{ValoreTensione(mV)}{10(mV)} \tag{2.2}$$

Una volta ottenuto il valore di temperatura in gradi centigradi si può ricavare la temperatura in gradi Fahrenheit con l'usuale formula riportata in riga 231:

$$Temperatura(°F) = \frac{9}{5} * Temperatura(°C) + 32 \tag{2.3}$$

```
%---------------------------Inizio Acquisizione Dati---------------------
221    startTime = datetime('now');
222
223
```

```
224    while get(hObject,'Value')~=0
225
226
227        % Read current voltage value
228        v = readVoltage(handles.board,handles.PIN);
229        % Calculate temperature from voltage (based on datasheet)
230        TempC = v*100;
231        TempF = 9/5*TempC + 32;
232        % Get current time
233        t =  datetime('now') - startTime;
234
235
236
237        %Check unit
238        if unit==1
239            Temp=TempC;
240            set(handles.tempCValue,'String',num2str(round(TempC,1)));
241            set(handles.tempFValue,'String',' ');
242            ax.Title.String = ' Temperature $^\cent C$ ';
243
244        else
245            Temp=TempF;
246            set(handles.tempFValue,'String',num2str(round(TempF,1)));
247            set(handles.tempCValue,'String',' ');
248            ax.Title.String = ' Temperature $^\cent F$ ';
249        end
250
251
252
```

Nell'interfaccia grafica vengono riportate le informazioni di valore massimo di temperatura rilevato, valore minimo ed escursione termica massima. Alla riga 253 è posto un *if* per la ricerca del massimo valore di temperatura, lo stesso viene fatto alla riga 262 per la ricerca del valore minimo. Per quanto riguarda l'escursione termica massima, questa viene aggiornata ogni qualvolta cambiano i valori di massimo e minimo e calcolata come la loro differenza (riga 269).

```
253        %Find Max Peak
254        if Temp>maxPeak
255            maxPeak = Temp;
```

```
256             set(handles.valueMax,'String',num2str(round(maxPeak,1)));
257             dateMax = datetime('now');
258             set(handles.dateMax,'String',char(dateMax));
259         end
260
261         %Find Min Peak
262         if Temp<minPeak
263             minPeak = Temp;
264             set(handles.valueMin,'String',num2str(round(minPeak,1)));
265             dateMin = datetime('now');
266             set(handles.dateMin,'String',char(dateMin));
267         end
268         %Thermal Excursion
269         et=maxPeak-minPeak;
270         set(handles.etValue,'String',num2str(round(et,1)));
271
```

Il codice che segue è relativo alle tabelle di recording. Nella prima sono memorizzati valori di temperatura significativi, ovvero ogni volta che la temperatura cambia di $\pm 1°C$. In questo modo si offre all'utente una lettura immediata ed efficiente dell'andamento nel tempo della temperatura. Al contrario, se fossero registrati tutti i valori acquisiti in un sessione, si avrebbe una forte ridondanza dei valori acquisiti non che una tabella con un numero ingente di righe che porterebbe ad una comprensione difficile e confusionaria dell'evoluzione della temperatura nel tempo. Inoltre, basti pensare ad una sessione in cui la temperatura rimane piuttosto costante per capire che la registrazione di tutti i valori acquisiti porterebbe solo ad uno spreco di memoria.

```
281         %Data Table: gap>1 between data
282         if nn == 1
283             cValue(nn,:)={ char(datetime('now')), num2str(round(Temp,1))};
284             set(handles.valueTable,'Data',cValue);
285             nn=2;
286             cTmp=Temp;
287
288         else
289             gap= abs(Temp-cTmp);
290             if gap>1
```

```
291                 cValue(nn,:)={ char(datetime('now')), num2str(round(Temp,1))}
292                 set(handles.valueTable,'Data',cValue);
293                 nn=nn+1;
294                 cTmp=Temp;
295             end
296         end
297         handles.cValue = cValue;
```

