Dalton Bermudez, Andrew Chen

05/14/16

Introduction:

Hardware and software are integrated to optimize the performance and functionality of devices. To optimize the performance of a temperature sensing devise an open source arduino integral development environment was incorporated along with the circuit. The programing language used to program the arduino board was MATLAB.

Circuit Design/Theory:

The circuit consists of a voltage divider with both regular and thermal resistors. The circuit sensitivity equation (a) was used to determine a combination of resistors and thermistors needed to maximize the voltage divider sensitivity.

(a) ** $S_x^y = \frac{lim}{\Delta x \to 0} \{\frac{\frac{y}{\Delta y}}{\frac{x}{\Delta x}}\} = \frac{y}{x}\frac{\partial y}{\partial x} = S_{R_{th}}^{T_{cd}} = \frac{T_{cd}}{R_{th}} \cdot \frac{\partial T_{cd}}{\partial R_{th}} ; \frac{\partial T_{cd}}{\partial R_{th}} = \frac{R}{(R+R_{th})^2}$

**Only applies when both R_{th} of each thermistor are equal and the resistance (R) of the regular resistors are equal to one another.

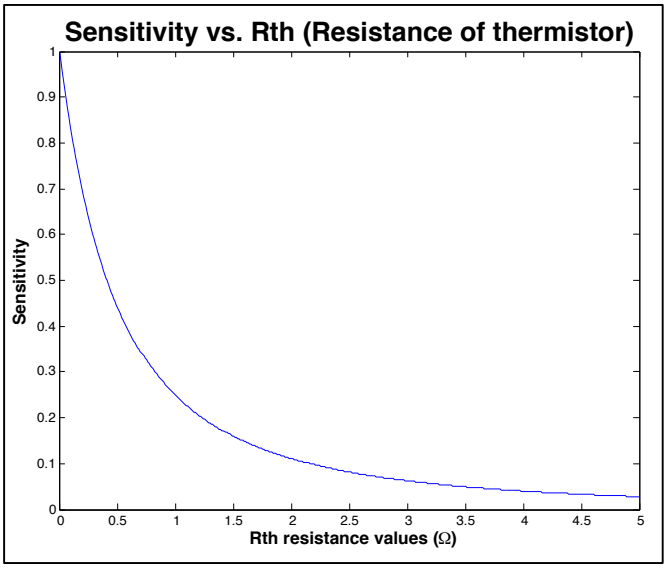


Fig 3. Instead of evaluating each resistor using the corresponding sensitivity value for the transmissibility formula for a thermistor-resistor voltage divider. A general sensitivity curve was plotted based on the derived sensitivity equation (a).

The sensitivity curve graph (Fig. 3) demonstrates that the closer the thermistors' resistances values are to that of the resistor, then the higher the sensitivity of the voltage divider. Based on this, one can infer that the resistor that will provide the highest sensitivity for the thermistor-resistor voltage divider is a 10 kΩ resistor.

The voltage divider equation (b) and the Steinhart-Hart Resistance equation (c) were used to determine the temperature values from the thermistor's potential difference.

(b) $V_{out} = \frac{R_{th}}{R_{th}+R} V_{in}$

(c) $T = \frac{\beta}{\frac{\beta}{298.15K} + \ln(\frac{R_{th}}{R_{25^oC}})}$

The circuit also consists of two 1.6kΩ resistors each in series with a light-emitting diode (Fig 1a). MATLAB programming was used to obtain the voltage readings from the thermistor through the Analogue in A0 pin. If, elseif, else statements were implemented in the code to set the thresholds for turning on the green LED (temperatures less than 10ºC) and for turning on the red LED (temperatures greater than 70ºC).

