IoT Integrated Smart Home
Project Final Report

Justin Yang justiny2
Advaith Ravikumar advaith2
Xutao (Jumbo) Jiang xutaoj2

ECE 110/120 Honors Lab
May 8, 2017

1 Introduction

1.1 Statement of Purpose

Our project is a smart home security system that helps notify you of unwanted activity around the valuables in your home. Many people have important things to keep care of, so this smart security system helps them to ensure that their items aren't stolen or tampered with while they are away outside the room or home. This project was inspired, in part, by the high tech security systems that might be seen in movies, especially the laser security system. With that in mind, we set out to build a security system that could have cool-looking functionalities while at the same time being something that could be easily used in one's day-to-day life. We also wanted our project to be connected, so a user could check in on the status of their home in a couple of clicks from their phone or computer.

Our project consists of IR emitter–detector pairs which detect when something is blocking their line of sight, a trip laser which changes a photoresistor's operating point when blocked, and a temperature sensor. The system uses a RedBoard microcontroller to monitor the conditions of the sensors. When activity is detected, signals are sent from a Bluetooth module to a smartphone, where the app allows you to view alerts and system conditions.

1.2 Features and Benefits

Since the system has each component connected individually to the Arduino microcontroller and Bluetooth module, then the app is able to display individual information about the condition of each component. For instance, if the alarm goes off, you could see which one of the components had an issue from the app. The system is also very sensitive, so it can detect changes for a short period of time. For instance, it could detect your slow moving turtle blocking the sensor, or your very agile cat.

The app displays information about whether each of the sensors is blocked, and what the current reading of the temperature sensor is. The app also displays information about the overall health of the system, which is determined by the AND gate logic.

2 Design

2.1 System Overview

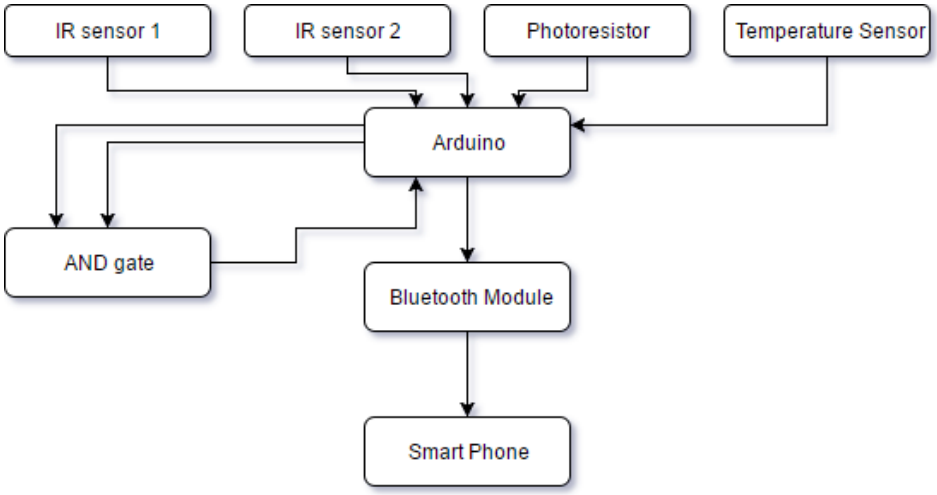

Figure 1: Complete block diagram

2.2 Design Details

2.2.1 IR sensors

We got two IR sensors, each has one emitter and one receiver set side by side, to detect the movement of the door. The receiver will change resistance while irradiated with infrared light. We put power to both of them using analog input to detect the voltage, and set a threshold to make a 0/1 signal to indicate whether the infrared light is blocked or not.

2.2.2 AND gate

The AND gate is a digital logic control for the IR sensors. In our design, we used the quad two-input AND gate (SN74S08). Digital inputs to the AND gate represent the status of each of the IR sensor pairs. A digital high means that the system is good; a digital low is presented when an object is blocking the path between the IR detector and emitter. Hence, the two-input AND gate will yield a 1 when both IR sensors are unobstructed.

