**Automated Light Switch with Socket Communication**

ECE 4220 – Real-time Embedded Systems

Professor Luis Rivera

Created by:

Andrew Sherrod

**Abstract**

This project used two infrared sensors in order to detect the movement of a person’s body through a door. It was a very small scale test and assumed that a person would take at least five seconds when moving from one side of the door to the other. This also communicated with any client through a TCP connection, and would send the client the count when the count had been altered. The main component of the program was the Raspberry Pi which acted as the server and also manipulated the light and read incoming information from an Arduino Uno. The Arduino read in the information from the infrared sensors and informed the Pi when one of the sensors detected a “body.” When the body then moved inside the door to the next sensor, a signal was sent and the light was triggered.

**Introduction**

This project was able to create a socket connection with an external computer and monitor the input from the Arduino Uno in order to tell if a person is leaving or entering the room. If a person was going from inside the room to the outside of the room it would decrement the global count of the occupants in the room. If it were the opposite scenario and a person were entering the room from the outside to the inside it would increment the count and turn the light emitting diode (LED), that we used to simulate our light source for the room, on. It would also notify the client that was connected to it of the change in the count. This program was designed in order to help homeowners conserve power by monitoring the occupants in each room and allow them to see whether the light is on. If the light is on then the user knows that there is both someone in the room and they are consuming power with the light being turned on. The main usefulness of this project is the automation of the light control. How often do you forget to turn off the light when you are in a hurry to leave the room? More than likely fairly often. This would waste energy and cost you a higher utility bill in your home. With this project, you would not be responsible for controlling the lights with a switch. This device would control it for you. The main goal of this project was to successfully automate this entire process and take the responsibility off of your mind.

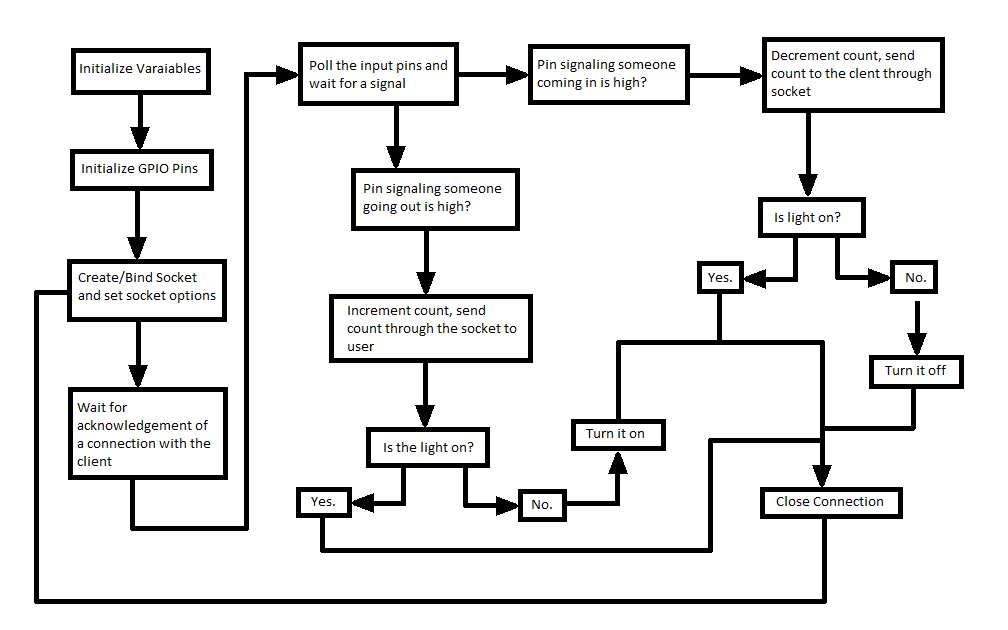
**Background**

This project was mainly used in order to make this process more hands-free, and hopefully keep the only information the user sees would be the number of people currently in the room. While researching similar projects all I could find were lights that would turn on and would continuously poll if there was still movement in the room in order to stay on. That is to say if you sat still long enough within the room, the light would once again turn off and you would be forced to stand up and activate it once again. This was not the only rendition I came across. Another was implemented using your smart phone. This is, however, still prone to the same power consumption problems I mentioned earlier. Again, the main goal of this project was to be entirely automated and keep the system hands free.

