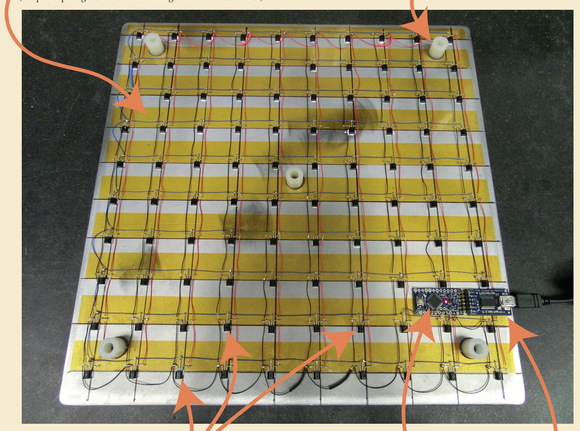
Apparatus for Quantitative Measurement of Heat Flow in Two Dimensions

Eric Ayars, Daniel Lund, and Lawrence Lechuga
California State University, Chico
ayars@mailaps.org

Base: 30 cm square, 5mm thick Al plate

Kapton tape
(Helps keep magic smoke from leaking out of the electronics.)

Stand-offs
(so we can put it right side up when we're done looking at this bottom side.)



100 DS18B20 temperature sensors, glued in place with thermal epoxy

Arduino Pro Mini (brains) with USB Adaptor board (Communications)

Red wire: +5V Black wire: ground Blue wire: data bus
Note: There's a 1k pullup resistor between +5V and data.

Before wiring the sensors, run this program with each sensor connected individually to find the address of each sensor.

This lets you build a 'map' of which address corresponds to each sensor position on the grid.

```
/*
 * FindAddress.ino
 *
 * Arduino code: When run this program will tell us the addresses
 * of any One-Wire devices on the bus.
 *
 */

#include "OneWire.h"
#include "DallasTemperature.h"
#define NDevices 16                     // Max number of devices for now

OneWire oneWire(10);                    // Bus on pin 10
DallasTemperature sensors(&oneWire);    // pass to the DT library

int N;                                  // number of sensors
DeviceAddress address[NDevices];        // space for addresses
DeviceAddress thisAddress;

void setup() {
    Serial.begin(57600);
    sensors.begin();

    // check for number of devices
    N = sensors.getDeviceCount();
    Serial.print(N, DEC);
    Serial.println(" sensor(s) detected.");

    // Find the address for each device
    for (byte j=0;j<NDevices;j++) {
        if (sensors.getAddress(thisAddress, j)) {
            for (byte k=0;k<8;k++) {
                Serial.print(thisAddress[k], HEX);
                Serial.print(" ");
                address[j][k] = thisAddress[k];
            }
            Serial.println("");
        }
    }
}

void loop() {
    // Do nothing.
    delay(1000);
}
```

Once all addresses are known, use this program to write them to the EEPROM on the Arduino. This just needs to happen once: after that the Arduino uses that EEPROM data to read all the temperature sensors in order.

```
/*
 * address_storage.ino
 *
 * Arduino code: run this ONCE.
 * Simply writes a list of addresses to the Arduino EEPROM, so that the
 * ThermoPlate.ino program can recall those addresses in the
 * correct order and send data to the computer for plotting.
 *
 */

#include <EEPROM.h>

byte j,k;               // Used for counting loops

// These are the addresses of the 100 sensors on the (current) board,
// in "page" order: the first 10 are the top row, the second 10 the
// next row, and so on.
// Note that this data is only valid for the particular 100 sensors
// on our particular board! You'll have to fill in your own data
// for your own apparatus.
byte addr[100][8] = {
{0x28, 0x0B, 0x15, 0xB0, 0x02, 0x00, 0x00, 0x96},
{0x28, 0x6C, 0x05, 0x0C, 0x03, 0x00, 0x00, 0x84},
... omit 96 lines of addresses here ...
{0x28, 0x80, 0x24, 0x0C, 0x03, 0x00, 0x00, 0xE5},
{0x28, 0x0D, 0xD9, 0xAF, 0x02, 0x00, 0x00, 0x85}
};

void setup() {

    for (j=0;j<100;j++) {   // Loop through 100 addresses
        for (k=0;k<8;k++) { // Loop through 8 bytes per address
            EEPROM.write((j*8)+k), addr[j][k]);
        }
    }

    // Let us know that something happened by flashing the LED.
    pinMode(13, OUTPUT);
    for (j=0;j<5;j++) {
        digitalWrite(13, HIGH);
        delay(200);
        digitalWrite(13, LOW);
        delay(200);
    }
}

void loop() {
    // Loop does nothing.
    delay(1000);
}
```