2.2.3 Photoresistor

This gadget will change resistance according to the light intensity of the side that it is facing. We also use analog input to receive signal, and we set it to a relatively high threshold so that only the laser beam could make the feedback above the threshold, while the room light won't, thus allowing absence of the laser to trigger the alarm.

2.2.4 Temperature Sensor MCP9808

This breakout chip could detect temperature and send back digital signal using I2C protocol. It is decent. But the library code Adafruit provides will stop the whole program if the sensor

is not detected.

2.2.5 Simblee Bluetooth Module

It needs to be hooked up to a separate programming header module so that we could connect it to the computer and program on it. It is quite easy after we figure out how it works. We could use the same IDE as the Arduino, while selecting Simblee as construction mode after downloading the required files. One of the problems we had was using `sprintf` on the decimal float value of the temperate we acquired from the Arduino, but after a lot of debugging, we figured out that we could use `%d.%d` to get around the trouble. Also, we have to use the app of Simbee to connect to the module, and the app development is kind of limited.

2.2.6 Simblee for Mobile

The smartphone app and Arduino library that we have to use to connect to the Bluetooth module. The Arduino library has decent functionality, but the documentation is not written very well. After a while, however, we were able to find the source files for the library which allowed us to find out how to create UI objects such as text boxes and colorful rectangles.

3 Results

3.1 Characterization of Sensors

IR Sensors – Range ~3 cm
LDR – Darkness – 0.40 MΩ
With Laser – 200 Ω
Bluetooth Module – Range ~5 m (straight line)

4 Problems and Challenges

The main problem that we faced was in getting our Bluetooth module to work correctly. At first, we had a few issues finding pins to solder it nicely, then found out that we needed another adaptor board. After this, getting the data to display perfectly was a bit of a struggle. The phone app wasn't updating the temperature and kept crashing. Eventually we were able to get it to update and print the correct values by changing the data type in our sprintf statement from float to two integers manually separated by a decimal sign. Calibrating the sensors was another smaller problem. During transportation, several wires had come loose and needed to be reconnected. Moreover, moving the IR emitter–receiver even one row on the breadboard required recalibration of the threshold signals in the arduino code. The LDR also had to be configured to the right level so as to differentiate between the laser and ambient lighting. Moreover, the responsiveness of this subcircuit was initially low, which was fixed by reducing the delay in all other parts of the code. More information is now passed through the circuit and Bluetooth module due to lesser delays, but the sensitivity of the entire security system increased which is more important in practical usage.

5 Future Plans

We would definitely like to implement our project in real world scenarios. To do this, we need to give more control in the other direction, i.e., from the phone to the IoT system. This would entail lights to turn on/off with the push of a button, or be able to toggle your thermostat at home and have the temperature actually change. Our Bluetooth range was acceptable for our small scale model, but we would need to extend this to Wi-Fi if this project is to be taken further. We also want to configure the system to send a text message to your phone when the alarm goes off.

6 References

[1]"sprintf - C++ Reference", *Cplusplus.com*, 2017. [Online]. Available: http://www.cplusplus.com/reference/cstdio/sprintf/. [Accessed: 08- May- 2017].

[2]"C library function sprintf()", *www.tutorialspoint.com*, 2017. [Online]. Available: https://www.tutorialspoint.com/c_standard_library/c_function_sprintf.htm. [Accessed: 08- May- 2017].

[3]"Simblee", *Simblee.com*, 2017. [Online]. Available: https://www.simblee.com/. [Accessed: 08- May- 2017].

[4]"Simblee BLE - SparkFun Electronics", *Sparkfun.com*, 2017. [Online]. Available: https://www.sparkfun.com/simblee. [Accessed: 08- May- 2017].

