Voice Coded Lock

Aman Thombre and Logan Greuel
Team 62

May 2023

Abstract

Accessing secure areas often requires some form of physical key, which can be lost or cumbersome to operate when a user's hands are full. To fix this issue, we designed a hands-free, automatic door lock that opens when a user gives the correct audio password. This paper details the design and verification of our door lock. Our lock was able to consistently (with around 85% accuracy) and quickly (with around one second of processing time) classify live audio inputs given to the door lock system, and automatically unlock the door when provided with the correct password.

Contents

1. Introduction

Accessing secure areas currently requires some form of physical token, including access cards, a password on a keypad, or keys. These can be lost, forgotten, or misplaced, leading to users being locked out of their areas of work. Additionally, operating these locks can be cumbersome, especially when the user's hands are full. As a solution to this problem, we created a hands-free, automatic door lock that takes a spoken password as its key. When a user says the word "hamster" to our system, the door unlocks, and if the system hears any other noise, word, or phrase, the door remains locked. Figure 1.1 below shows an image of our design.

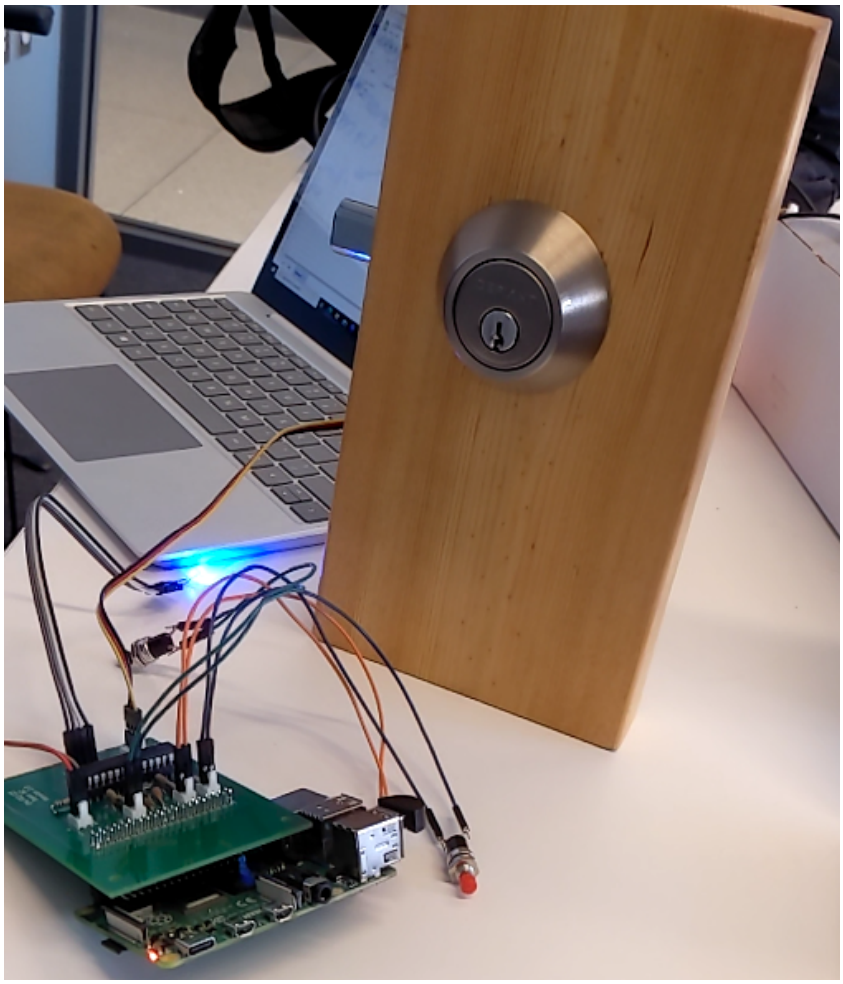

Figure 1.1. Our door lock design.

This design was broken up into five major subsystems: user interface, keyphrase recognition, control unit, mock door, and power. Each of these blocks has a critical role in our design. The user interface module allows the user to speak into our lock and notifies our user of the lock's status. The keyphrase recognition module determines whether audio inputs are the password or not. The control unit facilitates communication between modules and controls

functionality of elements in different modules. The mock door acts as our physical lock, and the power system provides power to our entire design. Figure 1.2 below shows these blocks and their relations.