This project contains many different resources that were compiled together in order to for it to function properly. The foremost resource would the WiringPi library for the Raspberry Pi. The WiringPi library enables a programmer on the raspberry pi to easily access the General Purpose Input/Output (GPIO) pins in order to read and write values. This library made digital reading and writing on the pi extremely simple. This library was used in reading in the digital signals sent from the Arduino Uno and writing the pin to the control the LED too high in order to operate the light. The second resource I used was a tutorial that gave examples how to write TCP connections using sockets with the raspberry pi. It was provided by Smith College and is readily available online. Using their code as a template, I modified it to fit my needs and only have the server send any information through the socket. Lastly, I used the forums on the

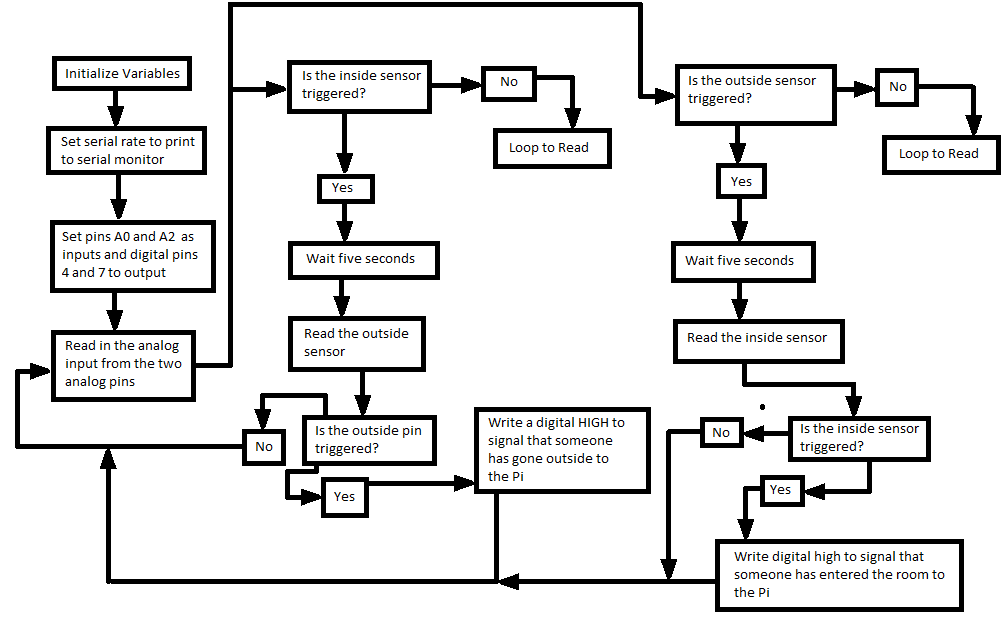
**Proposed Method / System Description / Implementation**

The end product that I produced was similar to the model I proposed. However, the addition of the Arduino Uno was one that I did not foresee when working on this project. Firstly, I still used the Raspberry Pi as the main source of software. The system begins with setting up the GPIO pins using the WiringPi library and designating them as either input or output respectively. Secondly, it will begin creating the socket and binding it. When the socket is created it will wait for an acknowledgement from the client. After this client has sent its reply and the connection has been made successfully, the program will wait for the status of the pins to come in through the Arduino. When a pin is set to high, this is the signal from the Arduino that someone has entered or exited the room. Next, if a person has entered the room the program will increment the count and send the signal containing the count to the client. If a person has exited the room it decrements the count and sends that information to the client. After informing the client, the light status is checked. If the count is greater than zero and the light was not already on, the light was that turned on. If the count was equal to zero and the light was currently on, the light was turned off. On the following page there is a diagram that represents this logic in a flow chart.