La seconda tabella memorizza le temperature illecite che oltrepassano le soglie prestabilite. Il criterio di memorizzazione è lo stesso della tabella precedentemente discussa con l'unica differenza che in questo caso un valore di alert verrà memorizzato se il gap tra esso ed il precedente è maggiore o uguale a 0.8.

```
299         %Check Alert & Alert Data Table
300         if Temp>handles.maxT || Temp<handles.minT
301
302             if row == 1
303                 Alert(row,:)={ char(datetime('now')), num2str(round(Temp,1))}
304                 writeDigitalPin(handles.board,handles.LED,1);
305                 set(handles.alertTable,'Data',Alert);
306                 row=row+1;
307
308             else
309
310                 e = abs(round(Temp,1)-tmp);
311
312                 if e>=0.8
313                     Alert(row,:)={ char(datetime('now')), num2str(round(Temp,
314                     set(handles.alertTable,'Data',Alert);
315                     row=row+1;
316                 end
317             end
318             tmp=round(Temp,1);
319         end
320         handles.Alert = Alert;
```

Capitolo 3

Possibili Sviluppi

Il prototipo realizzato è una chiara rappresentazione del potenziale dell'interazione tra analogico e digitale e di come sia possibile sfruttare in modo intelligente un strumento la cui funzione risulta semplice all'apparenza. Per ampliare l'efficacia del dispositivo, da subito è evidente la necessità di una connessione wireless tra la base hardware e quella software, in modo da elevare la libertà di utilizzo.

Inoltre,si potrebbe dotare il dispositivo di un sensore di temperatura di precisione in modo da ottenere una maggiore risoluzione. Per una maggiore fruibilità delle funzione proposte in questo lavoro, si potrebbe sfruttare una connessione WiFi per la comunicazione con più di un dispositivo esterno ai quali recapitare un messaggio di alert ogni volta che vengono rilevate temperature soprasoglia. In tal modo, si potrebbe sviluppare un'applicazione per i sistemi Androin o IOS, con la quale controllare lo status del monitoraggio. Ovviamente con le moderne tecnologie sarebbe possibile minimizzare le dimensioni del prototipo, in modo da poterlo posizionare in qualsiasi ambiente. Un interessante ulteriore progetto sarebbe la definizione di cluster di termometri digitali che comunicano tra loro e con un controller esterno. Posizionando ogni termometro intelligente in un punto diverso si otterrebbe una mappatura della temperatura dell'ambiente considerato, producendo così un'analisi statistica più completa e interessante.

I possibili campi di applicazione sono tutti quelli per cui si ha necessità di monitorare costantemente la temperatura di un ambiente, quale per esempio una serra o una cella frigorifera, ed agire velocemente all'occorrenza di valori illeciti.

