Project Title:  Knock/Tap enabled Security and Alarm System

## Goal:

To be able to use particular knocks for Home/Enterprise Security Systems as well as a SOS alarm, which will Blink according to the SOS mores code for a particular knock in case of emergency situations within a perimeter.

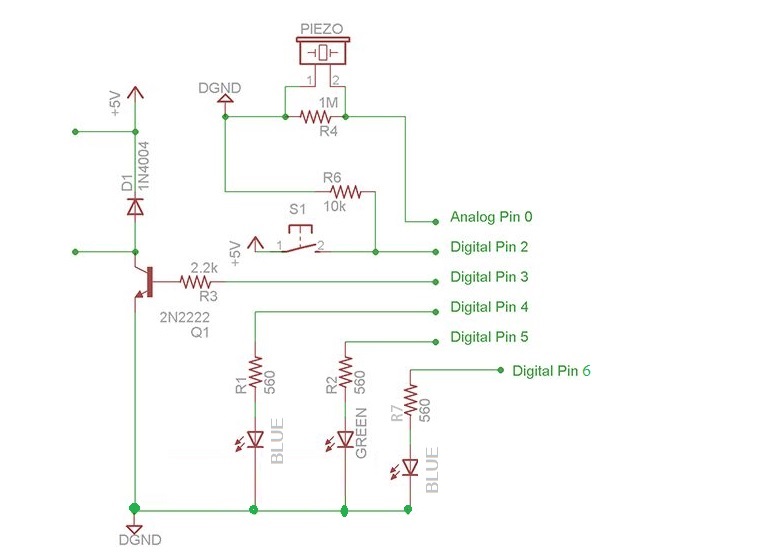
## Requirements:

1. Arduino Uno
2. Rotating Motor
3. Piezo Sensor CF-A1
4. 2 Blue LED
5. 1 Green LED
6. 1 NPN Transistor P2N2222A like [these](http://www.oomlout.co.uk/transistors-p2n222a-x10-p-197.html) or [these](http://www.solarbotics.com/products/tr2222/) (or [similar](http://www.sparkfun.com/commerce/product_info.php?products_id=521)).
7. 1 Rectifier Diode (1N4001 or similar) [this](http://www.solarbotics.com/products/d3/) or [this](http://www.sparkfun.com/commerce/product_info.php?products_id=8589) will do.
8. 1 2.2k ohm resistor (1/4 watt)
9. 1 10k ohm resistor (1/4 watt)
10. 1 1M ohm resistor (1/4 watt)
11. 3 560 ohm resistor (Or whatever will run your red and green LED's at 5v. [How to tell.](http://www.instructables.com/id/Choosing-The-Resistor-To-Use-With-LEDs/))
12. 1 small piece of perf board or Breadboard. 5x15 holes or longer. ([example](http://www.westfloridacomponents.com/HW097/Solderable+Perf-Board+SMALL+Copper+Pad+Circuit+Board.html))
13. 1 9 volt battery clip and 9v battery. (Or any other way you can think of to get 7-12v to the Arduino. [A wall adapter like this](http://www.adafruit.com/index.php?main_page=product_info&cPath=17_22&products_id=63&zenid=6e0700cd9c501ca6e2210cb14d5134e7) is a great option so you don't have to worry about batteries running out.
14. Connector wire. Male-Male,Male-Female.

## Procedure:

It would be advised that before proceeding you make sure all the components are functioning properly. After doing so, we shall do the following steps in order :

### Circuit:



1. **Wire the Piezo Sensor**  
     
   Connect it between Analog pin 0 and the ground. Attach the 1M ohm resistor between Analog pin 0 and the ground.

Test -      With your Arduino plugged into the computer via USB, open the serial monitor window, you should see the text “Program start”. Tap the piezo sensor and you should see “knock starting” each time you tap it.

1. **Wire the LED**

Connect the blue LED to digital pin 4 ,green LED to digital pin 5 as well as a 3rd blue LED to digital pin 6 with their corresponding 560\* ohm resistors in line.

Test - Powering on the circuit should make the green LED light up, if that is not the case, the LED might be connected the other way around. Every tap will make the green LED follow your tap where entering the correct sequence the LEDs should blink a few times, if not, the blue one should.

1. **Set up the programming toggle switch**

Connect one side of the button to +5v. The other pin on the button connect to digital pin 2 and, with a 10K resistor to the ground.