[5]C. Willenborg, "PCB info", 2016. [Online]. Available: https://docs.google.com/document/d/1igjXyXyJiLFourtGIShI42v7llou9brmnvKJY4AG5do/. [Accessed: 08- May- 2017].

[6]"SparkFun Simblee BLE Breakout - RFD77101 - WRL-13632 - SparkFun Electronics", *Sparkfun.com*, 2017. [Online]. Available: https://www.sparkfun.com/products/13632. [Accessed: 08- May- 2017].

[7]"SparkFun USB to Serial Breakout - FT232RL - BOB-12731 - SparkFun Electronics", *Sparkfun.com*, 2017. [Online]. Available: https://www.sparkfun.com/products/12731/. [Accessed: 08- May- 2017].

[8]2017. [Online]. Available: https://cdn.sparkfun.com/datasheets/IoT/Simblee%20RFD77101%20Datasheet%20v1.0.pdf. [Accessed: 08- May- 2017].

[9]"sparkfun/SparkFun_Simblee_Breakout_Board", *GitHub*, 2017. [Online]. Available: https://github.com/sparkfun/SparkFun_Simblee_Breakout_Board. [Accessed: 08- May- 2017].

[10]2017. [Online]. Available: https://cdn.sparkfun.com/datasheets/IoT/Simblee%20User%20Guide%20v2.05.pdf. [Accessed: 08- May- 2017].

[11]"sparkfun/Simblee_Tutorials", *GitHub*, 2017. [Online]. Available: https://github.com/sparkfun/Simblee_Tutorials. [Accessed: 08- May- 2017].

7 Code Appendix

Arduino_Security.ino

```
#include <Wire.h>
#include "Adafruit_MCP9808.h"

#define Laser    A0
#define GreenLed A1
#define BlueLed  A2
#define ANDOut   9
#define digiOut1 10
#define digiOut2 11
#define LaserLED 12
#define digiOut0 12
// Create the MCP9808 temperature sensor object
Adafruit_MCP9808 tempsensor = Adafruit_MCP9808();

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(digiOut1,OUTPUT);
  pinMode(digiOut2,OUTPUT);
  pinMode(ANDOut, INPUT);
  pinMode(digiOut0,OUTPUT);
  pinMode(LaserLED,OUTPUT);
  // Make sure the sensor is found, you can also pass in a
   ↪ different i2c
  // address with tempsensor.begin(0x19) for example
  if (!tempsensor.begin()) {
    Serial.println("Couldn't find MCP9808!");
    while (1);
  }
}

int BlueRead,GreenRead, Out, LaserRead;
int PWMsig;
```

```
33 void loop() {
34   // Temp Sensor
35   // Serial.println("wake up MCP9808.... "); // wake up MSP9808 -
     ↪ power consumption ~200 mikro Ampere
36   // tempsensor.wake(); // wake up, ready to read!
37
38   // Read and print out the temperature, then convert to *F
39   float c = tempsensor.readTempC();
40   float f = c * 9.0 / 5.0 + 32;
41   // Serial.println("Temp: "); Serial.print(c);
     ↪ Serial.print("*C\t");
42   // Serial.print(f); Serial.println("*F");
43
44   PWMsig = (int)(c*10*(255.0/1000.0));
45   analogWrite(3,PWMsig);
46   // Serial.println(PWMsig);
47   // Serial.println("Shutdown MCP9808.... ");
48   // tempsensor.shutdown(); // shutdown MSP9808 - power
     ↪ consumption ~0.1 mikro Ampere
49
50   delay(10);
51
52   //Door IR Sensor Code
53   BlueRead=analogRead(BlueLed);
54   GreenRead=analogRead(GreenLed);
55   delay(10);
56   if(GreenRead > 370) {
57     digitalWrite(digiOut1,HIGH);
58   }
59   else {
60     digitalWrite(digiOut1, LOW);
61   }
62   // Serial.println(BlueRead);
63   if (BlueRead>545) {
64     digitalWrite(digiOut2, HIGH);
65   }
66   else {
67     digitalWrite(digiOut2, LOW);
68   }
69   Out = digitalRead(ANDOut);
70   //  Serial.println(Out);
71
72   //Laser Code
73       LaserRead=analogRead(Laser);
74       Serial.println(LaserRead);
```

```
75        if(LaserRead>900){//should be 900
76          //lights are on
77          digitalWrite(digiOut0, LOW);
78          digitalWrite(LaserLED, LOW);
79        }
80        else{
81          digitalWrite(digiOut0, HIGH);
82          digitalWrite(LaserLED, HIGH);
83        }
84  }
```