Appendice A

Codice Matlab

```
\begin{flushleft}
function varargout = tempApp(varargin)
% TEMPAPP MATLAB code for tempApp.fig
%\begin{flushleft}
      TEMPAPP, by itself, creates a new TEMPAPP or raises the existing
%      singleton*.
%
%      H = TEMPAPP returns the handle to a new TEMPAPP or the handle to
%      the existing singleton*.
%
%      TEMPAPP('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TEMPAPP.M with the given input arguments.
%
%      TEMPAPP('Property','Value',...) creates a new TEMPAPP or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before tempApp_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to tempApp_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
\end{flushleft}

% Edit the above text to modify the response to help tempApp

% Last Modified by GUIDE v2.5 24-Jul-2017 14:34:17
```

```
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @tempApp_OpeningFcn, ...
    'gui_OutputFcn',  @tempApp_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before tempApp is made visible.
function tempApp_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to tempApp (see VARARGIN)
%-------------------------
%-------------------------

PIN= 'A0';
handles.PIN=PIN;
LED='D11';
handles.LED=LED;
board=arduino('COM3','Uno');   %inizializzazione arduino uno
handles.board = board;

% Choose default command line output for tempApp
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes tempApp wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = tempApp_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.PIN;
guidata(hObject,handles)

% --- Executes on button press in cent.
function cent_Callback(hObject, eventdata, handles)
% hObject    handle to cent (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
val=get(hObject, 'Value');

if val==1
    set(handles.far,'Value',0);
end
% Hint: get(hObject,'Value') returns toggle state of cent
guidata(hObject,handles)

% --- Executes on button press in far.
function far_Callback(hObject, eventdata, handles)
% hObject    handle to far (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of far
val=get(hObject, 'Value');
if val==1
    set(handles.cent,'Value',0);
```

```
end
guidata(hObject,handles)

% --- Executes on button press in saveSession.
function saveSession_Callback(hObject, eventdata, handles)
% hObject    handle to saveSession (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)dlm
name=get(handles.nameSession,'String');
if get(handles.cent,'Value')==1
    data = cell2table( get(handles.valueTable,'Data'),...
    ,'VariableNames',{'Date','Cent'});
else
    data = cell2table( get(handles.valueTable,'Data'),...
    ,'VariableNames',{'Date','Far'});
end

dataname=[name '_data.txt'];
writetable(data,dataname,'delimiter','\t');




if get(handles.cent,'Value')==1
    alertdata = cell2table( get(handles.alertTable,'Data'),...
    ,'VariableNames',{'Date','Cent'});
else
    alertdata = cell2table( get(handles.alertTable,'Data'),...
    ,'VariableNames',{'Date','Far'});
end

alertdataname=[name '_alertdata.txt'];
writetable(alertdata,alertdataname,'delimiter','\t');

orient landscape
[file, path] = uiputfile('*.pdf');
print([path file], '-dpdf','-fillpage');

function nameSession_Callback(hObject, eventdata, handles)
% hObject    handle to nameSession (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of nameSession as text
%        str2double(get(hObject,'String')) returns contents
%      of nameSession as a double


% --- Executes during object creation, after setting all properties.
function nameSession_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nameSession (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
set(hObject,'String','Type Text');
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgr
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in quitButton.
function quitButton_Callback(hObject, eventdata, handles)
% hObject    handle to quitButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
pause('on');
axes(handles.realTime)
cla;
fclose(serial(handles.board.Port));
clear handles.board;

close all;



% --- Executes on button press in onButton.
function onButton_Callback(hObject, eventdata, handles)
% hObject    handle to onButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
%-----------------------Parametri Iniziali--------------------------------
val=get(hObject,'Value');
row=1;
nn=1;
i=1;
tmpA=0;
Alert(1,:) = { '0' , 0 };
cValue(1,:) = { '0' , 0 };
maxPeak=1;
dateMax='1';
minPeak=1000;
dateMin='1';


%-----------------------Azione Pulsante On--------------------------------
if val == 1
    unit=get(handles.cent,'Value');   %check unità di misura

    set(hObject,'BackgroundColor', [0,1,0]); %cambio colore (verde) alla pressio
    set(hObject,'String','On');

    %----------------------Setting RealTime Plot-----------------------------
    axes(handles.realTime);
    minThresh=animatedline('Color',[0,0,1]);
    handles.minThresh=minThresh;
    maxThresh=animatedline('Color',[1,0,0]);
    handles.maxThresh=maxThresh;
    h=animatedline;
    handles.DATA=h;
    ax = gca;
    handles.ax=ax;
    ax.YGrid = 'on';
    ax.YLim = [handles.minT-5 handles.maxT+5];
    ax.XLabel.String = ' time (h:m:s) ';
    ax.XLabel.FontSize = 8;

    %----------------------------Inizio Acquisizione Dati---------------------
    startTime = datetime('now');


    while get(hObject,'Value')~=0
```

```
% Read current voltage value
v = readVoltage(handles.board,handles.PIN);
% Calculate temperature from voltage (based on data sheet)
TempC = v*100;
TempF = 9/5*TempC + 32;
% Get current time
t =  datetime('now') - startTime;



%Check unit
if unit==1
    Temp=TempC;
    set(handles.tempCValue,'String',num2str(round(TempC,1)));