Test - Toggling the switch should allow the blue LED to light up. At this moment a new sequence can be added, but first you need to do the original sequence first, this is for validation purposes, if the sequence is correctly entered, the user will be able to add a new tap code, after which the green and blue LEDs should alternate, indicating that the new code is now ready for use, which will be stored and used later when needed.

1. **Wire the motor up**   
     
    Connect the motor with the diode in parallel.

Test - Tap the default sequence, the motor should rotate, again if it doesn’t rotate, check to see if the diode pins are the other way around

**5. Program the Arduino:**

This section I assume that you know how to connect your Arduino microcontroller to you computer, compile and upload a sketch. The codes are in the codes section. Just copy and paste the code on the sketch.(as the code is too long it is given at the bottom of the report.)  
  
We're going to upload our sketch before doing any of the electronics so we can test the electronics as we go.

## Features:

### -Knock detection:

This is the major feature of the system. Users have ability to make household/company edifice more secure by using this system as it requires a tap code to proceed onto the next location. The tap code has to be exact, in order to unlock doors/gates

When turned on the green LED is always on but blinks each time a knock is registered. If the correct knock sequence is entered the motor rotates confirming that it has been unlocked and the green light blinks while a wrong input sequence triggers the blue LED to blink.

### -Setting a new knock pattern:

This allows the system to be programmable, i.e whenever the switch is pushed, the blue LED lights up and the green LED turns off indicating that we are now in programming mode. But to gain access the current knock sequence needs to be entered which causes the green and blue LEDs to light up in sequentially.

Now we're in programming mode, a new tap sequence can be set and after the pattern is registered, the green and blue LEDs will light up in a confirmation sequence. Now the new knock pattern has been set!

### -Entering SOS mode:

The system has a default SOS knock sequence embedded within it's code which enables the user to enter the SOS knock sequence thus triggering the SOS signal which is a 3rd blue LED which lights up following the SOS pattern. This can be used as morse code to indicate that the user is in some sort of trouble.

The SOS pattern is " . . . - - - . . ."

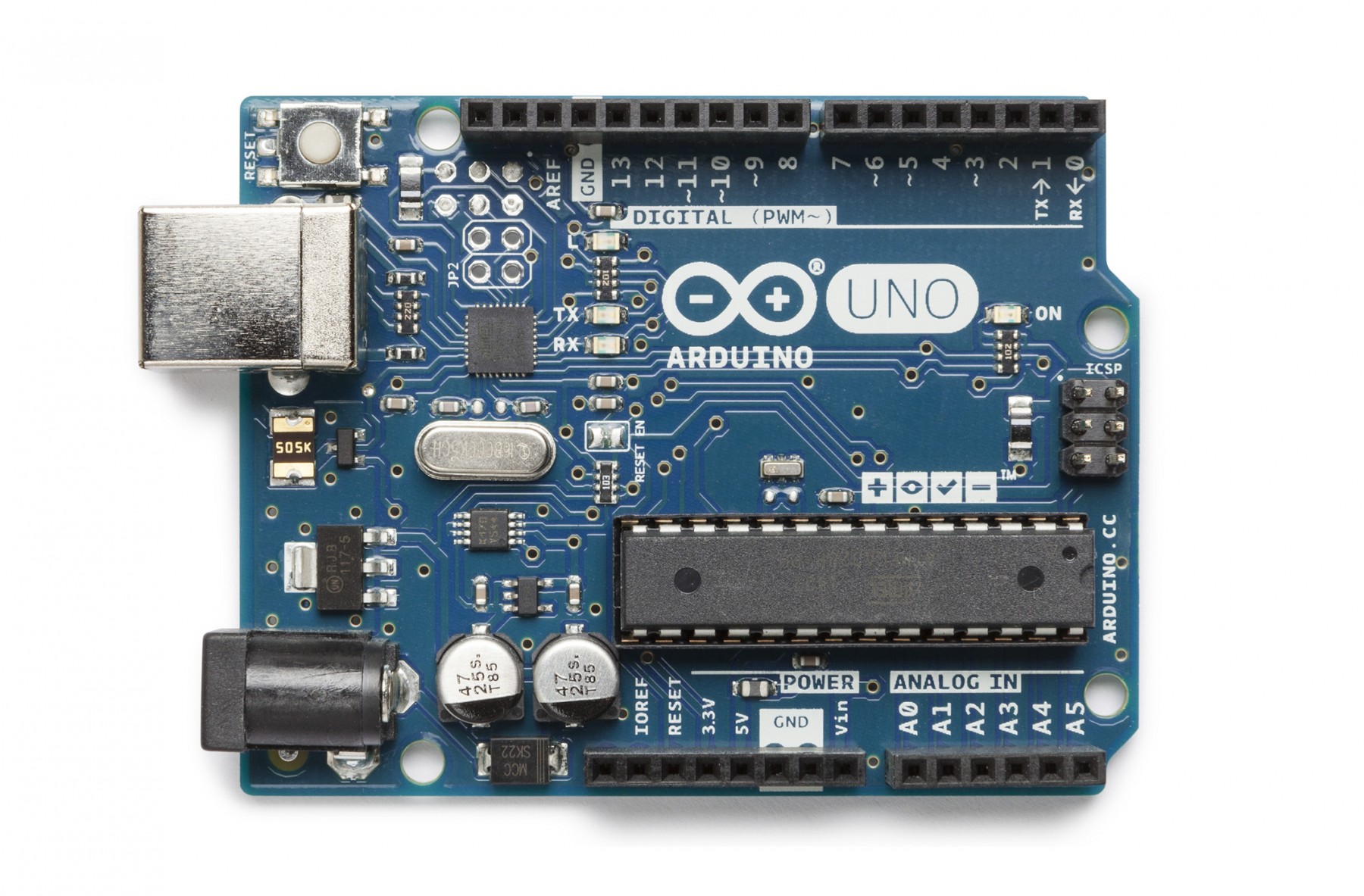
### -Intruder alert:

Whenever someone enters the wrong knock sequence for 4 consecutive times the SOS signal lights up in sequence locking the user from registering any input for a certain period of time while indicating that some intruder is trying to gain access.

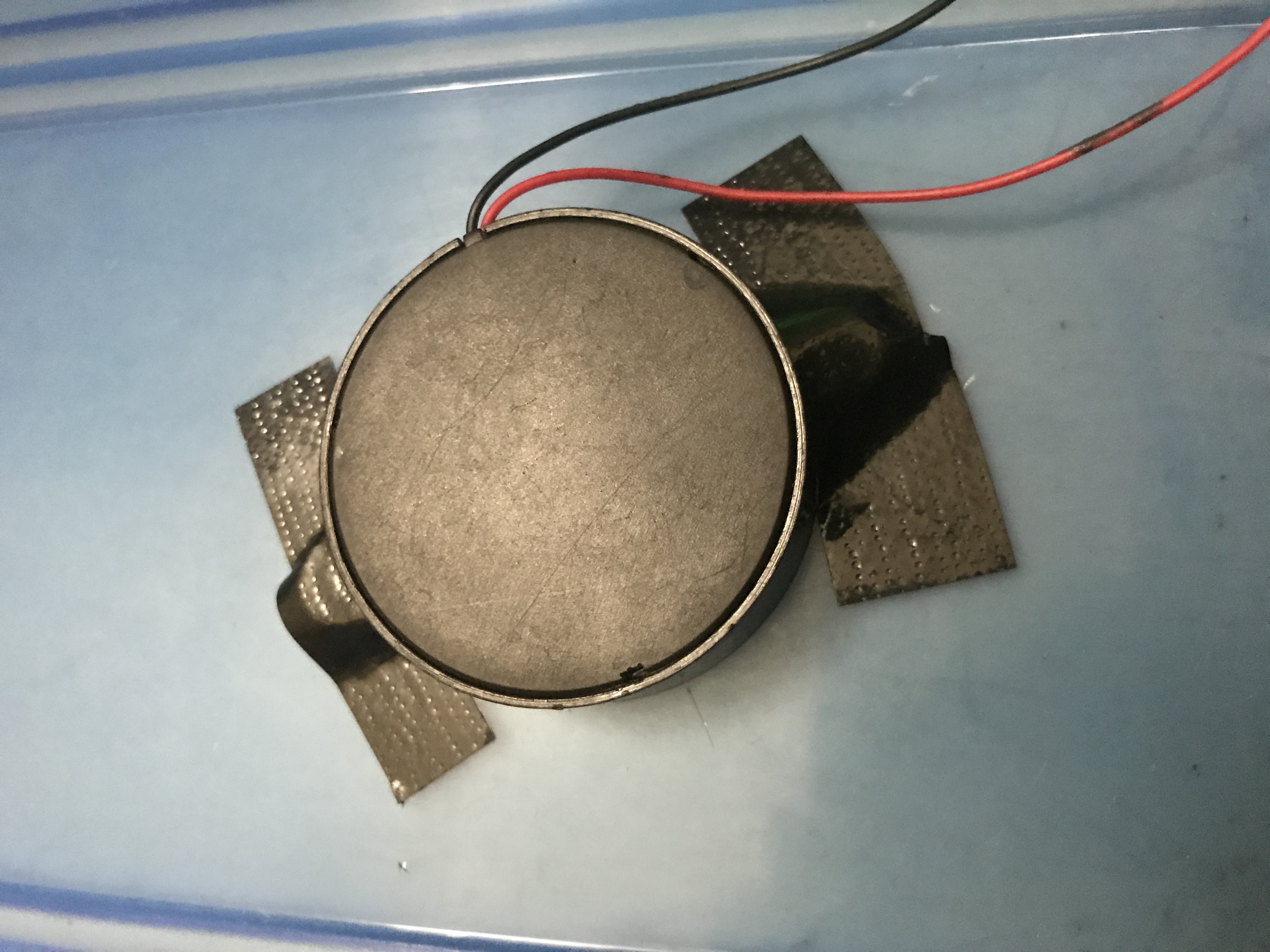
E.g: The LEDs can be set up in the security room to generate intruder alerts.