Security_System.ino

```
1  // Modified from Sparkfun Simblee Tutorials
2  // https://github.com/sparkfun/Simblee_Tutorials
3
4  // To use the SimbleeForMobile library, you must include this
   ↪  file at the top
5  // of your sketch. **DO NOT** include the SimbleeBLE.h file, as
   ↪  it will cause
6  // the library to silently break.
7  #include <SimbleeForMobile.h>
8  #include <Wire.h>
9
10 #define analogPin 11
11 #define diginput 12
12
13 const int led = 2; // The Simblee BOB (WRL-13632) has an LED on
   ↪  pin 2.
14 int ledState = LOW;
15
16 // uint8_t object ids
17
18 uint8_t btnID;
19 uint8_t switchID;
20 uint8_t textID;
21
22 uint8_t boxID;
23 uint8_t boxIR;
24 uint8_t boxLED;
25
26 const int btn = 9; // The Simblee BOB (WRL-13632) has a button on
   ↪  pin 3.
27
28 double frequency;
29
```

```
30 char buf[9];
31 int counter;
32
33 void setup()
34 {
35   pinMode(diginput, INPUT);
36   counter = 0;
37
38   Wire.beginOnPins(12, 15);
39
40   pinMode(led, OUTPUT);
41   pinMode(analogPin, INPUT);
42   digitalWrite(led, ledState);
43
44   // Protip: using INPUT_PULLUP very rarely causes any problems
     ↪ but can solve
45   // a lot of problems with input signals that aren't pulled
     ↪ strongly.
46   pinMode(btn, INPUT_PULLUP);
47
48   // advertisementData shows up in the app as a line under
     ↪ deviceName. Note
49   // that the length of these two fields combined must be less
     ↪ than 16
50   // characters!
51   SimbleeForMobile.deviceName = "Meme";
52   SimbleeForMobile.advertisementData = "Security";
53
54   // txPowerLevel can be any multiple of 4 between -20 and +4,
     ↪ inclusive. The
55   // default value is +4; at -20 range is only a few feet.
56   SimbleeForMobile.txPowerLevel = -4;
57
58   // This must be called *after* you've set up the variables
     ↪ above, as those
59   // variables are only written during this function and changing
     ↪ them later
60   // won't actually propagate the settings to the device.
61   SimbleeForMobile.begin();
62   Serial.begin(9600);
63   Serial.println(btn);
64 //  buf[2] = '.';
65 //  buf[5] = 0;
66 }
67
```

```
68 void loop()
69 {
70   bool laser_alert = digitalRead(diginput);
71   Serial.println(laser_alert);
72   double onTime = pulseIn(analogPin, HIGH);
73   frequency = onTime / 32.;
74   frequency = 1.58231*frequency + 1.20509;
75   int freq = frequency;
76   int frac = ((int) (frequency * 100)) % 100;
77   Serial.println(sprintf(buf, "%d.%02d oC", freq, frac));
78   buf[6] = 176;
79   //Serial.print("Frequency: ");
80   //Serial.println(frequency);
81   //Serial.print("buf: ");
82   //Serial.println(buf);
83   //Serial.println(1.58231*frequency + 1.20509);
84   // All we want to do is detect when the button is pressed and
     ↪ make the box on
85   // the screen white while it's pressed.
86
87   // This is important: before writing *any* UI element, make
     ↪ sure that the UI
88   // is updatable!!! Failure to do so may crash your whole
     ↪ program.
89   if (SimbleeForMobile.updatable)
90   {
91     // Okay, *now* we can worry about what the button is doing.
       ↪ The
92     // updateColor() function takes the id returned when we
       ↪ created the box and
93     // tells that object to change to the color parameter passed.
94     if (digitalRead(btn) && !laser_alert)
       ↪ SimbleeForMobile.updateColor(boxID, BLACK);
95     else {
96       Serial.println("angery");
         ↪ SimbleeForMobile.updateColor(boxID, RED);
97     }
98     if (digitalRead(btn)) {
99       SimbleeForMobile.updateColor(boxIR, GREEN);
100    }
101    else {
102      SimbleeForMobile.updateColor(boxIR, BLACK);
103    }
104    if (!laser_alert) {
105      SimbleeForMobile.updateColor(boxLED, GREEN);