    set(handles.tempFValue,'String',' ');
    ax.Title.String = ' Temperature $^\circ C$ ';

else
    Temp=TempF;
    set(handles.tempFValue,'String',num2str(round(TempF,1)));
    set(handles.tempCValue,'String',' ');
    ax.Title.String = ' Temperature $^\circ F$ ';
end



%Find Max Peak
if Temp>maxPeak
    maxPeak = Temp;
    set(handles.valueMax,'String',num2str(round(maxPeak,1)));
    dateMax = datetime('now');
    set(handles.dateMax,'String',char(dateMax));
end

%Find Min Peak
if Temp<minPeak
    minPeak = Temp;
    set(handles.valueMin,'String',num2str(round(minPeak,1)));
    dateMin = datetime('now');
```

```
    set(handles.dateMin,'String',char(dateMin));
end
%Thermal Excursion
et=maxPeak-minPeak;
set(handles.etValue,'String',num2str(round(et,1)));

%Data Plotting
addpoints(minThresh,datenum(t),handles.minT)
addpoints(maxThresh,datenum(t),handles.maxT)
addpoints(h,datenum(t),round(Temp,1))
% Update axes
ax.XLim = datenum([t-seconds(15) t]);
datetick('x','keeplimits')
drawnow

%Data Table: gap>1 between data
if nn == 1
    cValue(nn,:)={ char(datetime('now')), num2str(round(Temp,1))};
    set(handles.valueTable,'Data',cValue);
    nn=2;
    cTmp=Temp;

else
    gap= abs(Temp-cTmp);
    if gap>1
        cValue(nn,:)={ char(datetime('now')), num2str(round(Temp,1))};
        set(handles.valueTable,'Data',cValue);
        nn=nn+1;
        cTmp=Temp;
    end
end
handles.cValue = cValue;

%Check Alert & Alert Data Table
if Temp>handles.maxT || Temp<handles.minT

    if row == 1
        Alert(row,:)={ char(datetime('now')), num2str(round(Temp,1))};
        writeDigitalPin(handles.board,handles.LED,1);
        set(handles.alertTable,'Data',Alert);
        row=row+1;
```

```
    else

        e = abs(round(Temp,1)-tmp);

        if e>=0.8
            Alert(row,:)={ char(datetime('now')), num2str(round(Temp,1))
            set(handles.alertTable,'Data',Alert);
            row=row+1;
        end
    end
    tmp=round(Temp,1);
end
handles.Alert = Alert;
%-----------------------------Statistics---------------------------------
%-------------------------Mean Temperature-----------------------------
avg= (Temp+(i-1)*tmpA)/i;
tmpA=avg;
i=i+1;
x_avg= num2str(round(avg,1));

set(handles.meanTempTxt,'String', x_avg);

axes(handles.meanTemp);
mtBar= gca;
bar(avg,'green');
if unit == 1
    title('Mean Temperature $^\circ C$','FontSize',10)
else
    title('Mean Temperature $^\circ F$','FontSize',10)
end
mtBar.YGrid = 'on';

%----------------------Alert Mean Temperature-------------------------
if row>1
    a_avg=sum(str2num(cell2mat(Alert(:,2))))/length(Alert(:,2));
    xa_avg= num2str(round(a_avg,1));

    set(handles.alertMTemp,'String', xa_avg);

    axes(handles.alertMean);
```

```
            amtBar=gca;
            bar(a_avg,'red');
            if unit == 1
                title('Alert Mean Temperature $^\circ C$','FontSize',10)
            else
                title('Alert Mean Temperature $^\circ F$','FontSize',10)
            end
            amtBar.YGrid = 'on';

        end



    end %while end
else
set(hObject,'String','Stop');
set(hObject,'BackgroundColor','r');
cla(handles.realTime)

end %first if
guidata(hObject, handles);

% --- Executes on button press in pauseButton.
function minThreshold_Callback(hObject, eventdata, handles)
% hObject    handle to minThreshold (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of minThreshold as text
%        str2double(get(hObject,'String')) returns contents of minThreshold as a
minStr=get(hObject,'String');
minT=str2num(minStr);
handles.minT=minT;

guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function minThreshold_CreateFcn(hObject, eventdata, handles)
% hObject    handle to minThreshold (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
minStr=get(hObject,'String');
minT=str2num(minStr);
handles.minT=minT;

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgr
    set(hObject,'BackgroundColor','white');
end

guidata(hObject, handles);

function maxThreshold_Callback(hObject, eventdata, handles)
% hObject    handle to maxThreshold (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of maxThreshold as text
%        str2double(get(hObject,'String')) returns contents of maxThreshold as a
maxStr=get(hObject,'String');
maxT=str2num(maxStr);
handles.maxT=maxT;

guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function maxThreshold_CreateFcn(hObject, eventdata, handles)
% hObject    handle to maxThreshold (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
maxStr=get(hObject,'String');
maxT=str2num(maxStr);
handles.maxT=maxT;

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgr
    set(hObject,'BackgroundColor','white');
end
guidata(hObject, handles);
```

```
% --- Executes on button press in threshold.
function threshold_Callback(hObject, eventdata, handles)
% hObject    handle to threshold (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of threshold
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function onButton_CreateFcn(hObject, eventdata, handles)
% hObject    handle to onButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
guidata(hObject, handles);

% --- Executes on key press with focus on onButton and none of its controls.
function onButton_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to onButton (see GCBO)
% eventdata  structure with the following fields (see MATLAB.UI.CONTROL.UICONTRO
% Key: name of the key that was pressed, in lower case
% Character: character interpretation of the key(s) that was pressed
% Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed
% handles    structure with handles and user data (see GUIDATA)
guidata(hObject, handles);

\end{flushleft}
```