## Components :

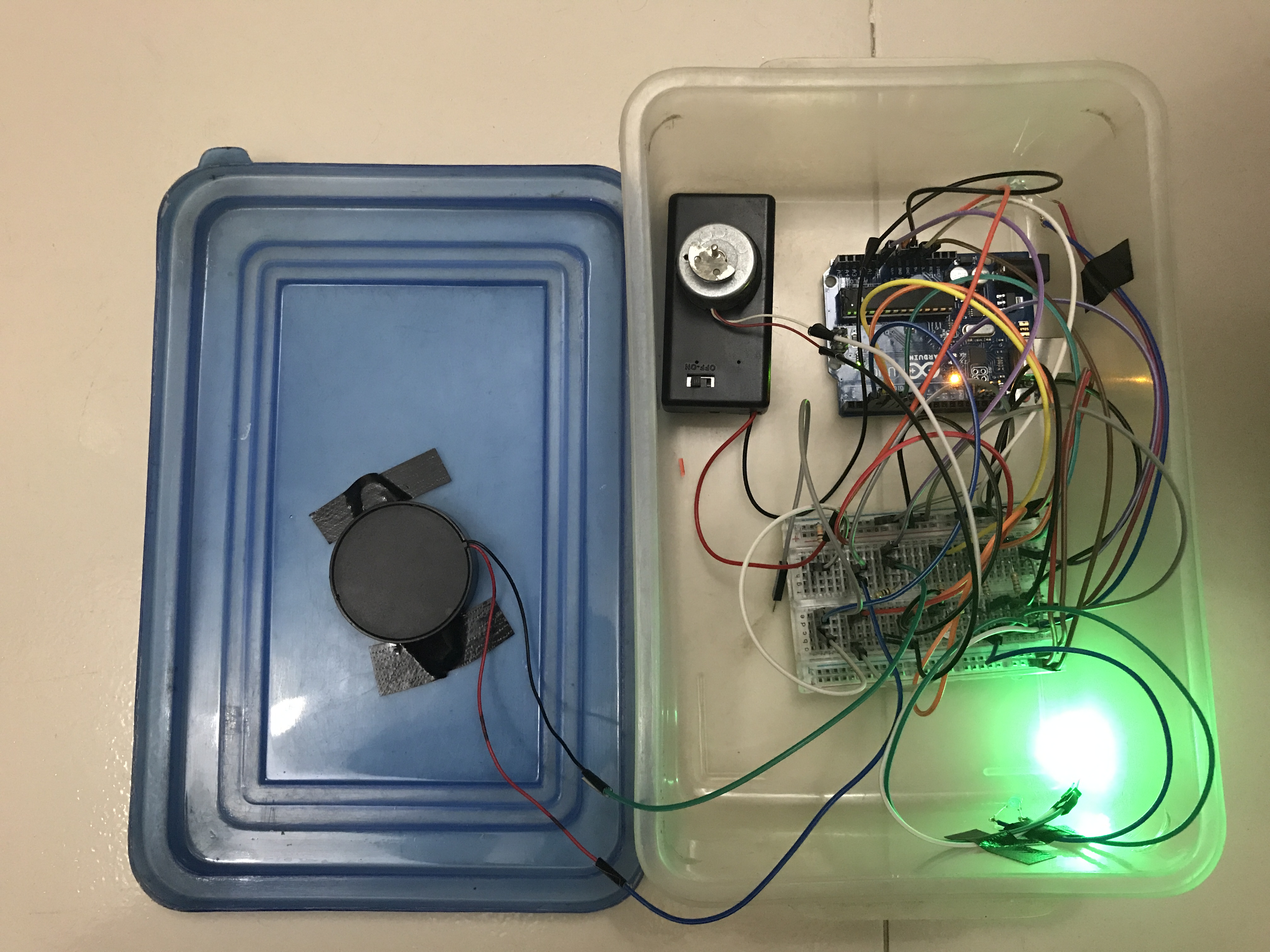
### Arduino UNO:



### Piezo Sensor CF-A1 and Motor:



### Final Project When it is on:



## Code :

/\* Detects patterns of knocks and triggers a motor to unlock

it if the pattern is correct. 4 Consecutive wrong tries will automatically

set the system to give an SOS signal

\*/

// Pin definitions

const int knockSensor = 0; // Piezo sensor on pin 0.

const int programSwitch = 2; // If this is high we program a new code.

const int lockMotor = 3; // Gear motor used to turn the lock.

const int redLED = 4; // Status LED

const int greenLED = 5; // Status LED

const int redLED2 = 6; // Led for SOS alarm

// Tuning constants. Could be made vars and hoooked to potentiometers for soft configuration, etc.

const int threshold = 5; // Minimum signal from the piezo to register as a knock

const int rejectValue = 25; // If an individual knock is off by this percentage of a knock we don't unlock..

const int averageRejectValue = 15; // If the average timing of the knocks is off by this percent we don't unlock.

const int knockFadeTime = 150; // milliseconds we allow a knock to fade before we listen for another one. (Debounce timer.)

const int lockTurnTime = 650; // milliseconds that we run the motor to get it to go a half turn.

const int maximumKnocks = 20; // Maximum number of knocks to listen for.

const int knockComplete = 1200; // Longest time to wait for a knock before we assume that it's finished.

// Variables.

int secretCode[maximumKnocks] = {50, 25, 25, 50, 100, 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // Initial setup: "Shave and a Hair Cut, two bits."

int sosCode[maximumKnocks] = {25, 25, 100, 50, 50, 100, 25, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

int knockReadings[maximumKnocks]; // When someone knocks this array fills with delays between knocks.

int sosReadings[maximumKnocks];

int knockSensorValue = 0; // Last reading of the knock sensor.

int programButtonPressed = false;// Flag so we remember the programming button setting at the end of the cycle.

int decider = 0;

int counter = 0;

void setup() {

pinMode(lockMotor, OUTPUT);

pinMode(redLED, OUTPUT);

pinMode(greenLED, OUTPUT);

pinMode(redLED2, OUTPUT);

pinMode(programSwitch, INPUT);

Serial.begin(9600); // Uncomment the Serial.bla lines for debugging.

Serial.println("Program start."); // but feel free to comment them out after it's working right.

digitalWrite(greenLED, HIGH); // Green LED on, everything is go.

}

void listenAfterGranted(){

Serial.println("knock starting");

int i = 0;

// First lets reset the listening array.

for (i=0;i<maximumKnocks;i++){

knockReadings[i]=0;

}

int currentKnockNumber=0; // Incrementer for the array.

int startTime=millis(); // Reference for when this knock started.

int now=millis();

digitalWrite(greenLED, LOW); // we blink the LED for a bit as a visual indicator of the knock.

if (programButtonPressed==true){

digitalWrite(redLED, LOW); // and the red one too if we're programming a new knock.