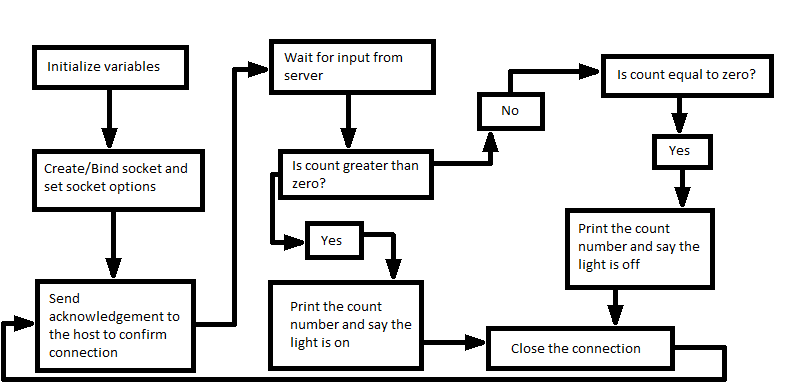
One of the most difficult things about setting up this part of the software was attempting to send multiple messages to the client. I ended up having to close the connection and reopen it every time I would send a message to the client. For whatever reason when I could send the first and second message fine without closing the connection. However, when I attempted to send the third message the pi would get stuck in a loop and continuously send garbage data. This was my reason for closing and reestablishing the connection every time a message was sent.

For the Arduino logic it was much simpler. I first initialized the variables and designated the desired pins to input and output. Next, set the serial rate so that I was able to monitor the output of the Arduino through the serial port. After that, I read in the analog data from the analog-to-digital converter pins on the Arduino. This information is read in on a scale from 0 to 1024 and varies based on the amount of voltage being delivered to the pin. The infrared sensors I used were only able to put out enough voltage for the reading to reach anywhere from 650 – 715. The closer an object got to the sensor, the higher the output voltage. Using these values I assigned a threshold of 500 to determine if a “person” was standing above the sensor and about to enter or exit the room. If the reading was greater than 500 on either sensor then there would be a delay of a five seconds. After this delay the second sensor would be read again and have its value checked. If the value of that pin was then above 500, it could be determined that a person had entered or exited the room. After determining that someone had entered the room, the Arduino would then write a digital HIGH output using another one of its digital pins. This signal was the one being read from the raspberry pi. Depending on the signal the raspberry pi received, it could determine whether someone had entered or exited the room. A diagram of the logic for the Arduino can be seen on the following page.



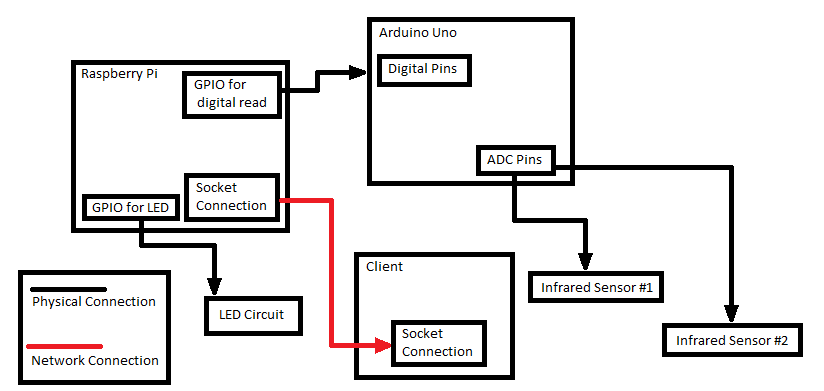
The sole reason for including the Arduino in the project was due to the analog-to-digital pins. After what I thought was a successful reading using the pins on the Pi, I was proven to be incorrect. I came to find out that the Pi did not actually have any ADC capabilities. At this point it was late enough in the project timeline that I did not have time to order another part in order to make this happen. Instead I improvised and used the ADC ports on the Arduino. Though this did take away my ability to use pthreads in monitoring the input from the sensors, it is still a working prototype. However, it cannot handle through traffic very quickly due to being strictly a serial program.