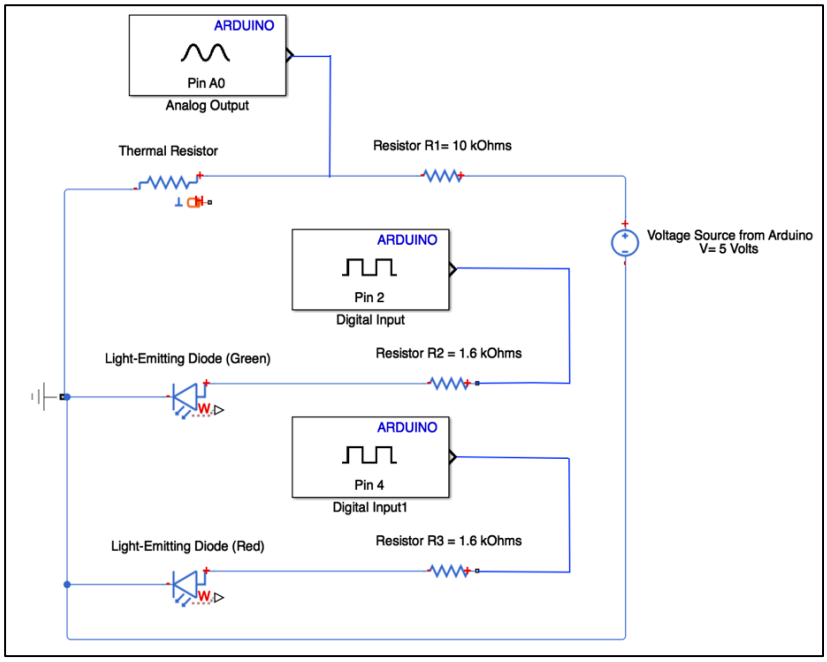
Calibrations:

To determine the coefficients of calibration (C1 and C2), the temperature values obtained from the device, for water at different temperature, were compared to those of the gold standard (multi-meter). The coefficient of calibration was obtained by taking the average differences between the temperatures of the devise from those of the gold standard. Then, the coefficient of calibration was incorporated into the Steinhart-Hart Resistance equation to obtain temperature readings from the device that are within 1ºC of the gold standard.

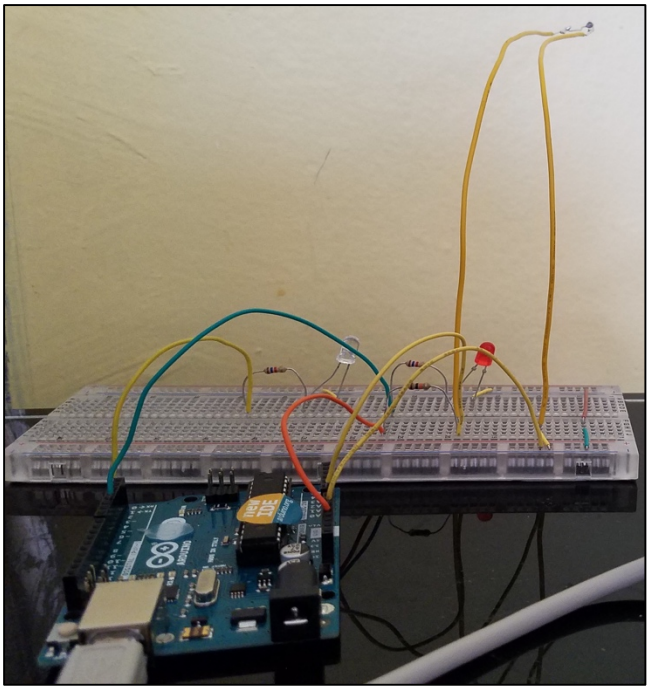
Schematic:

Figure 1:

(a)



(b)



(c)