```

```
106       }
107       else {
108         SimbleeForMobile.updateColor(boxLED, BLACK);
109       }
110
111
112       counter++;
113       if (counter > 25) {
114         SimbleeForMobile.updateText(textID, buf);
115         counter = 0;
116       }
117
118
119     }
120     else { Serial.println("SAD!"); }
121     // This function must be called regularly to process UI events.
122     SimbleeForMobile.process();
123
124     //delay(1000);
125     //fflush(buf);
126 }
127
128 // (15.55, 25.81), (19.62, 32.25)
129
130 // ui() is a SimbleeForMobile specific function which handles the
    ↪ specification
131 // of the GUI on the mobile device the Simblee connects to.
132 void ui()
133 {
134     // color_t is a special type which contains red, green, blue,
      ↪ and alpha
135     // (transparency) information packed into a 32-bit value. The
      ↪ functions rgb()
136     // and rgba() can be used to create a packed value.
137     color_t darkgray = rgb(85,85,85);
138
139     // These variable names are long...let's shorten them. They
      ↪ allow us to make
140     // an interface that scales and scoots appropriately regardless
      ↪ of the screen
141     // orientation or resolution.
142     uint16_t wid = SimbleeForMobile.screenWidth;
143     uint16_t hgt = SimbleeForMobile.screenHeight;
144
```

```
145    // The beginScreen() function both sets the background color
       ↪ and serves as a
146    // notification that the host should try to cache the UI
       ↪ functions which come
147    // between this call and the subsequent endScreen() call.
148    SimbleeForMobile.beginScreen(darkgray);
149
150    // SimbleeForMobile doesn't really have an kind of indicator-
       ↪ but there IS a
151    // drawRect() function, and we can freely change the color of
       ↪ the rectangle
152    // after drawing it! The x,y coordinates are of the upper left
       ↪ hand corner.
153    // If you pass a second color parameter, you'll get a fade from
       ↪ top to bottom
154    // and you'll need to update *both* colors to get the whole box
       ↪ to change.
155    boxID = SimbleeForMobile.drawRect(0, 0, wid, hgt, BLACK);
156  //  boxID = SimbleeForMobile.drawRect(
157  //                           (wid/2) - 50,        // x position
158  //                           (hgt/2) + 75,        // y positon
159  //                           1000,                 // x dimension
160  //                           1000,                 // y
     ↪ dimensionrectangle
161  //                           BLACK);              // color of
     ↪ rectangle.
162
163    SimbleeForMobile.drawText((wid/2) - 75, (hgt/2) - 125, "IR",
       ↪ WHITE, 30);
164    boxIR = SimbleeForMobile.drawRect(
165                             (wid/2) - 75,        // x position
166                             (hgt/2) - 75,        // y positon
167                             25,                 // x dimension
168                             25,                 // y
                                ↪ dimensionrectangle
169                             BLACK);              // color of
                                ↪ rectangle.
170
171    SimbleeForMobile.drawText((wid/2) + 40, (hgt/2) - 125, "LED",
       ↪ WHITE, 30);
172    boxLED = SimbleeForMobile.drawRect(
173                             (wid/2) + 50,        // x position
174                             (hgt/2) - 75,        // y positon
175                             25,                 // x dimension
```

```
176                                 25,                 // y
                                    ↪ dimensionrectangle
177                                 BLACK);             // color of
