Final Project: Knock Detection Entry System

December 11, 2012

Section 1   Introduction

Carrying keys around has always been a hassle for all of us. If one loses a key, he or she has to be assisted by someone else to get in to the room. In addition, there are many situations where you wouldn't want to bring your keys with you either, such as when you're going on a run. We wanted an easier way to enter and exit a locked room, and not ever have to worry about being locked out.

Our project involves an Arduino program which enables pulling a door handle down from the inside of a door once a specific pattern of knocks has been detected. This pattern is configurable via a "record" function, and, when powered, the device is always listening for the pattern, ready to trigger a motor to open the door. If an incorrect knock pattern is given, the program alerts you through a buzz in the speaker. This way, the room is secure, and you'll never be locked out because you don't have the key.

Section 2   Architecture

Section 2.1   Hardware Design

This program assumes a hardware configuration that includes a piezo element for knock sensing (connected on `piezo_pin`, in our code), two LEDs for control output, a button to trigger the recording mode, a servo (`s1`, connected on `servo_pin`) to pull the deadbolt, and a second piezo speaker to buzz when an incorrect pattern is recognized, or the system times out (and starts waiting for another knock series).

The wiring of the system ended up being pretty straightforward (see diagram below). The piezo on the left is for audio output (signaling error events or a reset of the listening system) and the piezo on the right (highlighted in purple) is used for knock detection. The left (red) LED blinks whenever a knock is detected (mainly a debugging tool for us, though also helpful when the user is recording), and the right (green) LED is a control signal: it is turned on during recording, and is used to both signal a timeout event (reset of input knock pattern) and to replay a knock pattern to the user after it has been recorded. When the system receives a correct knock pattern, it will rotate the servo 180 degrees (enough to pull down a door handle or draw back a deadlock), wait a few seconds, and then rotate back to its original position. The rest of our hardware implementation is reasonably simple, but is diagrammed in full below.

Made with ⬦ **Fritzing.org**
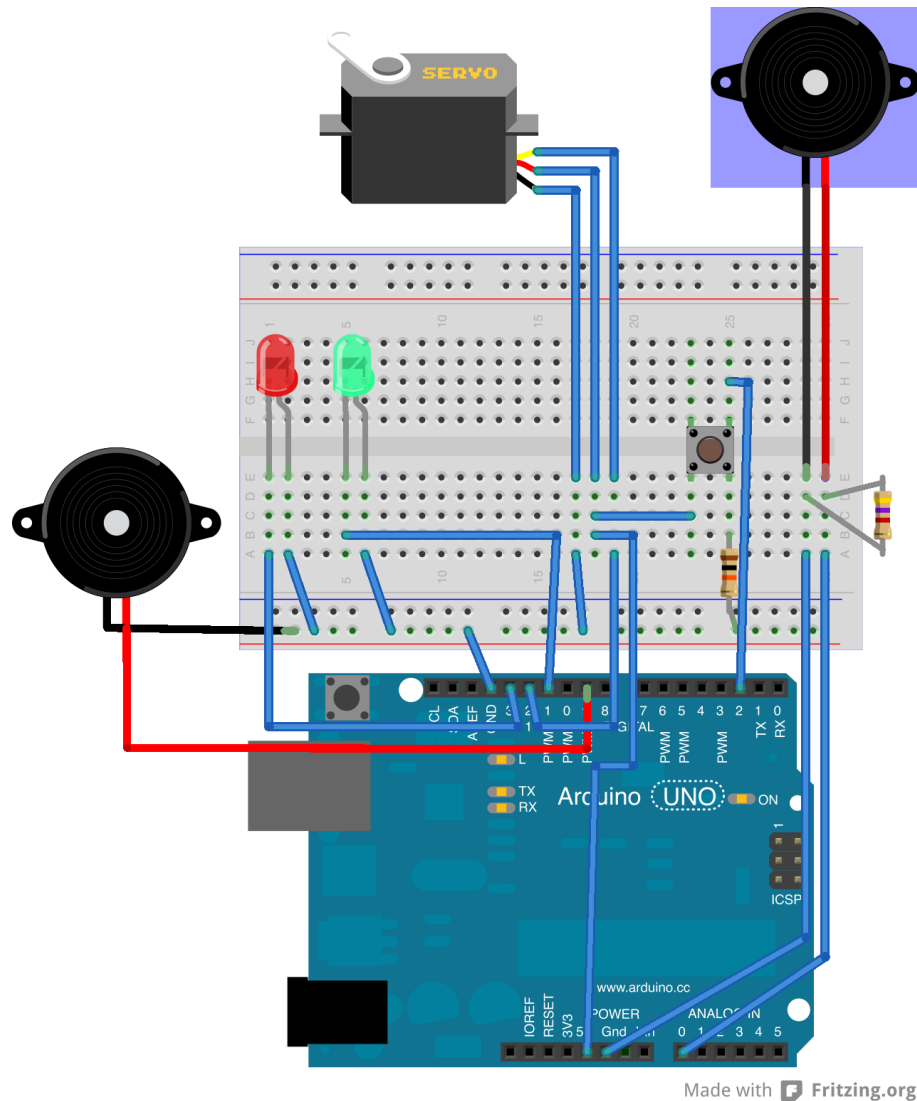
Section 2.2   Software Design

