CSCI 1600 Final Project

Wireless Cooking Thermometer

Gaurav Manek
Fall 2016
Brown University

Introduction

I have an unusual hobby. I like to cook. Sometimes food, sometimes chocolate, sometimes even candy. Some of my favorite recipes, like caramel truffles, require very close control of the temperature. A consistent problem faced in the kitchen is that of accurate and continuous temperature monitoring. This project fills that need: it is a device that can monitor the temperature of food and candy as it is being cooked.

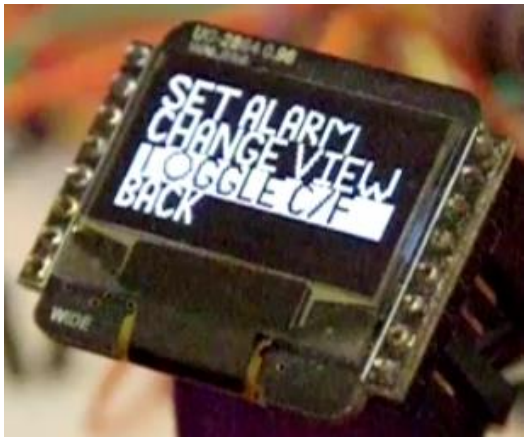
It comes in two parts, a base station and a wireless probe.

The wireless probe can be hooked onto the side of a cooking vessel, and the sensors will measure the temperature of the item directly. The wireless probe also has a status LED that “breathes” and changes color depending on the current temperature.

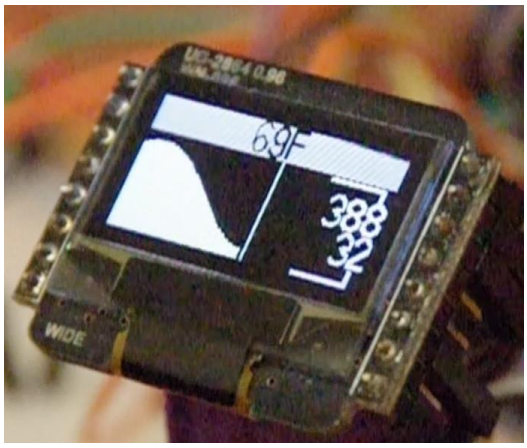
The separate base has a screen, a buzzer, and touch-free user interaction. This base station will display the current temperature, show on demand a graph of the temperature over time, and provide a temperature alarm that rings when the temperature goes above or below a user-settable threshold. When the alarm is triggered, the wireless probe’s LED flashes.

Features

Tells the Temperature!

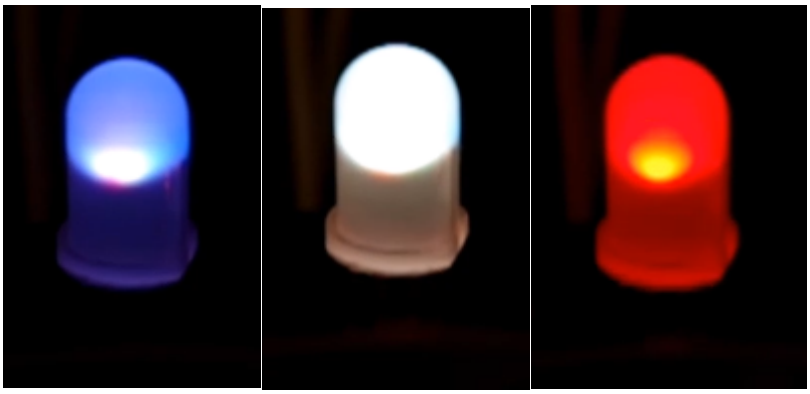


In Celsius and Fahrenheit



Graphs the temperature.
(Automatically scales the axis, too.)

Set an alarm



Status LED
Cold, Hot, Alarm.

Interaction

All interaction with the device is touch-free. Here's how you control it:

Step 1: Device displays Temperature/Graph

Step 2: Bring your hand close to get its attention. The menu is displayed, and it starts tracking your hand.

Step 3: Move your hand to select a menu item.

