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

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

Figure 1: Complete block diagram
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 temperature 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 Simbee 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


7 Code Appendix

Arduino_Security.ino

```c
#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;
```c
void loop() {
    // Temp Sensor
    // Serial.println("wake up MCP9808.... "); // wake up MSP9808 -
    // power consumption ~200 mikro Ampere
    // tempsensor.wake(); // wake up, ready to read!

    // Read and print out the temperature, then convert to *F
    float c = tempsensor.readTempC();
    float f = c * 9.0 / 5.0 + 32;
    // Serial.println("Temp: "); Serial.print(c);
    // Serial.print("*C\t");
    // Serial.print(f); Serial.println("*F");

    PWMsig = (int)(c*10*(255.0/1000.0));
    analogWrite(3,PWMsig);
    // Serial.println(PWMsig);
    // Serial.println("Shutdown MCP9808.... ");
    // tempsensor.shutdown(); // shutdown MSP9808 - power
    // consumption ~0.1 mikro Ampere

    delay(10);

    //Door IR Sensor Code
    BlueRead=analogRead(BlueLed);
    GreenRead=analogRead(GreenLed);
    delay(10);
    if(GreenRead > 370) {
        digitalWrite(digiOut1,HIGH);
    }
    else {
        digitalWrite(digiOut1, LOW);
    }
    // Serial.println(BlueRead);
    if (BlueRead>545) {
        digitalWrite(digiOut2, HIGH);
    }
    else {
        digitalWrite(digiOut2, LOW);
    }
    Out = digitalRead(ANDOut);
    // Serial.println(Out);

    //Laser Code
    LaserRead=analogRead(Laser);
    Serial.println(LaserRead);
}
```
if(LaserRead>900){
  //should be 900
  //lights are on
  digitalWrite(digiOut0, LOW);
  digitalWrite(LaserLED, LOW);
}
else{
  digitalWrite(digiOut0, HIGH);
  digitalWrite(LaserLED, HIGH);
}
}

Security_System.ino

// Modified from Sparkfun Simblee Tutorials
// https://github.com/sparkfun/Simblee_Tutorials

// To use the SimbleeForMobile library, you must include this
// file at the top
// of your sketch. **DO NOT** include the SimbleeBLE.h file, as
// it will cause
// the library to silently break.
#include <SimbleeForMobile.h>
#include <Wire.h>

#define analogPin 11
#define diginput 12

const int led = 2;  // The Simblee BOB (WRL-13632) has an LED on
// pin 2.
int ledState = LOW;

// uint8_t object ids

uint8_t btnID;
uint8_t switchID;
uint8_t textID;

uint8_t boxID;
uint8_t boxIR;
uint8_t boxLED;

const int btn = 9;  // The Simblee BOB (WRL-13632) has a button on
// pin 3.

double frequency;
char buf[9];
int counter;

void setup()
{
  pinMode(diginput, INPUT);
  counter = 0;

  Wire.beginOnPins(12, 15);

  pinMode(led, OUTPUT);
  pinMode(analogPin, INPUT);
  digitalWrite(led, ledState);

  // Protip: using INPUT_PULLUP very rarely causes any problems
  // but can solve
  // a lot of problems with input signals that aren’t pulled
  // strongly.
  pinMode(btn, INPUT_PULLUP);

  // advertisementData shows up in the app as a line under
  // deviceName. Note
  // that the length of these two fields combined must be less
  // than 16
  // characters!
  SimbleeForMobile.deviceName = "Meme";
  SimbleeForMobile.advertisementData = "Security";

  // txPowerLevel can be any multiple of 4 between -20 and +4,
  // inclusive. The
  // default value is +4; at -20 range is only a few feet.
  SimbleeForMobile.txPowerLevel = -4;

  // This must be called *after* you’ve set up the variables
  // above, as those
  // variables are only written during this function and changing
  // them later
  // won’t actually propagate the settings to the device.
  SimbleeForMobile.begin();
  Serial.begin(9600);
  Serial.println(btn);
  // buf[2] = ".";
  // buf[5] = 0;
}
```cpp
void loop()
{
    bool laser_alert = digitalRead(diginput);
    Serial.println(laser_alert);
    double onTime = pulseIn(analogPin, HIGH);
    frequency = onTime / 32.;
    frequency = 1.58231*frequency + 1.20509;
    int freq = frequency;
    int frac = ((int) (frequency * 100)) % 100;
    Serial.println(sprintf(buf, "%d.%02d oC", freq, frac));
    buf[6] = 176;
    //Serial.print("Frequency: ");
    //Serial.println(frequency);
    //Serial.print("buf: ");
    //Serial.println(buf);
    //Serial.println(1.58231*frequency + 1.20509);
    // All we want to do is detect when the button is pressed and
    // make the box on
    // the screen white while it’s pressed.