The final implementation I will discuss in my program was that of the client. This was by far the simplest branch of the program to implement. It began by initializing its variables as always. Next, it created the socket and bound it. After the socket had been create, it send an acknowledgement to the Pi in order to confirm it had connected successfully. Lastly, it waited for information to be passed to it using the read function as a blocking function and then printed that data to the screen depending on the count. This blocking function waited until it received any input from the socket before proceeding with the rest of the program. A logic diagram of the client code can be seen on the next page.



As you can tell by the diagram the logic here was much simpler. As I stated previously, the connection was closed each time to avoid the Pi sending a large amount of garbage data. After closing the connection and reopening it again, that issue was resolved and both the client and server worked as intended.

Overall this project has many working parts. The brain of the project being the Raspberry Pi and the Arduino assigning somewhat with the use of the ADC ports. The client is merely a listener for information through the socket. In order to better understand the overall view of the project, the diagram below shows the interconnections of the programs and hardware components.



**Experiments and Results**

With so many interconnected parts in this program it was easier for me to start with a small building in each section and then piece it all together in the end. For the first experiment, I ran a simple program that would flash the LED on the board from the Raspberry Pi. This test was run simply to ensure that I knew how to turn the LED on and off when the sensors called for it. This test was extremely simple. The second test was slightly more complex. I tested the Raspberry Pi in order to see if I could read in analog input. After what I thought was a positive result, this test proved to be a failure. After realizing this test failed. I moved on to reading the analog input using the Arduino. After successfully printing the values of the sensors to the serial monitor this test was successful. It took a few attempts due to me wiring up the sensors incorrectly the first time through. The third test I ran was the test on the server/client TCP code. I had not written any programs using the TCP model as opposed to the broadcast UDP model we used in the lab, so this was new to me. This took me a while to understand. I first tested the program from the tutorial simply to ensure that I could, in fact, get the two programs to communicate with one another. After many faulty attempts I realized what the error was. When I was attempting to obtain the hostname and connect using a specified IP address, the network here at my apartment complex gives me a generic IP address of 127.0.1.1. However, when typing “hostname –I” into the Raspberry Pi terminal it showed the true host name. After hardcoding the newly provided IP into the client program so that it could contact the Pi, the program ran successfully. The program simply sent numbers from the client to the server and the server multiplied them and sent them back. The final individual test I ran before placing any components together was to see if I could write a digital HIGH on a pin after the detection on the sensors. The logic for the sensors was a problem for me when I started writing the code. The logic I followed was that it would normally take a person a few seconds if they had to open the door, move through it, and close it on the opposite side. So, naturally, there would be a delay in between when they tripped the sensors. If I accounted for this delay and assumed it to be five seconds I can check for one sensor and when it is tripped, I can sequentially check the other one five seconds later. After both sensors had been tripped, I tested writing the digital HIGH through a port on the Arduino and this worked successfully by writing a logical HIGH out on a specified pin.

The next experiments all had to deal with placing the pieces together and forming the overall project. The first tasks I decided to place together were the sockets and the LED. Using a scanf() function to fake the input from the sensors I set up my if statements and turned the LED on and off when the count was greater than zero and equal to zero, respectively. My first attempt at this was successful and this proved to be fairly easy once I had figured out the instructions for the LED. The last, and most difficult test, was introducing the sensor inputs into the program. This took many attempts as I had many flaws in my logic while coding this section. Eventually this test proved to be successful and the logic ended up working out.