Left to describe is the operation of our software which is running on the Arduino, which is strictly
procedural and adheres to a predetermined flow (i.e. we do not make use of interrupts on the
Arduino platform).

    The basic operation of our software can be summed up well as a finite state machine (below).
The system's default state is iterating within the Arduino main loop, waiting for either a button
press or a knock. At both `Input` and `Record`, it will continue listening for subsequent knocks,
though the transitions out of these states are different: at `Input`, it will only move on to evaluation
of the knock pattern once `n knocks` (the number of knocks in the current valid pattern) have been
detected, while at `Record`, it will wait for a timeout (i.e. knocking has stopped) or until the user
has input the maximum allowable number of knocks to progress to the next state. Along both the
`timeout` and `incorrect` edges, a sound will be played on the speaker. The `Print` state corresponds
to a blocking process during which the current valid pattern is echoed in appropriately spaced blinks

on the control LED, and after this completes, it returns to its normal resting state. The rest of the system's operation within the state diagram should be self-explanatory, though we delve into a few more details (outside of the FSM) below.
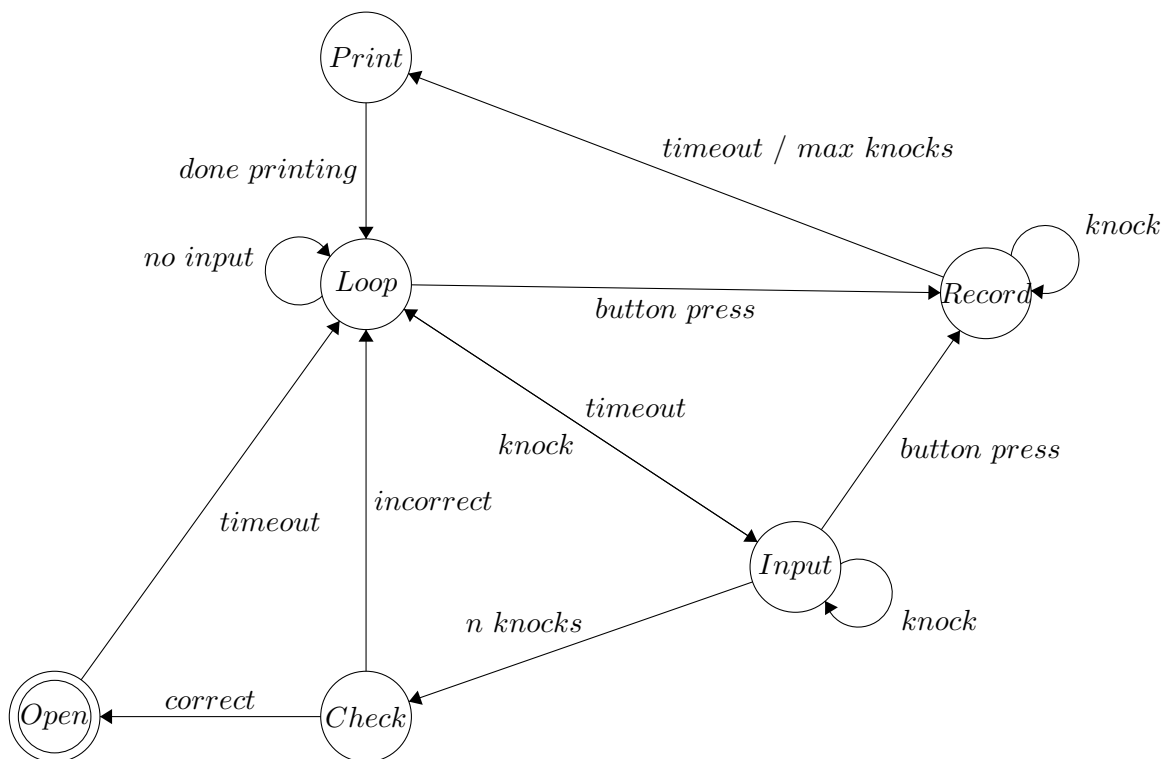
The device begins with a default knock pattern of ten equally spaced knocks. In our system, an n knock pattern is represented as a $n - 1$ sized array, each element being the ratio between adjacent knocks (so the default knock pattern is represented as $[1, 1, 1, 1, 1, 1, 1, 1, 1]$). When the system hears a first knock, it will begin recording the time offsets between subsequent knocks, and after it receives the number of expected knocks, relative ratios are calculated from the recorded offsets. Each of these ratios are then compared to those of the current key pattern (within an error margin), and if they all match then the servo is triggered and the deadbolt is pulled open.