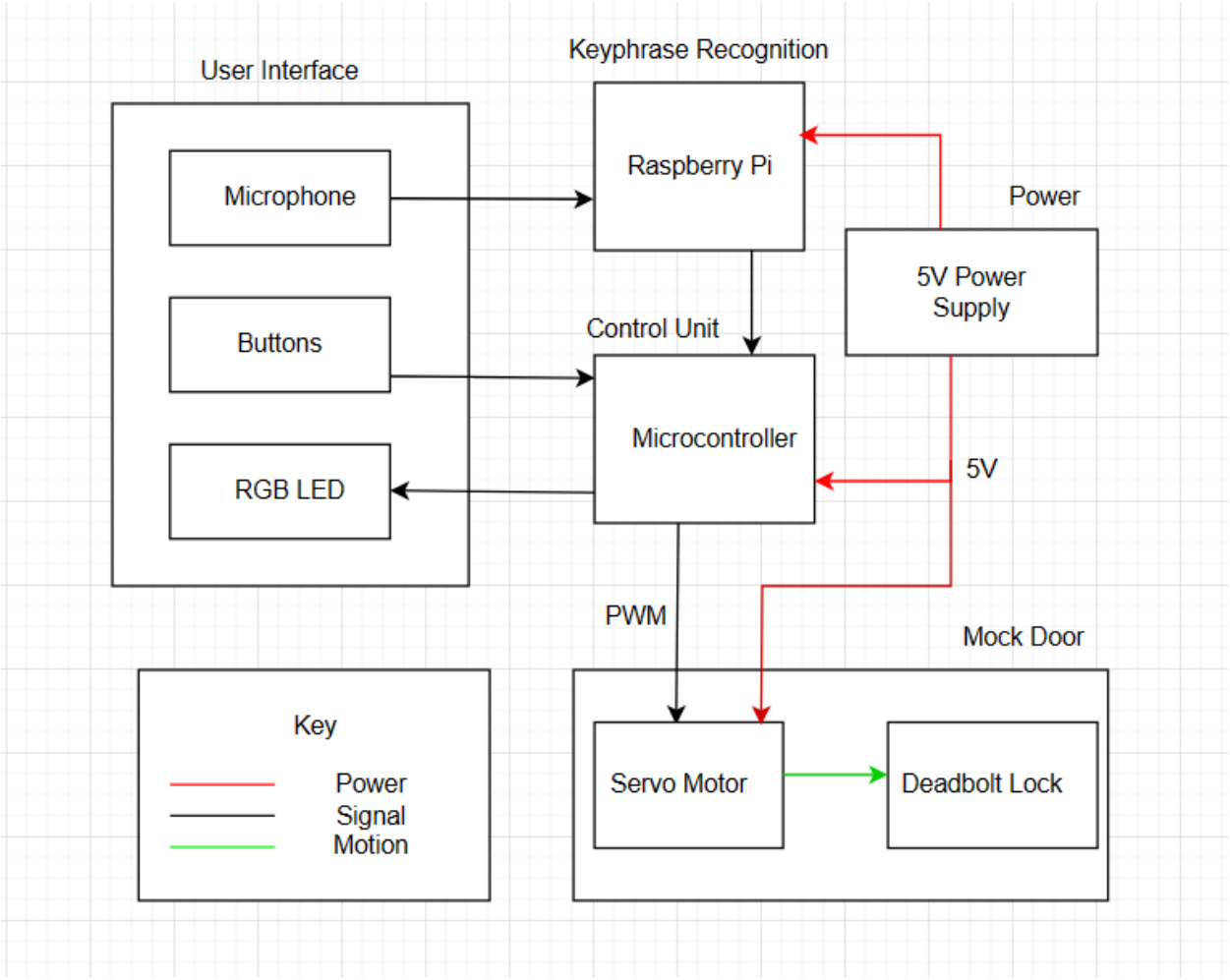

Figure 1.2. High-level block diagram.

In order for our lock to be successful, the keyphrase recognition had to be both accurate and fast. Our lock would also have to operate automatically, without the user needing to do anything apart from saying the correct password to unlock the door. We therefore set the following high-level requirements:

1. Keyphrase recognition should be able to correctly classify audio passwords as correct/incorrect consistently (at least 80±5% accuracy).
2. Operation of the door lock, from saying the password to the door unlocking, should be performed in reasonable time (less than 8 seconds).
3. The system should be able to operate the door lock automatically.

2. Design

2.1 Design Procedure

The design procedure was kept fairly simple. All inputs and outputs rely on simple logic except the audio input from the microphone and the PWM output for the servo motor. The majority of these can be connected directly where they need to go, and the only extra circuitry added consisted of an external oscillator for the microcontroller, debouncing for the buttons, and current-limiting resistors for the LED outputs. General design is shown below and important features will be expanded upon in their respective sections.

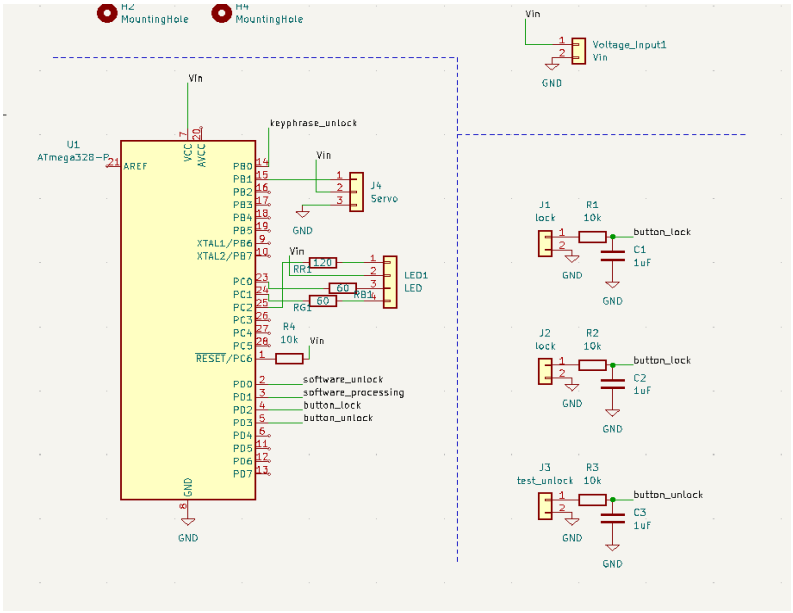

Figure 2.1. Full circuit schematic (excluding Raspberry Pi).