Step 4: Snap your fingers to select it.
(Or move your hand away to cancel selection.)

Architecture

Hardware

The hardware is divided into a base station and a remote sensor. The parts are as follows:

- Base Station (Figure 1)
    - Arduino Mega 2560, for processing.
    - SRF05 Ultrasonic Distance Sensor, to sense the user's hand position.
    - Microphone with a voltage comparator, to sense clicks.
    - Buzzer, for the alarm.
    - OLED screen, to communicate with the user.
    - XRF wireless radio, to communicate with the sensor.
- Remote Sensor (Figure 2)
    - Arduino
    - Two DS18B20 1-Wire Digital Temperature Sensors, to sense food temperature.
    - Tri-color LED, to show status to the user.
    - XRF wireless radio, to communicate with the base.

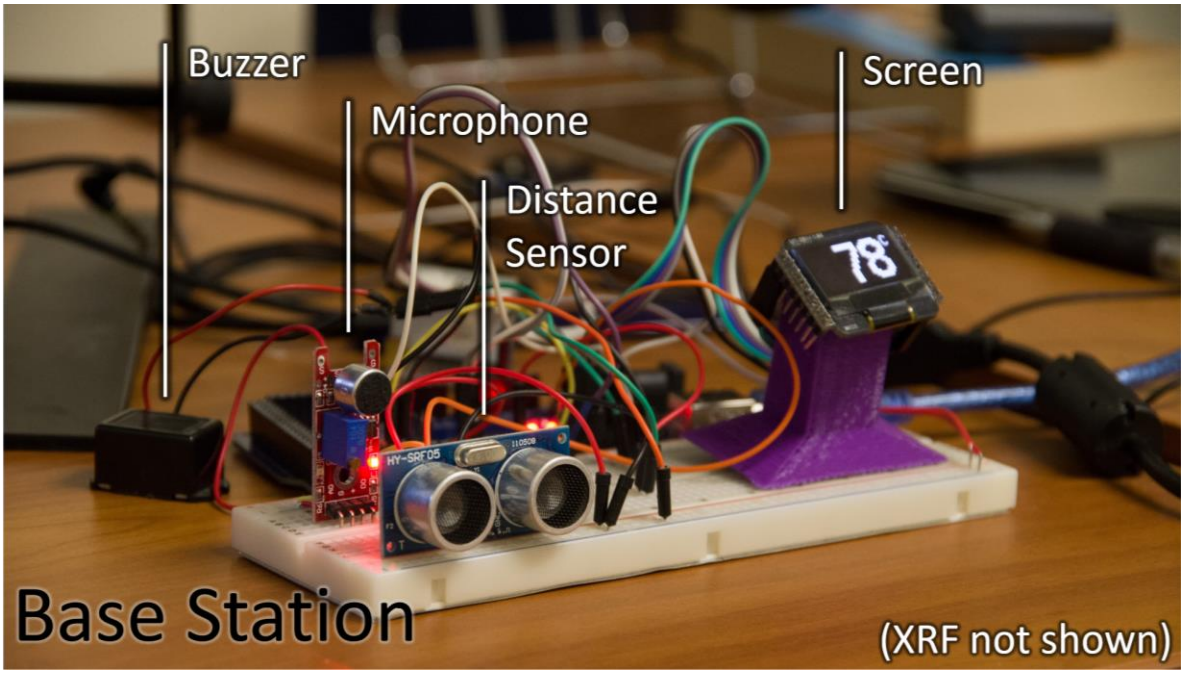


Figure 1: Picture of the Base Station

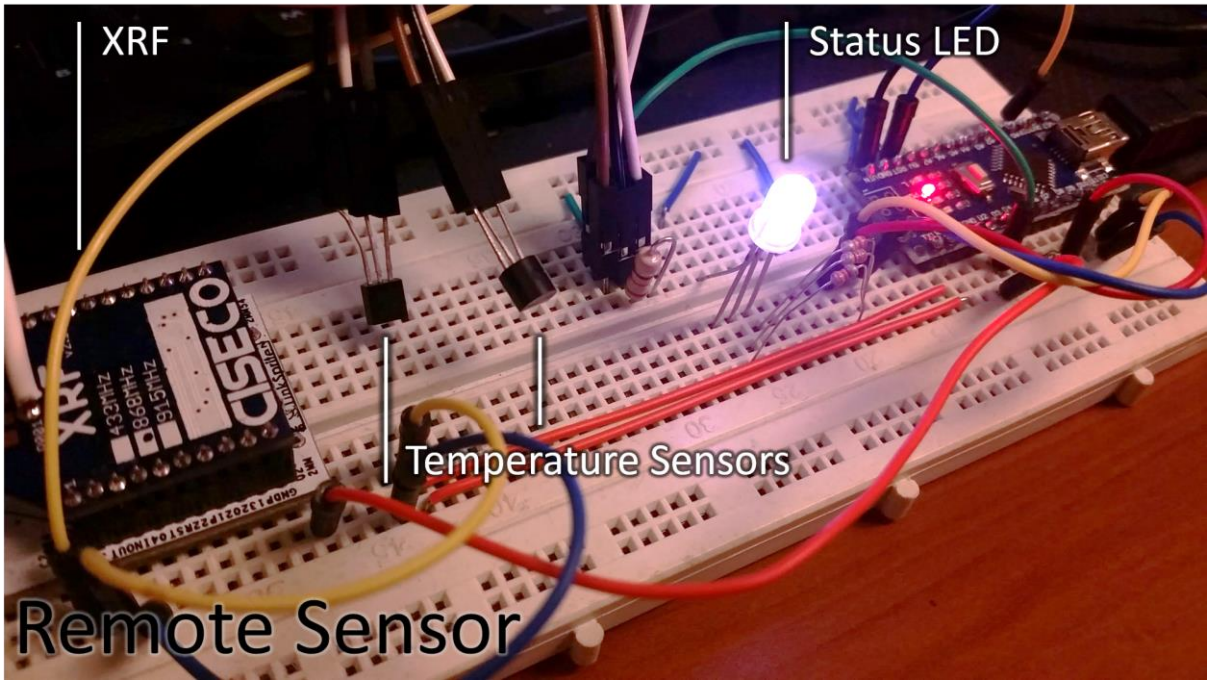


Figure 2: Picture of the Remote Sensor

State Diagram

The overall state diagram of the system is presented in Figure 3.