Recording is very similar: pressing the button triggers record mode, and then the current key pattern will be replaced by whatever ratioed pattern is recorded after the button press (and before a timeout, as stated). This pattern will then be played back to the user on a LED (as mentioned above), and the device will re-enter normal operation, listening for entry attempts. If fewer than three knocks are recorded during this process, the valid key pattern will not be replaced, and the previously (and still) valid key pattern will be what is played back on the LED.

Section 2.3   Conceptual

Following is a basic overview for how we imagine an end user approaching the operation of our device. After the above explanation, these things may be evident, but consider this the higher-level, more conceptual, usability-oriented side of our design.

Installation: the device must be installed such that the piezo is pressed directly against the door, to best register vibrations caused by knocking. The servo should be configured to pull the deadbolt or door handle when it rotates.

Recording: press the button once, and then knock out the desired pattern on the inside of the door. Wait for the timeout, and then make sure the replayed pattern (on the second, lower LED) is as you desire. Your patterns must consist of between three and ten knocks (inclusive); shorter knock patterns will not overwrite the current key pattern, and recording will cut off after ten knocks.

Unlocking: simply knock the pre-recorded pattern on the outside of the door, and if the time ratios between your knocks match the stored key pattern, the servo will pull the deadlock or door handle. It must be manually re-locked afterwards. If you make a mistake in the pattern, wait four seconds (the timeout) and try again.

Section 3   Implementation

In this section, we will briefly discuss some specific design choices we made in our implementation.

Timeout: We chose a timeout of 4 seconds. From a user standpoint, we decided it was the right amount of time because 4 seconds between knocks is too long for a pattern, and isn't so long that the user would get impatient waiting for it to timeout.

Threshold: We have a threshold of 0.5. This means that the most any interval between knocks can be wrong by is 0.5. For example, if the interval between 2 knocks is 3, a "correct" input is anywhere from 2.5 to 3.5. After multiple trials we discovered that this constant was the perfect error margin for a standard user.

Knock threshold: This threshold sets how loud a knock has to be in order to be registered. We chose 3 because it detects most knocks, and no static.

Keeping door open: The door stays open for 3 seconds, and nothing can happen with the program for 2 seconds after it closes. The door stays open that long because it's the right amount of time to give the user to enter. We had to block the program for 2 seconds after it closes becasue it would mess up the circuit voltages.