2.1.1 User Interface

The user will interact with our locking system through a microphone and LEDs. Keyphrases will be listened for using a microphone, which will send the audio signal it records in real time to our microcontroller. An RGB LED will be used to signal to the user the state of the locking system - we plan to use different colors to indicate locked, unlocked, and listening. Additional buttons were provided to close the lock or unlock from the inside.

Alternative ways we could include the user interface could run a microphone through a separate ADC to then feed that signal into the Pi. However, our selected microcontroller is not suitable for processing audio signals, and the workload on the Pi is not high enough to justify

running the signal externally since we can use a USB microphone plugged directly into the board and it can very easily process the signal itself.

2.1.2 Keyphrase Recognition

The goal of our keyphrase recognition software is to classify input audio sequences as either the password (in vocabulary, or INV) or not the password (out of vocabulary, or OOV). At a high level, our model consists of three parts: audio processing and transforming input audio into feature vectors by computing the Mel-frequency cepstral coefficients (MFCCs); passing these MFCCs into a hidden Markov model, giving probabilities for being INV or OOV; and passing these probabilities into a support vector machine (SVM) to classify the audio as either INV or OOV.

Figure 2.2. Block diagram of password recognition model.

The use of MFCCs in speech recognition tasks is standard, as they have been shown to give good performance in speech recognition tasks [1]. For classification of the MFCCs, several possible models exist, including deep neural networks [2] and HMMs [3]. While deep neural networks generally yield better results, they require more training data and training time, of which we lacked. Additionally, neural net architecture may be quite complex, which poses a problem for implementation on a Raspberry Pi. Since our model has to classify inputs quickly on a Raspberry Pi as per our high-level requirements, we determined that a HMM model would be best suited for this task. The HMM takes as inputs the MFCCs computed from the previous step, and sends probabilities to an SVM to make final classification decisions. Including an SVM at the end has been shown to boost performance of HMMs built for wake-up-word recognition, a similar problem to ours [4].

2.1.3 Control Unit

The control unit consists of an ATmega328P microcontroller. The microcontroller was used to send signals to/from our user interface and send a PWM signal to the servo on the mock door subsystem. If the audio signal is verified from the keyphrase software, the microcontroller will send a signal to the locking system to disengage the lock. The microcontroller will also read the buttons and send signals to our RGB LED, displaying whether the door is locked, unlocked, or whether a phrase is being processed.

These tasks could be done by almost any number of microcontrollers available, however the ATmega328P was selected because it is a fairly simple controller that is easily programmable through an Arduino Uno board which was already owned.

2.1.4 Mock Door

For our mock door we used a HS-311 Servo which was attached to a standard deadbolt by the machine shop. The key is still accessible by the user in the event of a power failure which is a very convenient fail-safe for the design. The servo motor itself is run by a PWM signal sent from the microcontroller.

Other options for a lock were available. A solenoid lock would be a simple alternative, since it would only require logic to hold it open instead of a PWM signal. This could be done using some sort of switch and the microcontroller using logic to open or close the switch, but solenoid locks typically require higher voltage (for anything beyond an extremely small lock) and a lot more power, where the servo can be simply set to a position and then it waits for the next instructions. A solenoid lock would not require installation through the machine shop, but using the servo itself was a very preferable option to add extra robustness to the design.

2.1.5 Power

Every component in our design requires power, so we required a consistent supply. Since our design was mounted on a door, we used an AC voltage adaptor in a nearby outlet to power our design. All the components are able to be run off of 5V DC and current draw is primarily dominated by the Raspberry Pi, which recommends a 3A supply, but can be run off of less especially since we only have very minimal loading onto the board.

An alternative option for power is to run the design off of batteries. However, both boards constantly require power, and we would most likely need to add a voltage converter to obtain the needed voltage. This would also add concern about the lifetime of the batteries and needing to swap them out. As mentioned above our design will not move however, so it makes much more sense to just use a 5V adapter directly to power the design.

2.2 Design Details

All the blocks above were implemented onto a PCB that provided appropriate ports and circuitry for each component. Headers were included for each chip and all components were through-hole designs.

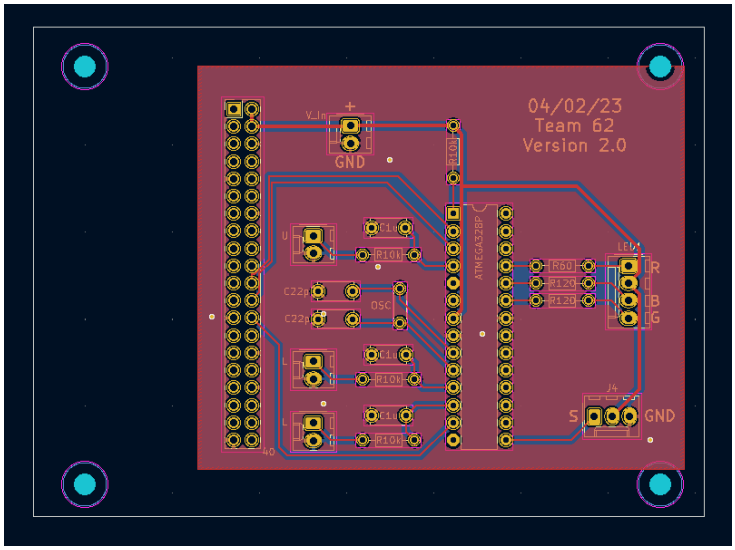

Figure 2.3 above shows final PCB version used in design

2.2.1 User Interface

Buttons

Each button included a low-pass filter debouncer shown in Figure 2.4. The microcontroller takes active low inputs, and hardware debouncing is very simple to implement. We don't want our design to try to shift states too rapidly so this is a simple way to help confirm this.