                                    ↪ rectangle.
178
179     // Create a button slightly more than halfway down the screen,
        ↪ 100 pixels
180     // wide, in the middle of the screen. The last two parameters
        ↪ are optional;
181     // see the tutorial for more information about choices for
        ↪ them. The BOX_TYPE
182     // button has a bounding box which is roughly 38 pixels high by
        ↪ whatever the
183     // third parameter defines as the width.
184     btnID = SimbleeForMobile.drawButton(
185                                 (wid/2) - 75,           // x
                                    ↪ location
186                                 (hgt/2) - 22 + 150,         // y
                                    ↪ location
187                                 150,                    // width of
                                    ↪ button
188                                 "Reverse LED",          // text
                                    ↪ shown on button
189                                 WHITE,                  // color of
                                    ↪ button
190                                 BOX_TYPE);              // type of
                                    ↪ button
191
192     // Buttons, by default, produce only EVENT_PRESS type events.
        ↪ We want to also
193     // do something when the user releases the button, so we need
        ↪ to invoke the
194     // setEvents() function. Note that, even though EVENT_PRESS is
        ↪ default, we
195     // need to include it in setEvents() to avoid accidentally
        ↪ disabling it.
196     SimbleeForMobile.setEvents(btnID, EVENT_PRESS | EVENT_RELEASE);
197
198     // Create a switch above the button. Note the lack of a title
        ↪ option; if you
199     // want to label a switch, you'll need to create a textBox
        ↪ object separately.
200     // A switch's bounding box is roughly 50 by 30 pixels.
201     switchID = SimbleeForMobile.drawSwitch(
```

```
                                    (wid/2) - 25,           // x
                                    ↪ location
                                    (hgt/2)+22 + 150,           // y
                                    ↪ location
                                    BLUE);                  // color
                                    ↪ (optional)

  SimbleeForMobile.drawText(wid/2-125, 50, "IoT Security
  ↪ System!", WHITE, 30);

  textID = SimbleeForMobile.drawText(wid/2-65, hgt/2, buf, WHITE,
  ↪ 45);
  //textID = SimbleeForMobile.drawText(wid/2-45, hgt/2-250,
  ↪ 1.58231*frequency + 1.20509, WHITE, 45);

  SimbleeForMobile.endScreen();
}

// This function is called whenever a UI event occurs. Events are
↪ fairly easy
// to predict; for instance, touching a button produces a
↪ "PRESS_EVENT" event.
// UI elements have default event generation settings that match
↪ their expected
// behavior, so you'll only rarely have to change them.
void ui_event(event_t &event)
{
  // We created the btnID and switchID variables as globals, set
  ↪ them in the
  // ui() function, and we'll use them here.


  if (event.id == btnID)
  {
    if (event.type == EVENT_PRESS)
    {
      if (ledState == HIGH) digitalWrite(led, LOW);
      else digitalWrite(led, HIGH);
    }
    if (event.type == EVENT_RELEASE)
    {
      if (ledState == HIGH) digitalWrite(led, HIGH);
      else digitalWrite(led, LOW);
    }
  }
```

```
237
238     // If the event was a switch press, we want to toggle the
        ↪ ledState variable
239     // and then write it to the pin.
240     if (event.id == switchID)
241     {
242       if (ledState == HIGH) ledState = LOW;
243       else ledState = HIGH;
244       digitalWrite(led, ledState);
245     }
246   }
```