Max number of knocks: We limited the user to a maximum number of 10 knocks for their pattern. We chose this number because it allows for creativity and unique knocking patterns. We limited it to 10 because longer knock patterns would result in the user forgetting the pattern.

Debounce-delay: The sensor doesn't detect a knock as a singular event. It will detect a singular knock as many knocks that happen extremely close together. We need a debounce delay to make the program wait to make sure the program doesnt think the vibrations after the knock are more knocks. We chose .1 seconds because you would never try to do two knocks that quickly, and it fixes the problem stated.

Section 4   Evaluation

We were able to successfully connect the finished product to a door. The piezo sensor used to detect knocks is fairly cheap, so sometimes knocks are not detected correctly, but it works fairly well. To get the best results, the components must be placed somewhere in the center of the door so that all vibrations are picked up clearly.

Recording and playback of patterns works perfectly, and with the addition of a piezo speaker it is easy to tell when you haven't correctly knocked out the pattern, or your knocks have timed out. One problem with the finished product is that the servo we purchased only is able to turn 180 degrees. This means that it is only able to pull a doorknob or lock a small distance. We were able to solve this problem by pre-pulling the knob a small amount, so that the knob only has to be pulled a small amount before the door is opened.

One edge case to deal with is ambient vibrations in the door that the system is connected to. If it picks up stray vibrations, it could either mess up the pattern the user is trying to knock out, or pick up knocks when the user isn't even using the device. Luckily, the first problem is solved by merely re-knocking the pattern, as the buzzer lets you know when you have made a mistake. If knocks are being picked up when the user isn't knocking at all, the buzzer will sound after a timeout, but nothing else will happen.

Section 5   Related Work

There aren't many similar affordable solutions. Doorknob setups with fingerprint scanners exist but are extremely expensive, costing around $300.

"The Clapper" also exists, that gives current to an outlet if it detects claps, and this could potentially be used to implement a similar door-opening system. However, it doesn't let you customize the sequence of claps. It doesn't directly connect to a motor either. Lastly, it detects sound as opposed to vibrations. Our project won't detect random noises, only vibrations on the door whereas "The Clapper" could interpret white noise and randomly open and close the door. "The Clapper" goes for around $20.

Probably the best alternative to our solution is Lockitron: a Kickstarter project that uses smart phones to lock and unlock doors. To use Lockitron, you first need to attach the Lockitron shell around your deadbolt. This shell is smart; it's connected to WiFi and has Bluetooth. From anywhere in the world, you can unlock or lock your door from your smartphone. You can also grant access to other users. Lockitron also automatically unlocks the door when you walk near it with your smartphone. With Lockitron, you would still either need your key or a phone to get into you're locked house. It doesn't entirely address the problem we were having. You can still lock your phone inside the house, and if you don't want to run with keys, why would you run with your phone. We need a solution that doesn't physically require any hardware, and Lockitron doesn't give us that. It also costs around $200. The device can be pre-ordered at: `www.lockitron.com/preorder`

Our project's cost came out to around $50. If we were to sell this project, our costs would decrease due to ordering parts in bulk, so we could sell for around $70 and turn a fairly significant profit.

Section 6   Future Work

If we had had the time, our first stretch goal for this project was to implement remote unlocking via SMS messaging. The basic idea here would be to send a text to a specific number with a control command (eg. "open" or "close"), and if this message originates from a verified number the command would be executed on the device. This could work well as both an override (if we forget the pattern), as well as a good way to let friends into a room which you're not in, et cetera.

We managed to write the backend (i.e. server code) for this aspect of the application, though didn't have time to implement the hardware and Arduino code side of this system. In the `remote_unlock/` directory of our repo and project handin, we have a script which runs as a Twilio endpoint (see `www.twilio.com`), and also has a route to advertise the current (intended) state of the device. Assuming we had our board hooked up to an ethernet connection from which we could access the server, it would be simple to poll against `[server port]/state` and trigger the servo when the "Unlocked" state is observed.

Two further improvements which could be made to our device are making it battery powered (currently it must be connected to a laptop or a wall socket at all times) and having audio as well as visual playback for echoing recorded patterns.