There were cases that I tested with the program as a whole. I tested to see what would happen if I ran the client without the server to accept it. It ended up continuously attempting to reconnect to the server over and over again. Another case I tested was if the count was above one, and then decrementing the count to ensure the light stayed on. This cased turned out to be successful as the light stayed on even when the count went from two to one. The last test case I attempted was if the count tried to decrement passed zero. It should always just stay at zero, never go below it. The first time I tested this, the logic was faulty and the count managed to go into negative values. This was quickly sorted out with a few if-statements. I tested the project multiple times giving many different inputs to ensure that it was running as intended when the light would turn on and off at the correct times. The program has a turn-around time of around 7 - 9 seconds to where it can begin to detect the next person entering the room. This was the biggest issue with my program, was how slow it was running. This could be solved with a different implementation, which is discussed in the conclusion.

**Discussion and Conclusion**

Almost all of the results from each of my experiments were expected. The most bizarre result I encountered was that of sending multiple messages through the socket. I’m still not entirely sure why it would only allow me to send two messages at a time and then begin to continuously send garbage. It would send garbage without any regard for any other input as well. It would be locked in a continuously loop of sending nothing but random text. The solution to the problem that I came up with is probably one of the most inefficient ways of deal with the problem. However, the task then performed as it was originally intended. Another set of results that did not make sense was when I was attempting to read analog data without realizing it on the Raspberry Pi. The fact that the results came through as positive on one run makes it seem like it was just dumb luck. Every run after that was met with unsuccessful attempts, but this all makes sense due to the lack of ADC pins.

Some of the other problems that I encountered throughout the project were just strictly logic errors. The most difficult of the logic errors I came across was when programming the Arduino to read in the analog input and then determine whether someone had entered or exited the room. I had written in my if-statements and then checked the opposite sensor with a nested if-statement. However, when I would attempt to check the second sensor, I would never see it reach a threshold of 500 and cause the second if-statement to trigger. I was confused by this for some time until I realized I needed to do another read of the second sensor before I actually checked it within the if-statement. This seems trivial now when I look back on it, but at the time it was such a small thing that it was easy to miss and overlook. Another problem I had while working on this project was a simple problem of wiring up the hardware components correctly and reading in the values from these components. I first read the datasheet for the infrared sensors incorrectly and confused the pins while wiring the sensors into the Arduino. After finally realizing I connected the pins to the incorrect place, there was the problem of labelling the input ports correctly. On the Arduino, if you are using the ADC pins, you must signify that when you are setting up the variables that are to be used. Instead of saying “sensorPin = 1” you must clarify to the Arduino that the sensor is going to be an ADC port by setting the value as follows: “ADCPin = A1.” Without the “A” the value is merely one of the digital I/O pins on the Arduino. Before learning this, I was confused as to why my sensors were not reading in any data whatsoever. They would simply print out the value of the port which I had set them to. For instance if I were using port three, all the Serial monitor would receive when I printed the value read from port three would simply be “3.” This was frustrating for some time, but after realizing it was a simple logic error it was easily fixed.

Through this project I have learned many things about using socket communication, and also learned to use TCP/IP communication as opposed to the UDP connection we had previously used in class. I learned that in order extra hardware is required if you are planning on using analog-to-digital conversion with the Raspberry Pi, and that the pins on the Raspberry Pi are extremely sensitive. When I first started working with the GPIO pins on the Pi, I was attempting to use alligator clips to connect them with the wires on the breadboard. I quickly came to find out that these clips were much too big to use on the GPIO pins. If the pins so much as clipped a second pin while I was placing them on my desired pin, the entire Raspberry Pi would force restart. This makes sense in hindsight since all of the pins on the Pi carry various values and if you shorted any of the wires together, it could possibly damage the board.