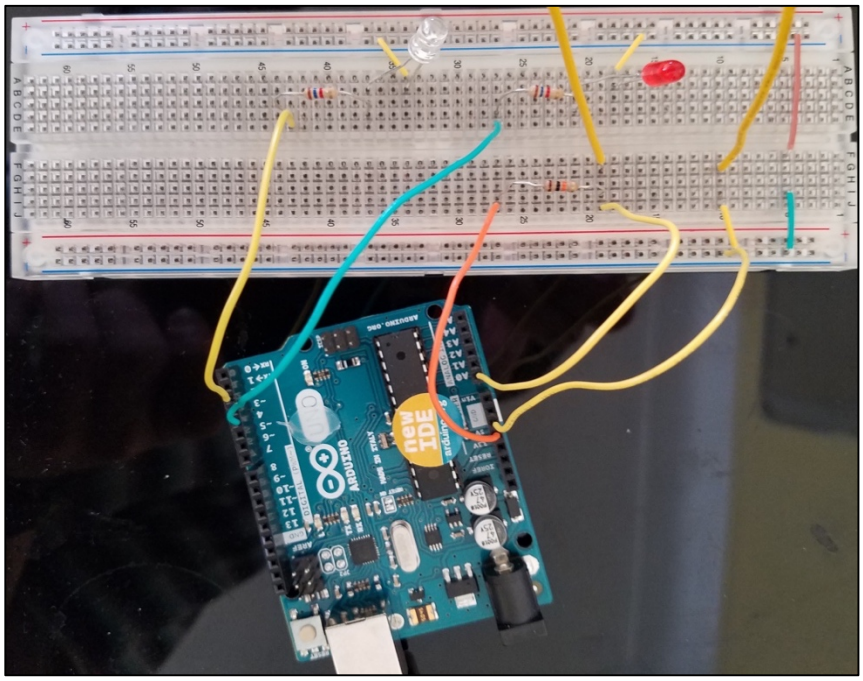


Figure # 1:
(a) Schematic of the different components in the circuit. **(b)** Image the whole circuit. **(c)** Image of a close up of the circuit board and arduino.