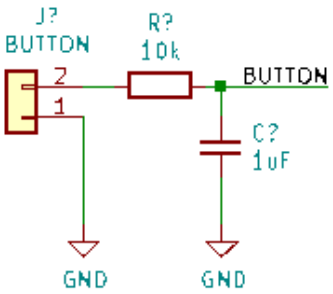

Figure 2.4. Low-pass filter debouncer.

LED

The LED also required current-limiting resistors since the voltage of the LED itself was not the same as the output voltage of the microcontroller. This had to be limited to 20 mA for each pin, and the voltage on the LED pins consisted of 2.2 V or 3.3 V depending on the pin [5], while the microcontroller would output approximately 4.5 V at 20 mA output. Using Ohm's law we can determine the resistance needed to be 115 Ω and 60 Ω respectively. However to

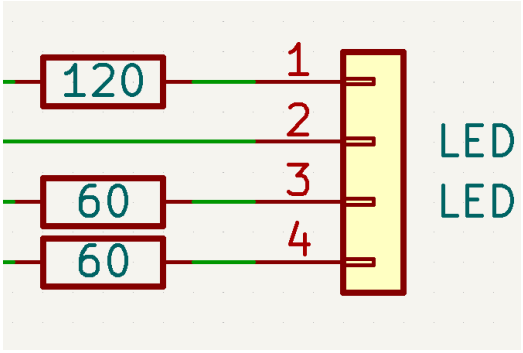

Figure 2.5. Current-limiting resistors.

be safe in the actual design, higher values were used (220 Ω and 100 Ω) to simplify the components needed and assure the actual value doesn't exceed the rated value.

Microphone

The microphone required no design on our part; we used a USB plug and play microphone, which was inserted directly into the Raspberry Pi.

2.2.2 Keyphrase Recognition

Audio Preprocessing and MFCCs

As described in section 2.1.2, the first block of our keyphrase recognition module was audio preprocessing and computing feature vectors via MFCCs. Audio processing was performed prior to computing the MFCCs in order to only record when inputs are being given to the microphone, reduce noise, and remove dead space in audio inputs. The entire process is as follows:

1. Start recording for three seconds when the audio input at the microphone reaches a volume threshold.
2. Apply a Savitzky-Golay filter to boost signal to noise ratio.
3. Remove dead space from the end of the signal.
4. Compute MFCCs.

Removing dead space from the end of the signal was done to reduce the number of MFCC vectors sent to the HMM, and was done by only keeping parts of the signal with high variance. As is standard, 15 MFCCs were computed, and the first two were discarded - this is standard practice as the first two MFCCs generally do not contain differentiating information [1]. The Savitzky-Golay filter was implemented using the SciPy library [6], and the MFCCs were computed using the librosa library [7]. In figure 2.6 below, each column of the MFCC plot is one "frame" (the number of frames depends on the length of the signal), with each frame being a length-13 vector of MFCCs.

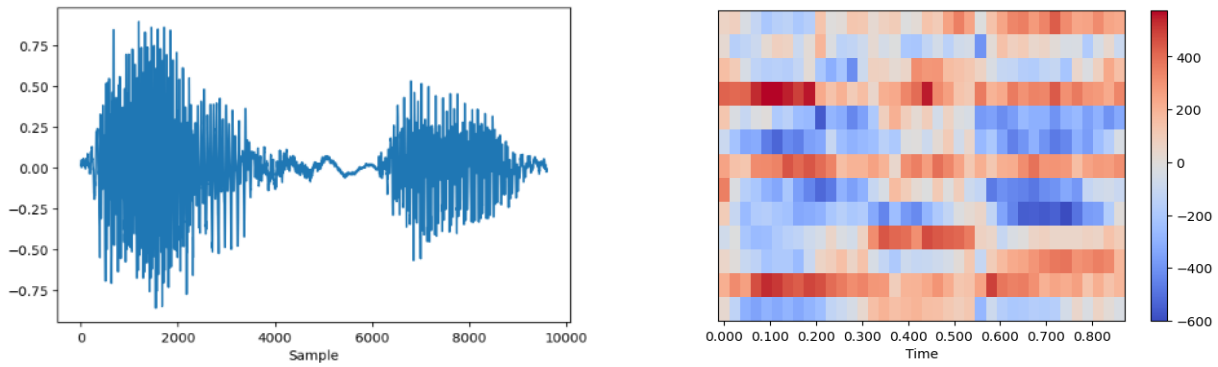

Figure 2.6. Extracted and filtered audio signal of the word “hamster” (left) and MFCCs (right).

Hidden Markov Model

HMMs allow us to calculate the probability that a given model generates a certain observation sequence. We can leverage this by developing a model for INV words and a model for OOV words, and comparing probabilities that a sequence of MFCCs (the observation sequence) is generated by each model. Developing this model includes finding the transition probabilities (A), emission probabilities (B), and initial state probabilities (π) for both the INV and OOV models. These probabilities can be written as:

$$A_{ij} = P(q_t = j | q_{t-1} = i).$$
$$B_i(x_t) = P(x_t | q_t = i).$$
$$\pi_i = P(q_1 = i).$$

We then use the forward algorithm [8] to calculate the probability of an observation sequence (X) given a model (Λ): $P(X|\Lambda)$. The forward algorithm is shown below:

1. Define probability α in state i at time t as the probability of our observation sequence so far $(x_1, \ldots, x_t)$ and that we are in state i $(q_t = i)$ given Λ.
$$\alpha_i(t) = P(x_1, \ldots, x_t, q_t = i | \Lambda). \quad (2.1)$$

2. Initialize α at each state at time $t = 1$:
$$\alpha_i(1) = P(x_1, q_1 = i | \Lambda) = \pi_i * B_i(x_1).$$

3. Iterate calculating α at each time step and each state:
$$\alpha_j(t) = \sum_i \alpha_i(t-1) * A_{ij} * B_j(x_t).$$

4. Calculate $P(X|\Lambda)$ through marginalization of eq 2.1:

$$P(X|\Lambda) = \sum_i \alpha_i(T).$$

Our problem then becomes finding A, B, and π for each model given examples of INV and OOV words. One key aspect of this is modeling B as a multivariate Gaussian, as finding example emissions from the training data will not represent all possible emissions, and will not represent the overall distribution well. B then is modeled by mean vector μ and covariance matrix Σ. Initial state probabilities are trivial, as we define our states in time; the initial state is then always state 1, so π is a one-hot vector, with 1 as its first element. With this in mind, we can estimate A, μ, and Σ using the Baum-Welch algorithm, a maximum likelihood estimation algorithm for HMMs [9]. We can write the Baum-Welch algorithm as follows:

1. Make initial guesses for A, μ, Σ by splitting training samples into N states, calculate A by counting transitions, μ, Σ by finding mean and covariance of each state's MFCC vectors.

2. Define probability γ in state i at time t as the probability of our being in state i ($q_t = i$) given the observation sequence and model:
$$\gamma_i(t) = P(q_t = i|X, \Lambda). \tag{2.2}$$
Using Bayes's rule, we can write:
$$\gamma_i(t) = \frac{P(X, q_t=i|\Lambda)}{\sum_k P(X, q_t=k|\Lambda)}. \tag{2.3}$$
Note that we know half of this using α, eq (2.1). Calculate the other half by defining
$$\beta_i(t) = P(x_t + 1, \ldots, x_T, q_t = i|\Lambda). \tag{2.4}$$
Calculate β similarly to α, except iterating backwards in time:
$$\beta_i(t) = \sum_j A_{ij} * B_j(x_{t+1}) * \beta_j(t+1)$$
Then, combine eqs 2.1, 2.3, and 2.4 to calculate γ:
$$\gamma_i = \frac{\alpha_i(t) * \beta_i(t)}{\sum_k \alpha_k(t) * \beta_k(t)}.$$

3. Define probability ξ between states i, j at time t as the probability of transitioning from state i to state j given X, Λ.
$$\xi_t(i, j) = P(q_t = i, q_{t+1} = j|X, \Lambda). \tag{2.5}$$
Calculate ξ using definition of α (eq 2.1):
$$\xi_t(i, j) = \frac{\alpha_i(t) * A_{i,j} * B_j(t+1}{\sum_{k,l} \alpha_k(t) * A_{kl} * B_l(t+1)}.$$

4. Using our definitions of γ, ξ (eqs 2.2 and 2.5), calculate new A, μ, Σ:

$$A_{ij} = \frac{\sum_k \xi_t(i,k)}{\sum_k \sum_t \xi_t(i,k)}.$$

$$\mu_i = \frac{\sum_t \gamma_i(t) x_t}{\sum_t \gamma_i(t)}.$$

$$\Sigma_i = \frac{\sum_t \gamma_i(t)(x_t - \mu_i)(x_t - \mu_i)^T}{\sum_t \gamma_i(t)}.$$

5. Repeat steps 2-4 until convergence.

To summarize these two algorithms, we use Baum-Welch to learn parameters of our models given many training examples, and we then use the forward algorithm to find the probability of each model generating a given audio sequence. Figure 2.7 below shows plots of the probabilities of being generated by each model for INV and OOV words in our training data. Figure 2.8 below shows a plot of the probability ratio vs probability of being generated by INV.

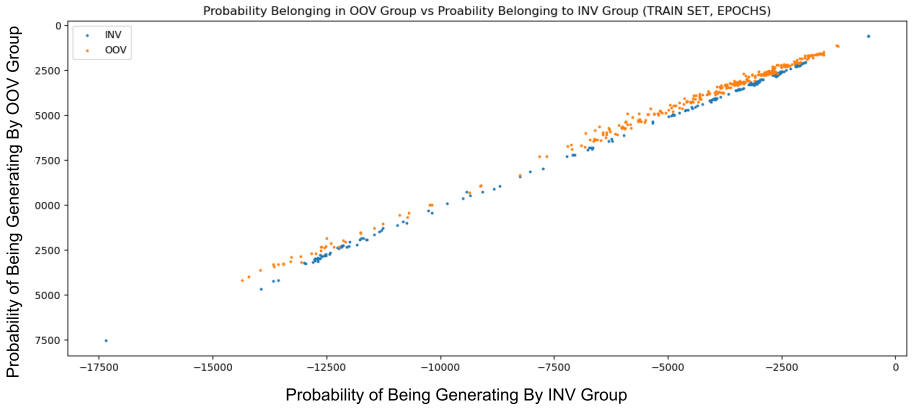

Figure 2.7. Probabilities of being generated by each model for INV and OOV samples.