}

delay(knockFadeTime); // wait for this peak to fade before we listen to the next one.

digitalWrite(greenLED, HIGH);

if (programButtonPressed==true){

digitalWrite(redLED, HIGH);

}

do {

//listen for the next knock or wait for it to timeout.

knockSensorValue = analogRead(knockSensor);

if (knockSensorValue >=threshold){ //got another knock...

//record the delay time.

Serial.println("knock.");

now=millis();

knockReadings[currentKnockNumber] = now-startTime;

currentKnockNumber ++; //increment the counter

startTime=now;

// and reset our timer for the next knock

digitalWrite(greenLED, LOW);

if (programButtonPressed==true){

digitalWrite(redLED, LOW); // and the red one too if we're programming a new knock.

}

delay(knockFadeTime); // again, a little delay to let the knock decay.

digitalWrite(greenLED, HIGH);

if (programButtonPressed==true){

digitalWrite(redLED, HIGH);

}

}

now=millis();

//did we timeout or run out of knocks?

} while ((now-startTime < knockComplete) && (currentKnockNumber < maximumKnocks));

//we've got our knock recorded, lets see if it's valid

if (programButtonPressed==false){ // only if we're not in progrmaing mode.

if (validateKnock() == true){

triggerDoorUnlock();

} else {

Serial.println("Secret knock failed.");

digitalWrite(greenLED, LOW); // We didn't unlock, so blink the red LED as visual feedback.

for (i=0;i<4;i++){

digitalWrite(redLED, HIGH);

delay(100);

digitalWrite(redLED, LOW);

delay(100);

}

digitalWrite(greenLED, HIGH);

}

} else { // if we're in programming mode we still validate the lock, we just don't do anything with the lock

validateKnock();

// and we blink the green and red alternately to show that program is complete.

Serial.println("New lock stored.");

digitalWrite(redLED, LOW);

digitalWrite(greenLED, HIGH);

for (i=0;i<3;i++){

delay(100);

digitalWrite(redLED, HIGH);

digitalWrite(greenLED, LOW);

delay(100);

digitalWrite(redLED, LOW);

digitalWrite(greenLED, HIGH);

}

}

}

void loop() {

// Listen for any knock at all.

knockSensorValue = analogRead(knockSensor);

if (digitalRead(programSwitch)==HIGH){ // is the program button pressed?

programButtonPressed = true; // Yes, so lets save that state

digitalWrite(redLED, HIGH); // and turn on the red light too so we know we're programming.

digitalWrite(greenLED, LOW); //Tashreque

} else {

programButtonPressed = false;

digitalWrite(redLED, LOW);

//digitalWrite(redLED2, LOW);

}

if(counter == 4){

sosBlink();

counter = 0;

}

if (knockSensorValue >=threshold && decider == 0){

listenToSecretKnock();

}

else if(knockSensorValue >=threshold && decider == 1){

listenAfterGranted();

}

}

boolean validateSOS(){

int i=0;

// simplest check first: Did we get the right number of knocks?

int currentKnockCount = 0;

int secretKnockCount = 0;

int maxKnockInterval = 0; // We use this later to normalize the times.

for (i=0;i<maximumKnocks;i++){

if (sosReadings[i] > 0){

currentKnockCount++;

}

if (sosCode[i] > 0){ //todo: precalculate this.

secretKnockCount++;

}

if (sosReadings[i] > maxKnockInterval){ // collect normalization data while we're looping.

maxKnockInterval = sosReadings[i];

}

}

// If we're recording a new knock, save the info and get out of here.

if (programButtonPressed==true){

for (i=0;i<maximumKnocks;i++){ // normalize the times

secretCode[i]= map(knockReadings[i],0, maxKnockInterval, 0, 100);

}

// And flash the lights in the recorded pattern to let us know it's been programmed.

digitalWrite(greenLED, LOW);

digitalWrite(redLED, LOW);

delay(1000);

digitalWrite(greenLED, HIGH);

digitalWrite(redLED, HIGH);

delay(50);

for (i = 0; i < maximumKnocks ; i++){

digitalWrite(greenLED, LOW);

digitalWrite(redLED, LOW);

// only turn it on if there's a delay

if (secretCode[i] > 0){

delay( map(secretCode[i],0, 100, 0, maxKnockInterval)); // Expand the time back out to what it was. Roughly.

digitalWrite(greenLED, HIGH);

digitalWrite(redLED, HIGH);

}

delay(50);

}

return false; // We don't unlock the door when we are recording a new knock.