Code:

```
function varargout = lmtemp(varargin)
% LMTEMP MATLAB code for lmtemp.fig
%      LMTEMP, by itself, creates a new LMTEMP or raises the existing
%      singleton*.
%
%      H = LMTEMP returns the handle to a new LMTEMP or the handle to
%      the existing singleton*.
%
%      LMTEMP('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in LMTEMP.M with the given input arguments.
%
%      LMTEMP('Property','Value',...) creates a new LMTEMP or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before lmtemp_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to lmtemp_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

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

% Last Modified by GUIDE v2.5 28-Nov-2015 23:29:17

% Begin initialization code for the GUI figure - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @lmtemp_OpeningFcn, ...
                   'gui_OutputFcn',  @lmtemp_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 lmtemp(GUI figure) is made visible.
function lmtemp_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 lmtemp (see VARARGIN)

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

% Update handles structure
guidata(hObject, handles);
% UIWAIT makes lmtemp wait for user response (see UIRESUME)
```

```
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = lmtemp_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.output;% DO NOT Change the variable varargout{1}
clear all;
global a;% defines variable 'a' as a global variable
a = arduino;% declares variable a as the arduino being used



% DO NOT EDIT
function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (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 edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double
%DO NOT EDIT

%DO NOT EDIT
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
 set(hObject,'BackgroundColor','white');
end
%DO NOT EDIT

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)% functions which defines the
run pushbutton in the matlab GUI figure to start running the graph of Temperature Reading
x = [];% creates a empty matrix so the corresponding tie values can be fitted into the
matrix
c1= 1.0; % constant determined based on the calibrations
c2= 2.8;
c3= 12.0;
global a;% makes a = arduino defined for the entire loop
for i = 1:180 % defines the time interval the program will run (set to 900 sec = 15
minutes)
    v =readVoltage(a,'A0'); %reads the voltage output for the thermistor which is
connected to Analog A0
    Rt = 10*v/(5-v);% voltage divider equation use to calculate the resistance of
thermistor

    if v > 2.45 % if statement to indicate which calibration constant will be applied to
equation depending on voltage reading obtained
    temp = (((3977)/(((3977)/(298.17))+(log(Rt/10))))-273.15)-c1;% equations which
computes the temperature value based on the resistance of the thermistor
```

```
    elseif v<= 2.45 && v>1.77
       temp = (((3977)/(((3977)/(298.17))+(log(Rt/10))))-273.15)+c2;
    elseif v<= 1.77
        temp = (((3977)/(((3977)/(298.17))+(log(Rt/10))))-273.15)+c3;
    end

     x = [x temp];% creates matrix which concats together the temperature values with the
corresponding time values
     plot(handles.axes1, x, '*r')% plots the temperature value readings from the
thremistors with the corresponding times
     axis(handles.axes1);% enables to access axis for the set up GUI figure
     xlabel('Time (s)', 'FontSize', 20)% labels the x-axis of the active graph in the GUI
figure
     ylabel('Temperature (∫C)', 'FontSize', 20)% labels the y-axis of the active graph in
the GUI figure
     title('Temperature Reading', 'FontSize', 20)% Adds title to the active graph of the
GUI figure
    grid;% adds a grid on the active graph of the GUI figure
    set(handles.edit1,'String',num2str(temp));% displays the temperature values in the
display box of the GUI figure
    disp(temp);% displays the computed temperature values in the command windo
    pause (1);% the temperature values are outputted every .01 second

   % statements which set the thresholds for the two LED lights to turn on
   % or off
    if temp < 15 %declares the any temperature value less than 15∫C the green LED will
turn on (connected to the Digital PWM out #2)
        writeDigitalPin(a, 'D2', 1)
        else if  temp > 40 %declares the any temperature value more than 40∫C the red LED
will turn on (connected to the Digital PWM out #4)
        writeDigitalPin(a, 'D4', 1)
    else % any other temperature not in the ranges states above the Two LED lights will
reamin off
        writeDigitalPin(a, 'D2', 0)
        writeDigitalPin(a, 'D4', 0)
            end
    end
end
H = 1:i;% makes a vector of all the times at which the temperature is being measured.
uisave({'H', 'x'}, 'var1');% Function which saves output values for temperature and time
coordinates into a matlab m-file
guidata(hObject,handles);% sets the settings for the GUI figure


% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Graphs (Save temperature values to Computer files):

Figure 2:

(a) **(b)**

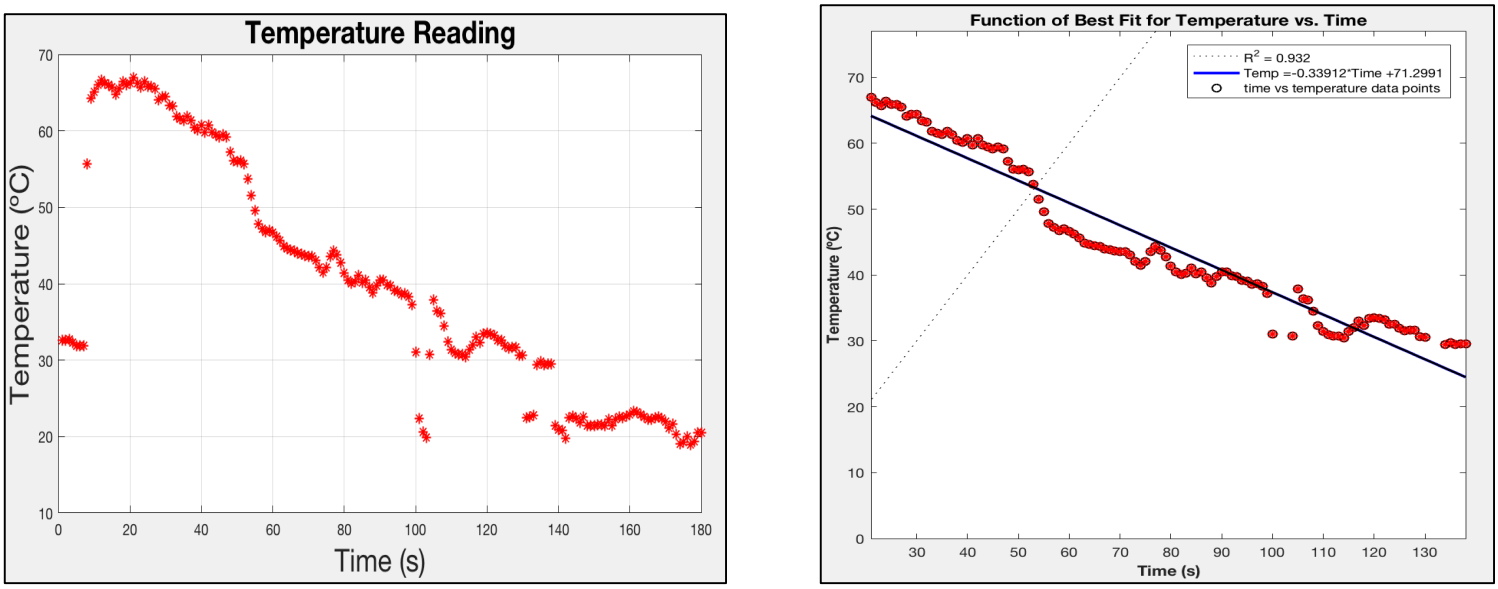


Figure 2:
(a) Displays an image of the time vs. temperature (raw data) values obtained by exposing the thermistor to boiling water and gradually adding cold water. **(b)** Plotting processed data points from region were the thermistors was initial exposed to boiling water, cold water was gradually added, and linear regression line of best fit was determined to obtain characteristic linear equation.

Code (To find the equation of the changes in temperature over time):