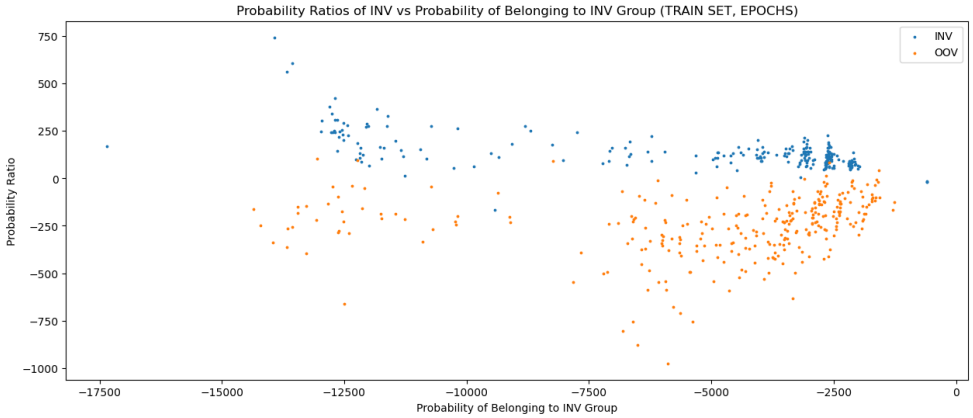

Figure 2.8. Probability ratio vs probability of being generated by INV group.

Support Vector Machine

As can be seen in figures 2.7 and 2.8, the HMM outputs fairly separable data, indicating a good use case for an SVM classifier. We fit an SVM on the training data using the sci-kit learn library [10], passing in the probability of being generated by the INV model, the probability of being generated by the OOV model, and the ratio of these two probabilities. We selected a polynomial kernel with degree = 3. Figure 2.9 below is an example decision boundary trained on the probability ratio vs probability of being generated by INV. As can be seen, the SVM has fairly good performance in placing INV words and OOV words on the correct sides of the decision boundary. Note that this is not a representation of our actual decision boundary, which is in three dimensions.

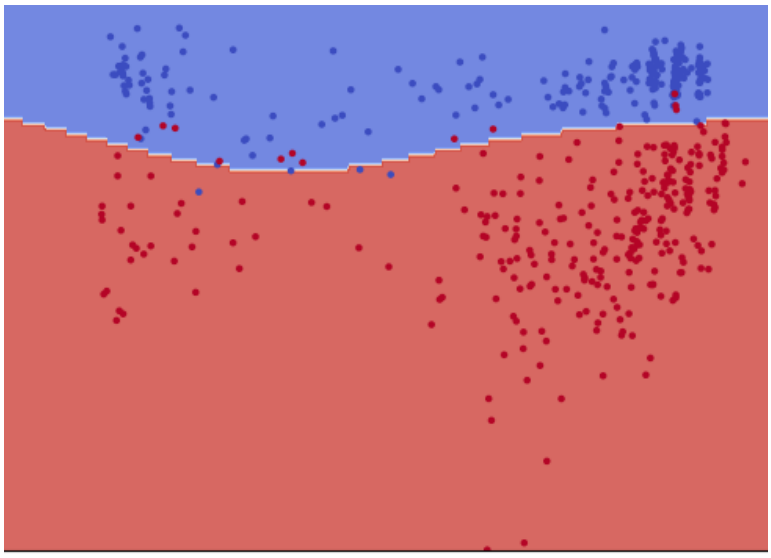

Figure 2.9. Example SVM decision boundary, using the same data as figure 2.8.

Datasets

In order to train, tune, and test our model, we created train, validation, and test datasets, respectively. In order to create the INV corpus, one teammate recorded themselves saying the word “hamster” 350 times. Of these 350 samples, 250 went into the train dataset, 50 into validation, and 50 into test. In order to create the OOV corpus, a combination of custom recordings and online datasets were used. One teammate recorded themselves saying 300 random words, of which 200 went into the train dataset, 50 into validation, and 50 into test. Additionally, 900 0.3-1.2 second long “chunks” were extracted from the LibriSpeech dataset [11], with 800 going into train, 50 into validation, and 50 into test. Tables 2.1 and 2.2 below summarize this.

	Train	Validation	Test	Total
“Hamster” recordings	250	50	50	350
Total	250	50	50	350

Table 2.1. INV datasets breakdown.

	Train	Validation	Test	Total
Random word recordings	200	50	50	300
LibriSpeech chunks	800	50	50	900
Total	1000	100	100	1200

Table 2.2. OOV datasets breakdown.

2.2.3 Control Unit

Ports on the ATmega were grouped according to their respective function (input or output) and then accessibility in the PCB design. The following table lists each pin used and its function along with a figure showing the pinout in relation to the chip [12] (Software: SW, Button: BT).

Table 2.3. Pin assignments for ATmega328P

Pin Selected	Input/Output	Description
B1	O	PWM to servo
C0	O	BLUE LED
C1	O	GREEN LED
C2	O	RED LED
D0	I	PROCESSING SW
D1	I	UNLOCK SW
D2	I	UNLOCK BT

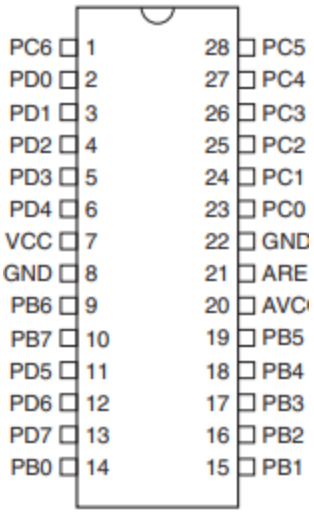

Figure 2.10. ATmega328P Pinout.

