Final Project: Knock Detection Entry System

December 8, 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. A lot of times, there are situations where you wouldn't want to bring your keys with you either, such as when you're going on a run. I personally have forgotten my keys in my room, or have been locked out by my roommate numerous times. We wanted an easier way to enter and exit a locked room, and not ever have to worry about being locked out.

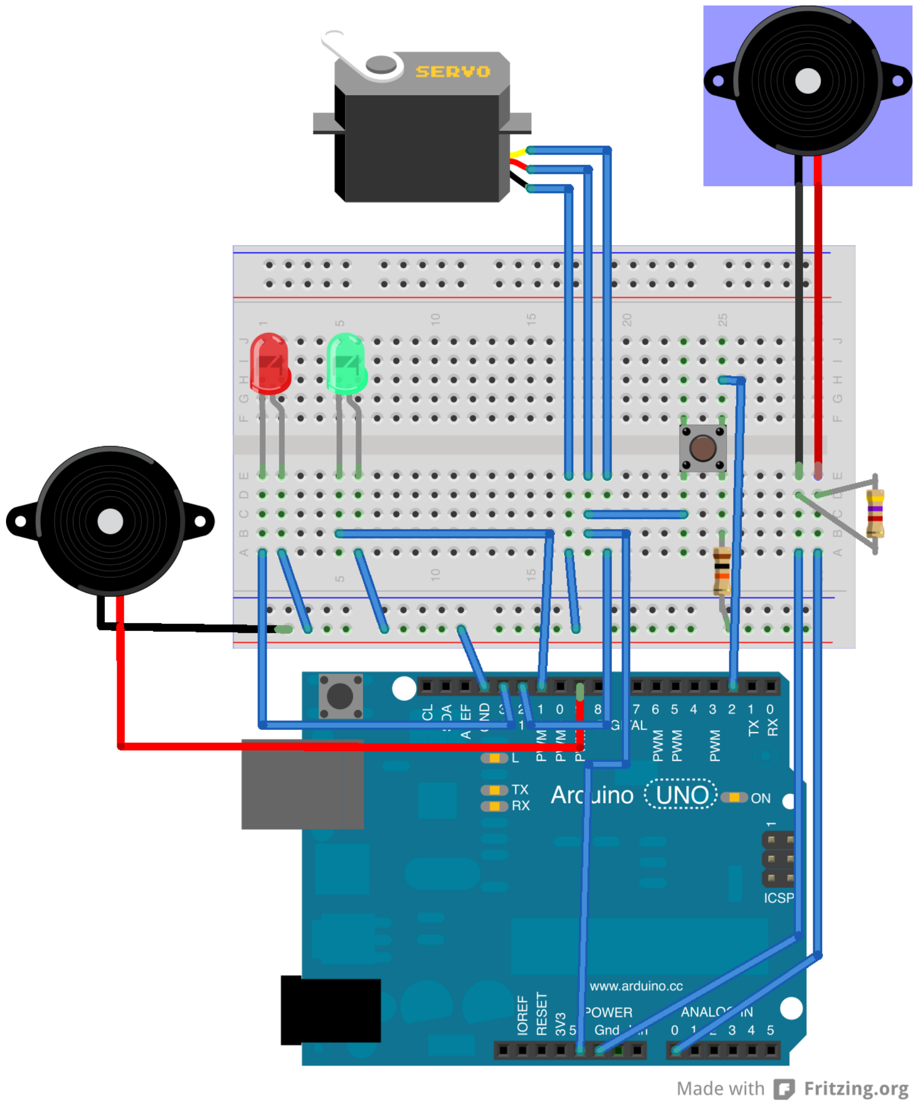
Our project is a reasonably straightforward Arduino program which enables pulling a door handle down from the inside of a door once a specific pattern of knocks has been detected. This pattern is configurable via a "record" function, and, when powered, the device is always listening for the pattern, 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).