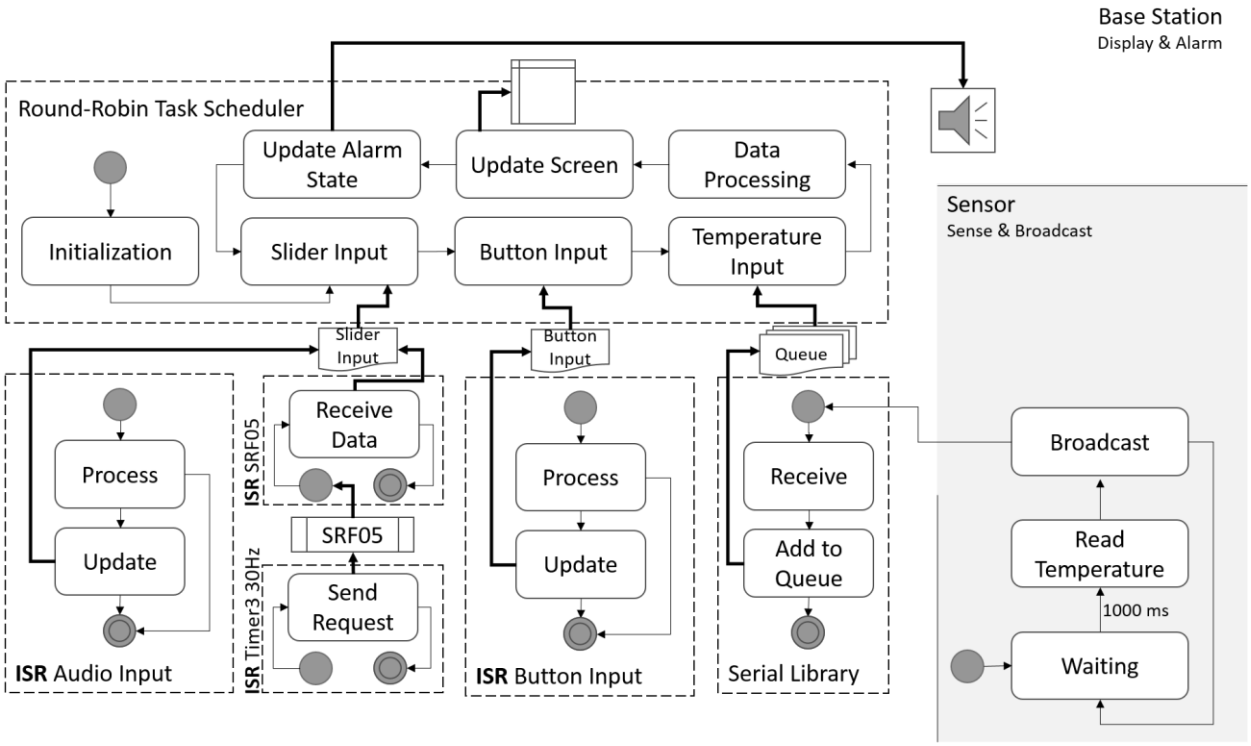


Figure 3: The Overall State Diagram

There are several interrupt-driven tasks running on the Arduino:

- The distance sensor is triggered 30 times a second.
- The distance sensor's response is handled using a pin change ISR that fires twice.
- The audio input ISR triggers whenever a loud sound is detected.

The remaining tasks are handled in the round-robin scheduler:

- Processing the distance and sound input to interpret user actions.
- Updating the temperature graph.
- Redrawing the screen.
- Updating the alarm state, including turning on and off the external buzzer.
- Sending LED state information.

The gray box on the lower right is an abridged state diagram of the sensor portion that excludes the LED updating, which occurs in its own interrupt service routine.

Code Structure

The code is broken into nice, reusable chunks using professional design patterns. For example, the main program handles user interaction through the `slider_*` functions, while those functions rely on the

`srfhandler_*` and `audiohandler_*` functions. No layer is aware of the structure of code in any other layer.

Another example are the `graphics_*` routines, which draw and compose the various parts of the interface. They depend on the `display_*` routines, which provide access to display hardware as though it were a memory buffer, which in turn depends on the communication support from `ssd1306_*`.

Implementation

Challenges

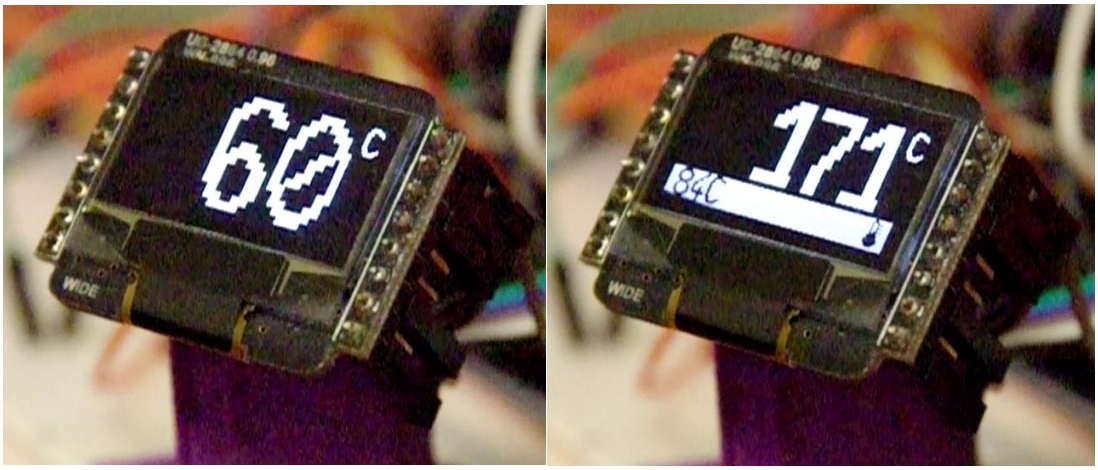
Interrupt-Safety

The software relies heavily on interrupts to handle various events – the ultrasonic distance sensor is triggered by an interrupt handler, and its signal is received by another. Incoming finger snaps are heard by a microphone and processed by an ISR.