D5	I	LOCK BT1
D6	I	LOCK BT2
D7	I	LOCK SW

2.2.4 Mock Door

The HS-311 servo uses 50 Hz pulse width signals to determine its position according to Figure 2.9 [13]. With the servo mounted on the door the lock and unlock positions for our design were found to be at 1.4 ms and 2.0 ms respectively.

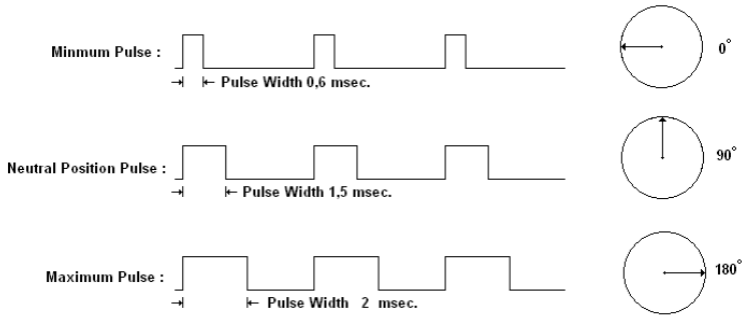

Figure 2.11. Servo Pulse Widths.

The ATmega328P outputs the pulse width signals by using counters to output high until a given value, then low until the reset value is reached. Equation 2.1 is used by the controller to determine the number of counts required for this operation. Frequency of the clock will be 16 MHz, TOP will be the number we use in the code to specify count number, and then N is a prescaler divider which changes how many pulses before the counter counts. This affects signal resolution and how much work the controller has to do (N is 1,8,64, or 256, set as 8).

$$f_{OCnxPWM} = \frac{f_{clk\ I/O}}{N \cdot (1 + TOP)} \tag{2.6}$$

Values were found by modifying equation (2.6) [11], using a PWM frequency of 50 Hz to find the reset value, then solving the equation for times according to 1.4 and 2.0 ms to find the times for those as well. Then the servo is set to one of those values to set the position.

2.2.5 Power

Since our power supply was a purchased adapter, we didn't have to add any additional circuitry or adjust anything on it. We used a barrel jack adapter to plug into the supply and convert that to usable Vcc and GND pins for our circuit.

3. Verification

Table 3.1 below shows our requirements and verification (RV) table.

Requirements	Verification
● The LED will display three different colors based on status: ○ Red, meaning door locked ○ Green, meaning door open ○ Blue, meaning processing	● Initialize the lock system with no one talking and no ambient sound, ensure that the LED is red. ● Then, speak an incorrect password to the system. The LED should turn blue for processing time, and then turn back to red. ● Then, speak the correct password to the system. The LED should turn blue for processing, and then green as the door unlocks.
● The keyphrase recognition software should be able to handle audio inputs up to 3 seconds long.	● Audio inputs up to 3 seconds long will be provided to the keyphrase recognition software. ● Ensure that the software can accurately detect the correct password if it is uttered in the 3 second audio input, passing accuracy requirements detailed below.
● Keyphrase recognition should not take more than 5 ± 1 seconds.	● After inputs are fed to the keyphrase recognition system, the Python time package will be used to time how long keyphrase recognition takes.
● Should be able to distinguish correct/incorrect keyphrases with at least 80% ± 5% accuracy. ● Precision, recall, and specificity should also pass this same benchmark.	● Each team member will provide 20 attempts to unlock the door (10 with the correct password and 10 with the incorrect password for 40 attempts total). ● Results will be noted in a confusion matrix, which will be used to calculate accuracy, precision, recall, and specificity. ○ Accuracy = TP+TN/Total ○ Precision = TP/TP+FP ○ Recall = TP/TP+FN ○ Specificity = TN/TN+FP

• The servo motor must have enough torque to operate the deadbolt lock.	• Assemble servo onto the lock and test varying PWM signals to check that lock will turn
• Send PWM signals to the Servo to control locking and unlocking controls	• Connect servo and confirm output from microcontroller will rotate into desired positions

Table 3.1. RV Table.

Verifications for the LED, servo motor, locking system, and timing for speech recognition were verified at our demo, and can also be verified in our video: https://youtu.be/tJNp-pIufbc.

Our verification for the speech recognition module is shown below, in table 3.2.

		Ground Truth (spoken word)	
		INV (“Hamster”)	OOV (Random)
Classified as (Door action)	INV (Door unlocks)	18	3
	OOV (Door locks)	2	17

Table 3.2. Confusion matrix generated in verification of speech recognition.

From table 3.2, we calculated metrics of classifier performance, including precision, recall, specificity, accuracy, and F1 score. These are summarized below, along with their meanings:

- Precision = 0.857.
  - If the door unlocks, the probability that the user said “hamster” is 85.7%.
- Recall = 0.9.
  - If the user says the word “hamster,” the door unlocks 90% of the time.
- Specificity = 0.85.
  - If the user says a random word, the probability that the door stays locked is 85%.
- Accuracy = 0.875.
  - Probability that the lock correctly classifies the provided password.
- F1 Score = 0.878.
  - Blend of precision and recall, standard measure of classifier accuracy.

As can be seen above, our design passes all of the requirements listed in table 3.1, and passes our high-level requirements described in section 1.