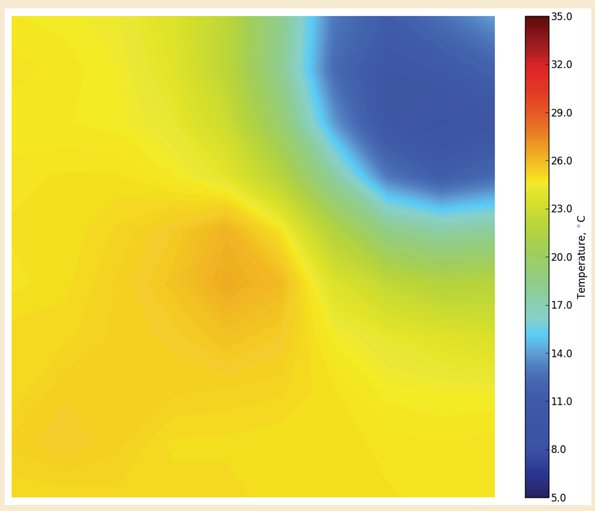
Unless you have a good infrared video camera handy, it's rather difficult to watch the Heat Equation in action. And even with that infrared camera, getting quantitative data is difficult. Thermocouples and other electronic theromometers give quantitative measurements, but it's hard to watch heat flow when you only measure temperature at certain points... Unless you measure temperature at a LOT of points!

This is where the DS18B20 comes in. The DS18B20 is a small 3-lead temperature sensor that can measure temperatures from -55°C to 125°C with a guaranteed accuracy of 0.5°C and precision of up to 0.02%. It communicates with microcontrollers via the "One-Wire" protocol, and —this is the useful part— *each DS18B20 has a unique hard-coded address.*

This unique address allows one to connect "many" sensors to a single microcontroller input pin, and the microcontroller can still measure the temperature of each sensor individually.

How many is "many"? The only limits are the amount of memory in the microcontroller and the amount of time you want to take to get a complete set of measurements. Here we used 100 sensors in a 10x10 grid: enough to get pretty good contour maps of temperature on a 30cm-square aluminum plate, but few enough that we can still get a complete data set every two seconds.

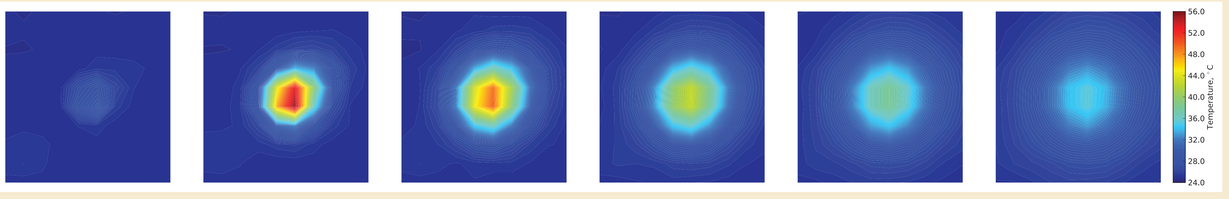
The result: quantitative two-dimensional heat flow measurements in real time.



One can use just about anything as a thermal source: it's sensitive enough that hot/cold drinks, ice cubes, hands, and even breathing on it can produce interesting results.

The image at the left is a single frame from a test run on the apparatus: there was a small bag of ice at the top right and a soldering iron near the center.

In addition to making pretty pictures like these, the computer can save numeric data for more complete analysis and for comparison with predictions of the Heat Equation. The sequence of images below comes from just such a data run: we hit the center of the plate with a propane torch for a few seconds then removed the torch and watched how the heat wave continued to propagate.