All these interrupts need to communicate with the main thread, and they do so by raising flags and writing to shared memory in a manner that is thread safe. In fact, once the interrupts are set up in the initialization, the global interrupt flag is *never cleared* in the program.

Code Size

The final compiled code for the base station is only 14,560 bytes, including the font (which is 1,245 bytes by itself.) To keep the code size manageable only a single font is used and it is upscaled as necessary. Different parts of the user interface require different font sizes: the alarm bar at the bottom of the screen on the right uses the default font size. The larger number in the same image uses a font scaled by a factor of 3. The image on left is that of a font scaled by a factor of 4.



To reduce the aliasing caused by this upscaling, the font is slightly smoothed by blending adjacent columns. This helps maintain visual quality without requiring us to store larger fonts in ROM.

Performance

A great deal of effort went into performance of the system. The typical framerate achieved while rendering the graph is 19.3 Hz – and it's much faster when rendering just the temperature.

This was achieved through caching strategies while rendering both graph and upscaled fonts, and cleverly arranging the composition of each screen so that each screen pixel is touched at most once in each iteration.

Evaluation

It worked, as I hoped it would.

One limiting factor in the project is the temperature update rate. At the current rate (1Hz), the Serial library is well behaved, but at update rates of 2-5Hz, the Arduino is unable to keep up with the demand. This is due to the choice of protocol – sending plain text over serial is an excellent choice for debugging, but is unnecessarily verbose and not fault-tolerant. I would either write a simple binary protocol (with a checksum for verification), or I would use Google's ProtoBuf project[1] to generate such a protocol.

This limitation is perfectly fine for cooking, and the project serves its intended purpose very well.

The touch-free interaction with the device works surprisingly well, and it's a very novel way to set alarms and monitor temperature curves with hands covered in flour and chocolate, and it is generally quite user friendly. It certainly beats out the laminated membrane switches that are on virtually every kitchen appliance.

Related Work

I don't see any products that fulfill this particular need.

There are products that are much simpler, such as infrared temperature guns, that measure temperature on demand. There are also products that are much more sophisticated, like automatic grilling machines (that measure moisture content as well as temperature), or stove controllers (that rotate the stove's control knob to maintain the desired temperature).

One particular product is meld, a pair of devices that connect a temperature sensor and a stove knob controller. Their Kickstarter page has some technical details.

[1] https://github.com/google/protobuf

Future Work

Low-power support

I planned on including support for low-power modes so that the sensor can be run off a battery. This would require replacing the power-hungry XRF with an nRF201 or similar 2.4 GHz communication chip. Additionally, the software would have to be modified to use the low-power modes to conserve battery when not being used.

IR Thermocouple

The original plan called for a TI TMP007 Infrared Thermocouple, which is able to sense temperature at a distance. This would allow me to detect temperatures up to 250C (for candy and sugar), when the current sensors (Dallas DS18B20) are only safe up to 150C (for creams, chocolate and crepes).

Unfortunately, the first thermocouple was faulty, and the replacement arrived too late to be included in the project.

Attribution

All code in the base station part of the project is by me, though I did use some data from other projects:

- Font design and rendering code from previous work by me.
    - HT1632 for Arduino library
- Matplotlib colormap 'Magma' by Nathaniel J. Smith and Stefan van der Walt
    - mpl colormaps
    - Released under CC0 license
- SSD1306 command codes from work by Adafruit
    - Adafruit_SSD1306
    - BSD License
    - I only used the command codes, not the entire library.

For the remote sensor part of the project, I used some code from external sources:

- OneWire Library by Paul Stoffregen, Jim Studt, et. al.
    - OneWire Library
    - Released to public domain
- Arduino Library for Maxim Temperature Integrated Circuits by Miles Burton, Tim Newsome, et. al.
    - Arduino Temperature Control Library
    - Released under the GNU Lesser General Public License