}

if (currentKnockCount != secretKnockCount){

return false;

}

int totaltimeDifferences=0;

int timeDiff=0;

for (i=0;i<maximumKnocks;i++){ // Normalize the times

sosReadings[i]= map(sosReadings[i],0, maxKnockInterval, 0, 100);

timeDiff = abs(sosReadings[i]-sosCode[i]);

if (timeDiff > rejectValue){ // Individual value too far out of whack

return false;

}

totaltimeDifferences += timeDiff;

}

// It can also fail if the whole thing is too inaccurate.

if (totaltimeDifferences/secretKnockCount>averageRejectValue){

return false;

}

return true;

}

// Records the timing of knocks.

void listenToSecretKnock(){

Serial.println("knock starting");

int i = 0;

// First lets reset the listening array.

for (i=0;i<maximumKnocks;i++){

knockReadings[i]=0;

sosReadings[i]=0;

}

int currentKnockNumber=0; // Incrementer for the array.

int startTime=millis(); // Reference for when this knock started.

int now=millis();

digitalWrite(greenLED, LOW); // we blink the LED for a bit as a visual indicator of the knock.

if (programButtonPressed==true){

digitalWrite(redLED, LOW); // and the red one too if we're programming a new knock.

}

delay(knockFadeTime); // wait for this peak to fade before we listen to the next one.

digitalWrite(greenLED, HIGH);

if (programButtonPressed==true){

digitalWrite(redLED, HIGH);

}

do {

//listen for the next knock or wait for it to timeout.

knockSensorValue = analogRead(knockSensor);

if (knockSensorValue >=threshold){ //got another knock...

//record the delay time.

Serial.println("knock.");

now=millis();

knockReadings[currentKnockNumber] = now-startTime;

sosReadings[currentKnockNumber] = now-startTime;

currentKnockNumber ++; //increment the counter

startTime=now;

// and reset our timer for the next knock

digitalWrite(greenLED, LOW);

if (programButtonPressed==true){

digitalWrite(redLED, LOW); // and the red one too if we're programming a new knock.

}

delay(knockFadeTime); // again, a little delay to let the knock decay.

digitalWrite(greenLED, HIGH);

if (programButtonPressed==true){

digitalWrite(redLED, HIGH);

}

}

now=millis();

//did we timeout or run out of knocks?

} while ((now-startTime < knockComplete) && (currentKnockNumber < maximumKnocks));

//we've got our knock recorded, lets see if it's valid

if (programButtonPressed==false){ // only if we're not in progrmaing mode.

if (validateKnock() == true){

counter = 0;

triggerDoorUnlock();

}

else if(validateSOS() == true){

Serial.println("SOS mode");

sosBlink();

}

else {

Serial.println("Secret knock failed.");

counter = counter + 1;

digitalWrite(greenLED, LOW); // We didn't unlock, so blink the red LED as visual feedback.

for (i=0;i<4;i++){

digitalWrite(redLED, HIGH);

delay(100);

digitalWrite(redLED, LOW);

delay(100);

}

digitalWrite(greenLED, HIGH);

}

} else {

programButtonPressed = false; // if we're in programming mode we still validate the lock, we just don't do anything with the lock

if(validateKnock() == true){

Serial.println("password for programming mode granted.");

if(digitalRead(programSwitch) == HIGH){

programButtonPressed = true;

}

else{

programButtonPressed = false;

}

decider = 1;

for (i=0; i<4; i++){

digitalWrite(redLED, HIGH);

delay(200);

digitalWrite(greenLED, HIGH);

delay(200);

digitalWrite(greenLED, LOW);

delay(200);

digitalWrite(redLED, LOW);

delay(200);

}

}

else{

decider = 0;

}

//validateKnock();

// and we blink the green and red alternately to show that program is complete.