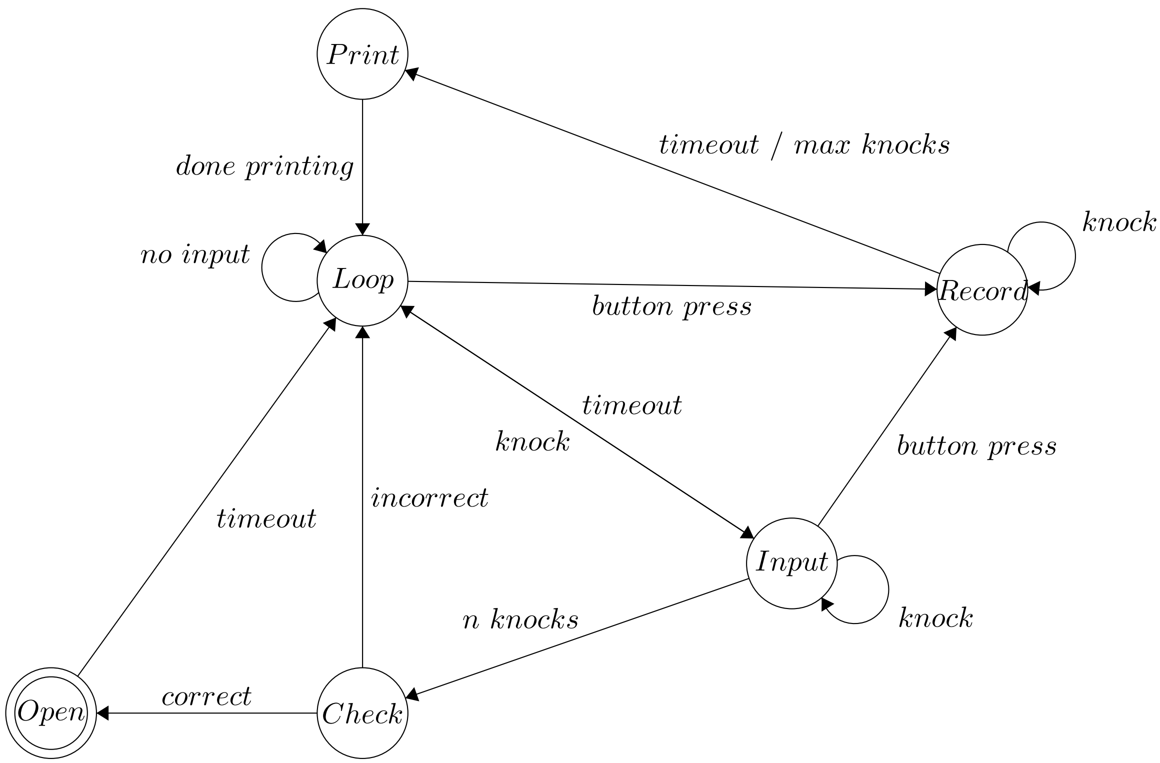
The wiring of the system ended up being pretty straightforward (see diagram below). The piezo on the left is for audio output (signaling error events or a reset of the listening system) and the piezo on the right (highlighted in purple) is used for knock detection. The left (red) LED blinks whenever 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.



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 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

Section 4 Evaluation

Yes it did work.

Section 5 Related Work

There aren't many similar affordabe solutions. Doors with fingerprint scanners exist but are extremely expensive, even though they're more secure. "The Clapper" also exists, that gives current to an outlet if it detects claps. 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 a pose 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.

Section 6 Future Work

In the remote_unlock/ directory of the repo, we have the backend/server script for implementing SMS-triggered unlocking as well (via Twilio). Alas, we haven't yet had any successes in configuring the device's side of this functionality; however, should the device be configured with an ethernet shield, it would be simple to poll against [server port]/state and trigger the servo when the "Unlocked" state is observed.

Battery powered