My solution to this problem is, sadly, more than likely one of the less efficient implementations. The various delay and wait statements slow the program down significantly and do not help when attempting to quickly track people as they enter and leave the room. If I were able to accomplish the proposed version of my project, it would more than likely run more smoothly. The solution I proposed used two separate pthreads to monitor the sensors and would set a flag for a few seconds and continue to monitor their sensor for a potential second person. After a certain amount of time, say the five seconds that my Arduino waits, it would set the flag back to low. If the other sensor had been tripped while the flag was high from the other sensor, it would know that someone had entered or exited the room. This would greatly increase the efficiency of the program by being able to continue to monitor one sensor while the second sensor was also collecting data. Another improvement to the program would be being able to send more than two messages through the socket at a time. If I were able to keep a continuous connection without being forced to close it every write, it would greatly improve the efficiency of the program and the wait time.

This project overall turned out to be quite successful. Not only was I able to make the process completely hands-free, I was able to get each component of the project working correctly. This was a great challenge for me, and caused me quite a bit of confusion on occasion. This project took around two and a half weeks to complete when I was working on it in small increments. I’m extremely happy that this project was successful, but slightly disappointed that I was unable to implement it in the way that I had proposed.

**Appendices**

1. **Raspberry Pi Code:**

#include <wiringPi.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <unistd.h>

void sendData(int sockfd, int x){

int n;

//declare buffer to store the data to send

char buffer[32];

//store the data into the buffer to be sent

sprintf(buffer, "%d\n", x);

//send the data through the specified socket and check for errors

if((n = write(sockfd, buffer, strlen(buffer))) < 0){

printf("error writing to socket\n");

}

buffer[n] = '\0';

}

int main(int argc, char \*argv[]){

//declare variables

int sockfd, newsockfd, portno = 51717, clilen;

char buf[250]; char \*myIP;

struct sockaddr\_in serv\_addr, cli\_addr;

int n;

int data, incdec, count = 0;

struct hostent \*myStruct;

struct in\_addr \*\*addr\_list;

int lowFlag = 0; int highFlag = 0;

//setup the wiringPi GPIO pin setup and set which pins will be inputs and outputs

wiringPiSetup();

pinMode(0, OUTPUT);

pinMode(1, INPUT);

pinMode(2, INPUT);

int inside = 0, outside = 0;

//inform the user what port number we are using

printf("using port #: %d\n", portno);

//set socket options and check for errors on creation

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if(sockfd < 0){

printf("sock create err\n");

}

//clear a buffer that will serve as the server address

bzero((char \*) &serv\_addr, sizeof(serv\_addr));

//set the specified fields in order to begin transmission

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

serv\_addr.sin\_port = htons(portno);

//bind the socket and check for errors

if(bind(sockfd, (struct sockaddr \*) &serv\_addr, sizeof(serv\_addr)) < 0){

printf("error on bind\n");

}

//listen for a connection

listen(sockfd, 100);

clilen = sizeof(cli\_addr);

//loop forever

while(1){

printf("client accept...\n");

//wait for the handshake from the connection and check for errors on the accept

if((newsockfd = accept(sockfd, (struct sockaddr \*) &cli\_addr, (socklen\_t \*) &clilen)) < 0){

printf("error on accept\n");

}

printf("reopened, polling\n");

//loop until the status of a sensor has been changed

while(1){

//read the status of both sensors from the Arduino

inside = digitalRead(1);

outside = digitalRead(2);

//if the inside pin has been tripped, this means a person has entered the room

if(inside){

//increment the count and send the new count the the client

count++;

sendData(newsockfd, count);

break;

}

//if the outside pin has been tripped, this means a person has exited the room

else if(outside){

//decrement the count and ensure that it never goes below zero

count--;

if(count < 0){

count = 0;

}

//transmit the new count to the client

sendData(newsockfd, count);

break;

}

}

//if the count is equal to zero and the light is on, turn the light off

if(count == 0 && lowFlag == 0){

digitalWrite(0, LOW);

lowFlag = 1;

highFlag = 0;

}

//if the count is greater than zero and the light is current off, turn the light off

if(count > 0 && highFlag == 0){

//turn light off

digitalWrite(0, HIGH);

//set high flag to show the light is currently on

highFlag = 1;

//set low flag to show that the light is not off

lowFlag = 0;