/\*Serial.println("New lock stored.");

digitalWrite(redLED, LOW);

digitalWrite(greenLED, HIGH);

for (i=0;i<3;i++){

delay(100);

digitalWrite(redLED, HIGH);

digitalWrite(greenLED, LOW);

delay(100);

digitalWrite(redLED, LOW);

digitalWrite(greenLED, HIGH);

}\*/

}

}

void sosBlink(){

for(int i=0; i<5; i++){

digitalWrite(redLED2, HIGH);

delay(200);

digitalWrite(redLED2, LOW);

delay(200);

digitalWrite(redLED2, HIGH);

delay(200);

digitalWrite(redLED2, LOW);

delay(200);

digitalWrite(redLED2, HIGH);

delay(200);

digitalWrite(redLED2, LOW);

delay(600);

digitalWrite(redLED2, HIGH);

delay(400);

digitalWrite(redLED2, LOW);

delay(400);

digitalWrite(redLED2, HIGH);

delay(400);

digitalWrite(redLED2, LOW);

delay(400);

digitalWrite(redLED2, HIGH);

delay(400);

digitalWrite(redLED2, LOW);

delay(600);

digitalWrite(redLED2, HIGH);

delay(200);

digitalWrite(redLED2, LOW);

delay(200);

digitalWrite(redLED2, HIGH);

delay(200);

digitalWrite(redLED2, LOW);

delay(200);

digitalWrite(redLED2, HIGH);

delay(200);

digitalWrite(redLED2, LOW);

delay(900);

}

}

// Runs the motor (or whatever) to unlock the door.

void triggerDoorUnlock(){

Serial.println("Door unlocked!");

int i=0;

// turn the motor on for a bit.

digitalWrite(lockMotor, HIGH);

digitalWrite(greenLED, HIGH); // And the green LED too.

delay (lockTurnTime); // Wait a bit.

digitalWrite(lockMotor, LOW); // Turn the motor off.

// Blink the green LED a few times for more visual feedback.

for (i=0; i < 5; i++){

digitalWrite(greenLED, LOW);

delay(100);

digitalWrite(greenLED, HIGH);

delay(100);

}

}

// Sees if our knock matches the secret.

// returns true if it's a good knock, false if it's not.

// todo: break it into smaller functions for readability.

boolean validateKnock(){

decider = 0;

int i=0;

// simplest check first: Did we get the right number of knocks?

int currentKnockCount = 0;

int secretKnockCount = 0;

int maxKnockInterval = 0; // We use this later to normalize the times.

for (i=0;i<maximumKnocks;i++){

if (knockReadings[i] > 0){

currentKnockCount++;

}

if (secretCode[i] > 0){ //todo: precalculate this.

secretKnockCount++;

}

if (knockReadings[i] > maxKnockInterval){ // collect normalization data while we're looping.

maxKnockInterval = knockReadings[i];

}

}

// If we're recording a new knock, save the info and get out of here.

if (programButtonPressed==true){

for (i=0;i<maximumKnocks;i++){ // normalize the times

secretCode[i]= map(knockReadings[i],0, maxKnockInterval, 0, 100);

}

// And flash the lights in the recorded pattern to let us know it's been programmed.

digitalWrite(greenLED, LOW);

digitalWrite(redLED, LOW);

delay(1000);

digitalWrite(greenLED, HIGH);

digitalWrite(redLED, HIGH);

delay(50);

for (i = 0; i < maximumKnocks ; i++){

digitalWrite(greenLED, LOW);

digitalWrite(redLED, LOW);

// only turn it on if there's a delay

if (secretCode[i] > 0){

delay( map(secretCode[i],0, 100, 0, maxKnockInterval)); // Expand the time back out to what it was. Roughly.

digitalWrite(greenLED, HIGH);

digitalWrite(redLED, HIGH);

}

delay(50);

}

return false; // We don't unlock the door when we are recording a new knock.

}

if (currentKnockCount != secretKnockCount){

return false;

}

/\* Now we compare the relative intervals of our knocks, not the absolute time between them.

(ie: if you do the same pattern slow or fast it should still open the door.)

This makes it less picky, which while making it less secure can also make it

less of a pain to use if you're tempo is a little slow or fast.

\*/

int totaltimeDifferences=0;

int timeDiff=0;

for (i=0;i<maximumKnocks;i++){ // Normalize the times

knockReadings[i]= map(knockReadings[i],0, maxKnockInterval, 0, 100);

timeDiff = abs(knockReadings[i]-secretCode[i]);

if (timeDiff > rejectValue){ // Individual value too far out of whack

return false;

}

totaltimeDifferences += timeDiff;

}

// It can also fail if the whole thing is too inaccurate.

if (totaltimeDifferences/secretKnockCount>averageRejectValue){

return false;

}

return true;

}