    // This is important: before writing *any* UI element, make
    // sure that the UI
    // is updatable!!! Failure to do so may crash your whole
    // program.
    if (SimbleeForMobile.updatable)
    {
        // Okay, *now* we can worry about what the button is doing.
        // The
        // updateColor() function takes the id returned when we
        // created the box and
        // tells that object to change to the color parameter passed.
        if (digitalRead(btn) && !laser_alert)
            SimbleeForMobile.updateColor(boxID, BLACK);
        else
        {
            Serial.println("angery");
            SimbleeForMobile.updateColor(boxID, RED);
        }
        if (digitalRead(btn)) {
            SimbleeForMobile.updateColor(boxIR, GREEN);
        }
        else {
            SimbleeForMobile.updateColor(boxIR, BLACK);
        }
        if (!laser_alert) {
            SimbleeForMobile.updateColor(boxLED, GREEN);
        }
    }
}
```
}  
else {
    SimbleeForMobile.updateColor(boxLED, BLACK);
}

counter++;
if (counter > 25) {
    SimbleeForMobile.updateText(textID, buf);
    counter = 0;
}
}

else { Serial.println("SAD!"); }
// This function must be called regularly to process UI events.
SimbleeForMobile.process();

//delay(1000);
//fflush(buf);

// (15.55, 25.81), (19.62, 32.25)

// ui() is a SimbleeForMobile specific function which handles the
// specification
// of the GUI on the mobile device the Simblee connects to.
void ui() {
    // color_t is a special type which contains red, green, blue,
    // and alpha
    // (transparency) information packed into a 32-bit value. The
    // functions rgb()
    // and rgba() can be used to create a packed value.
    color_t darkgray = rgb(85,85,85);

    // These variable names are long...let’s shorten them. They
    // allow us to make
    // an interface that scales and scoots appropriately regardless
    // of the screen
    // orientation or resolution.
    uint16_t wid = SimbleeForMobile.screenWidth;
    uint16_t hgt = SimbleeForMobile.screenHeight;
// The beginScreen() function both sets the background color
// and serves as a
// notification that the host should try to cache the UI
// functions which come
// between this call and the subsequent endScreen() call.
SimbleeForMobile.beginScreen(darkgray);

// SimbleeForMobile doesn’t really have an kind of indicator-
// but there IS a
// drawRect() function, and we can freely change the color of
// the rectangle
// after drawing it! The x,y coordinates are of the upper left
// hand corner.
// If you pass a second color parameter, you’ll get a fade from
// top to bottom
// and you’ll need to update *both* colors to get the whole box
// to change.
boxID = SimbleeForMobile.drawRect(0, 0, wid, hgt, BLACK);

// boxID = SimbleeForMobile.drawRect(
//   (wid/2) - 50, // x position
//   (hgt/2) + 75, // y position
//   1000, // x dimension
//   1000, // y
t // dimensionrectangle
//   BLACK); // color of
t // rectangle.

SimbleeForMobile.drawText((wid/2) - 75, (hgt/2) - 125, "IR",
// WHITE, 30);
boxIR = SimbleeForMobile.drawRect(
  (wid/2) - 75, // x position
  (hgt/2) - 75, // y positon
  25, // x dimension
  25, // y
t // dimensionrectangle
  BLACK); // color of
t // rectangle.

SimbleeForMobile.drawText((wid/2) + 40, (hgt/2) - 125, "LED",
// WHITE, 30);
boxLED = SimbleeForMobile.drawRect(
  (wid/2) + 50, // x position
  (hgt/2) - 75, // y positon
  25, // x dimension
176  // y
dimensionrectangle
BLACK);   // color of
defectangle.

178 // Create a button slightly more than halfway down the screen, 100 pixels
// wide, in the middle of the screen. The last two parameters are optional;
// see the tutorial for more information about choices for them. The BOX_TYPE
// button has a bounding box which is roughly 38 pixels high by whatever the
// third parameter defines as the width.
btnID = SimbleeForMobile.drawButton(
    (wid/2) - 75,  // x
    (hgt/2) - 22 + 150,  // y
    150,  // width of
    "Reverse LED",  // text
    "White",  // color of
    BOX_TYPE);  // type of

186 // Buttons, by default, produce only EVENT_PRESS type events.
// We want to also
// do something when the user releases the button, so we need to invoke the
// setEvents() function. Note that, even though EVENT_PRESS is default, we
// need to include it in setEvents() to avoid accidentally disabling it.
SimbleeForMobile.setEvents(btnID, EVENT_PRESS | EVENT_RELEASE);

190 // Create a switch above the button. Note the lack of a title option; if you
// want to label a switch, you’ll need to create a TextBox object separately.
// A switch’s bounding box is roughly 50 by 30 pixels.
switchID = SimbleeForMobile.drawSwitch(
SimbleeForMobile.drawText(wid/2-125, 50, "IoT Security System!", WHITE, 30);

textID = SimbleeForMobile.drawText(wid/2-65, hgt/2, buf, 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);
        }
    }
// If the event was a switch press, we want to toggle the
// ledState variable
// and then write it to the pin.
if (event.id == switchID)
{
    if (ledState == HIGH) ledState = LOW;
    else ledState = HIGH;
    digitalWrite(led, ledState);
}