}

//wait for the connection to close and then loop back to the accept to wait for another handshake from the client

delay(1000);

}

return 0;

}

1. **Client Code:**

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <netdb.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <arpa/inet.h>

int getData( int sockfd ) {

//declare variable, create a buffer to store the received information

char buffer[32];

int n;

//read information from the server

n = read(sockfd,buffer,31);

//append an exit character at the end of the message

buffer[n] = '\0';

//return the message to the main program

return atoi( buffer );

}

int main(int argc, char \*argv[])

{

//declare variables

int sockfd, portno = 51717, n;

//hardcode IP address of server

char serverIp[] = "100.65.7.82";

struct sockaddr\_in serv\_addr;

struct hostent \*server;

char buffer[256];

int count;

//loop forever

while(1){

//set socket options

sockfd = socket(AF\_INET, SOCK\_STREAM, 0) ;

//set the server IP address

server = gethostbyname( serverIp );

//clear a buffer for the server address information

bzero( (char \*) &serv\_addr, sizeof(serv\_addr));

//set the necessary options and copy the information obtained from the host name

serv\_addr.sin\_family = AF\_INET;

bcopy( (char \*)server->h\_addr, (char \*)&serv\_addr.sin\_addr.s\_addr, server->h\_length);

serv\_addr.sin\_port = htons(portno);

//send connection request and confirmation to the server

connect(sockfd,(struct sockaddr \*)&serv\_addr,sizeof(serv\_addr));

//read the count from the server, this is a blocking function and the program will not proceed without input from the server

count = getData(sockfd);

//if the count is greater than zero print the message

if(count > 0){

//let the user know the light is on and what the current count is

printf("current count: %d -> the light is on.\n", count);

//close the socket connection

close(sockfd);

//allow time for the connection to close

sleep(3);

}

//if the count is equal to zero print the message

else if(count == 0){

//inform the user that the light is off and what the current count is

printf("current count: %d -> the light is off.\n", count);

//close the socket connection

close(sockfd);

//allow time for the connection to close

sleep(3);

}

}

return 0;

}

1. **Arduino Uno Code**

/\*declare variables\*/

const int sensorPin = A0;

const int sensorPin1 = A2;

const int InPin1 = 7;

const int OutPin1 = 4;

int var = 0, var2 = 0;

void setup() {

// Start the serial transmission and set which pins will be input and output

// the ADC pins require no specification

Serial.begin(9600);

pinMode(InPin1, OUTPUT);

pinMode(OutPin1, INPUT);

}

void loop() {

//read in the analog data from both of the infrared sensors

var = analogRead(sensorPin);

var2 = analogRead(sensorPin1);

//print the information read from the sensors to the serial monitor for debugging purposes

Serial.print("sensor in: ");

Serial.println(var);

Serial.print("sensor out: ");

Serial.println(var2);

//if the outside sennsor is tripped execute the following

if(var >= 500){

//wait for five seconds and then read the inside sensor

delay(5000);

var2 = analogRead(sensorPin1);

//if the inside sensor is tripped execute the following

if(var2 >= 500){

//print that someone is entering the room

Serial.println("GOING IN");

//write digital high to signal to the raspberry pi

digitalWrite(InPin1, HIGH);

//wait a half a second to allow the Pi to pick up the change in signal state

delay(500);

//set the pin back to low

digitalWrite(InPin1, LOW);

}

}

//if the inside pin has been tripped execute the following

if(var2 >= 500){

//wait five seconds and read the outside pin

delay(5000);

var = analogRead(sensorPin);

//if the outside pin has been tripped perform the following

if(var >= 500){

//print that someone is leaving the room

Serial.println("GOING OUT");

//write a high signal tot he pi to show that someone is going out of the room

digitalWrite(OutPin1, HIGH);

//wait half a second to allow for time to pick up the change in state

delay(500);

//set the pin low again

digitalWrite(OutPin1, LOW);

}

}

//wait one second between data sample collection

delay(1000);

}