4. Cost

Hourly wages will be $40/hr, found using average expected salary for ECE graduates at UIUC [14] and converted to an approximate hourly rate. There are 2 in our group and we averaged our hours spent on the project. We also had the machine shop build our mock door for us so we can value their time the same way and add 6 hours for that as well.

$$Total\ Labor\ Cost\ = 2 \cdot \frac{\$40}{hr} \cdot 2.5 \cdot 100\ hr\ + \frac{\$40}{hr} \cdot 2.5 \cdot 6\ hr\ = \$20,600$$

Table 4.1. Cost Analysis

Part Description	**Provider**	**Part Number**	**Quantity**	**Cost**
USB PnP Microphone	ShopSimple		1	$ 6.99
ATmega328 Microcontroller Bootloader Uno	ECE Supply	X000048	1	$7.20
SERVO MOTOR HS-311	ECE Supply	HS-311	1	$11.01
Single Cylinder Stainless Steel Deadbolt	Home Depot	NA	1	$12.47
Raspberry Pi 4 B	Adafruit	2885	1	$35.00
AC/DC Wall Mount Adapter 5V 3A	BestBuy	NA	1	$17.95
LED RGB Clear T-1 3/4 T/H	Digi-Key (Kingbright)	754-2153-ND	1	$2.02
Momentary Button - Panel Mount	Sparkfun	COM-11992	3	$3.15
Misc. Electronics for circuitry	ECE Supply	NA	NA	$10.00
Total Parts Cost				$103.80
Labor				$20,600
Total Cost				$20,705.79

5. Conclusions

Overall, our project proved to be successful. All hardware components operate as intended for user input/output and operating the lock. Meanwhile, our software exceeded all requirements placed upon it during planning. We met the accuracy expectation for the keyphrase recognition and the system functions well within the allotted time frame.

Our project has no major ethical or safety concerns. The only ethical concern that could be relevant is privacy concerns due to recording audio for the lock. If this were being used to record conversations this would be a violation of people's privacy which would break the IEEE Code of Ethics section I part 1 [15] and also the ACM code part 1.6- Respect Privacy [16]. However, our design will only record short inputs (3 seconds) which prevents it from recording entire conversations and will contain any audio recordings internally. Audio recordings will not be shared with any other device and will not be stored, so we do not pose the risk of violating user's privacy or sharing sensitive information.

Our project also does not have any broad impact in any type of large context, but it could provide some smaller impact on schools if the design were implemented. Since the design is meant for fairly specific uses, such as lab access, even if the design were implemented it would be very unlikely that there would be any huge impact from it. It would provide easier alternatives for places where it's expected that many people would need entry, but would in no way revolutionize entry into secure facilities.

6. References

[1] Nair, Pratheeksha. “The Dummy's Guide to MFCC.” *Medium*, Prathena, 27 July 2018, https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd.

[2] Chen, Guoguo, et al. “Small-Footprint Keyword Spotting Using Deep Neural Networks.” *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, https://doi.org/10.1109/icassp.2014.6854370.

[3] Rose, R.C., and D.B. Paul. “A Hidden Markov Model Based Keyword Recognition System.” *International Conference on Acoustics, Speech, and Signal Processing*, 1990, https://doi.org/10.1109/icassp.1990.115555.

[4] Këpuska, V.Z., and T.B. Klein. “A Novel Wake-up-Word Speech Recognition System, Wake-up-Word Recognition Task, Technology and Evaluation.” *Nonlinear Analysis: Theory, Methods & Applications*, vol. 71, no. 12, 2009, https://doi.org/10.1016/j.na.2009.06.089.

[5] “T-1 3/4 (5mm) Full Color LED Lamp.” *Kingbright USA,* Kingbright, Released Nov. 15, 2019. https://www.kingbrightusa.com/images/catalog/SPEC/WP154A4SEJ3VBDZGC-CA.pdf.

[6] “Scipy.signal.savgol_filter.” Scipy, SciPy, https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol_filter.html.

[7] “Librosa.feature.mfcc.” *Librosa 0.10.1dev Documentation*, https://librosa.org/doc/main/generated/librosa.feature.mfcc.html.

[8] Hasegawa-Johnson, Mark. “ECE 417 Lecture 14: Hidden Markov Models.” 2021.

[9] Hasegawa-Johnson, Mark. “ECE 417 Lecture 15: Baum-Welch.” 2021.

[10] “Sklearn.svm.SVC.” *Scikit*, Scikit-Learn, https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[11] Panayotov, Vassil, et al. “Librispeech: An ASR Corpus Based on Public Domain Audio Books.” *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, https://doi.org/10.1109/icassp.2015.7178964.

[12] *ATmega48A/PA/88A/PA/168A/PA/328/P, megaAVR Data Sheet*, Microchip, May 2019.

[13] *SERVO MOTOR CONTROL*. AjlonTech. http://www.ajlontech.com/7.SERVO%20MOTOR%20CONTROL.pdf#:~:text=Hitec%20HS-311%20Standard%20Detailed%20SpecificationsControl%20System%3A%20%2BPulse%20Width,Operating%20Temperature%20Range%3A%20-20%20to%20%2B60%20Degree%20C.

[14] "Salary Averages." *The Grainger College of Engineering Electrical and Computer Engineering*, Electrical and Computer Engineering, https://ece.illinois.edu/admissions/why-ece/salary-averages. Accessed 21 Feb. 2023

[15] "IEEE - IEEE Code of Ethics." *IEEE*, IEEE, https://www.ieee.org/about/corporate/governance/p7-8.html. Accessed 9 Feb. 2023.

[16] "Code of Ethics." *Association for Computing Machinery*, Association for Computing Machinery, https://www.acm.org/code-of-ethics. Accessed 9 Feb. 2023.