More details about the construction and programming of this device, including full video of the above dataset, can be viewed at http://hacks.ayars.org/2012/06/quantitative-two-dimensional.html

Once all the setup is done, the final Arduino program is pretty simple.

It repeatedly tells all sensors to make a measurement...

...then goes through the list of addresses...

...and asks each sensor what it measured...

...and sends that line of temperature data to the computer.

```
/*
 * ThermoPlate.ino
 *
 * This version reads the DS18B20 addresses from EEPROM, then reports
 * the temperatures on the serial line. Obviously the addresses must
 * be placed IN the EEPROM first, in the right order, via a program
 * such as address_storage.ino.
 *
 */

#include <OneWire.h>
#include <DallasTemperature.h>
#include <EEPROM.h>

#define NDevices 100                    // Number of DS18B20s
#define BUS 10                          // OneWire bus on pin 10

OneWire oneWire(BUS);
DallasTemperature sensors(&oneWire);    // pass to the DT library

DeviceAddress TAddress;                 // 8-bit T-sensor address
int EAddress;                           // EEPROM address pointer

void setup() {
    // Just get ready for communication. loop() does all the work.
    Serial.begin(57600);
    sensors.begin();
}

void loop() {

    // Send out a global "check temperatures" command
    sensors.requestTemperatures();

    // Now get those temperatures
    EAddress = 0;                       // Start at beginning
    for (byte j=0;j<NDevices;j++) {     // loop over devices
        // Retrieve address from EEPROM
        for (byte k=0;k<8;k++) {        // loop over address bytes
            TAddress[k] = EEPROM.read(EAddress);
            EAddress += 1;
        }
        // Now TAddress is the next sensor address, read it
        // and send the results on the serial line.
        Serial.print(sensors.getTempC(TAddress));
        Serial.print(" ");
    }
    // Now we're done with all sensors, so end the line.
    Serial.println("");
}
```

The data that reaches the USB port is just numeric data in text format, so anything that can read a serial port can take it from here. LabVIEW, IDL, C++, send it to the web, whatever you prefer! We used Python, because it's easy and open-source.

```
#!/usr/bin/env python
'''
T_reader.py
Program to capture and plot data from the 10x10 temperature sensor array.
'''

import serial, sys
from pylab import *

size = 12           # size of display window

try:
    minT = float(sys.argv[1])
    maxT = float(sys.argv[2])
except (ValueError, IndexError):
    print '''
Call program with three parameters:
    minimum expected temperature,
    maximum expected temperature,
    location of serial port.
'''
    exit(0)

# start the serial port
try:
    port = sys.argv[3]
    ser = serial.Serial(port, 57600, timeout=2)
except:
    print "Could not open port %s, exiting." % port
    exit(0)

# allow matplotlib animation
ion()

# set up matrix to receive temperature data
grid = zeros([10,10])

# read and discard a line to help get things in sync.
junk = ser.readline()

# open the view window
fig = figure(figsize=(size, size))

# define the levels at which to draw contours. The resolution
# of the sensors at 9-bit (default) sensitivity is about 0.12,
# so setting level spacing below this does not help.
levels = arange(minT, maxT, 0.12)

# now the main loop, which continues until an error occurs.
while True:

    try: # this is the stuff to do unless a problem occurs.

        # Read a line of temperature data from the Arduino
        line = ser.readline()

        # split the line into a list of floating-point values
        temperatures = [float(T) for T in line.split()]

        # sort the list of temperatures into a matrix for plotting
        for j in range(10):
            for k in range(10):
                grid[j,k] = temperatures[j*10+k]

        # plot the data
        contourf(grid, levels)
        draw()

    # Handle problems here
    except ValueError:
        # error at float conversion
        print "Conversion error: frame dropped, continuing collection."
        continue
    ... and so on ...
```

Why 2-D?

Ok, we confess: we did this in two dimensions because that's just *really cool*. It's probably more practical —even in an upper-division lab— to do thermal experiments in one dimension.

Typically, the one-dimensional heat-equation experiment is done with a piece of insulated copper pipe and a bunch of thermocouples. But by using this method of reading many DS18B20 sensors with a microcontroller, one can vastly improve that typical experiment. With just 25 of these sensors on a meter-long pipe, one can obtain 4cm spatial resolution and <1s temporaral resolution, at a cost of less than $100.