```
load('var1.mat') %loads variables with the data points that where saved as an m-file fro changes of temperature
changes over time from boiling to room temperature
H1 = H;% declares the saved time data as another variable
x1= x;% declares the saved temperature data as another variable
[M, I]= max(x1(:));% finds the maximum value of the temperature and its respective position within the x1 matrix
m = min(x1(:));% finds the minimum value of the temperature within the x1 matrix
threshold_1= I;% the position of the maximum value of the temperature within the x1 matrix is used as an upper
threshold point
threshold_2 = 25;% the room temperature of 25∫C is used as a lower threshold point
H1(H1< threshold_1) = NaN;%sets any time that comes before the time which corresponds to the highest temperature
reading equal to NaN
x1(x1< threshold_2) = NaN;%sets any temperature value below the standard room temperature threshold equal to
NaN.
% eliminates all the time and temperature NaN from the matrixes
valid_H1 = ~isnan(H1);
vaild_x1 = ~isnan(x1);
validdataboth = valid_H1 & vaild_x1;
H1 = H1(validdataboth);
x1 = x1(validdataboth);
% finds the line of best fit for the processed time and temperature data points
plotregression(H1,x1)% plots line of best fits and finds its R^2 value
plot(H1,x1, '*', 'color', [1 0 0])% plots data points collected
p= polyfit(H1,x1,1);% fines polynomial of best fit of degree 1 that fits the data collected
v= polyval(p, H1);% maps the values obtained as result of numerically solving the polynomial of best fit

hold on % enables to plot the polynomial of best fit of degree one on the same figure as the data points
plot(H1, v, 'color', [0 0 0], 'LineWidth', 1.5)% plot polynomial of best fit degree 1
xlabel('Time (s)', 'FontWeight', 'bold')% labels the x-axis of the figure (graph)
ylabel('Temperature (∫C)', 'FontWeight', 'bold')% labels the y-axis of the figure (graph)
title('Function of Best Fit for Temperature vs. Time')% Adds title to the figure (graph)
legend('R^2 = 0.932', strcat('Temp =', num2str(p(:,1)) ,'*Time +', num2str(p(:,2))),'time vs temperature data
points' ,'Location', 'NorthEast')% adds legends to the figure an displaysit at the NorthEast side of the figure
ylim([0 M+10])% sets the y-scale of the plot
hold off % ends statement of hold on that was previously declared
```