University of Missouri – St. Louis

Beyond Fobs: A Secure QR Code-Based Access System

Patrick Jennewein, 2025

Table of Contents

Abstract

In this project, I propose a secure QR code authentication system that enhances traditional access methods.

While QR codes are widely used for authentication, conventional systems simply attempt to match a displayed QR code to a database with basic technology. These standard QR code detection methods can be unreliable in challenging environments such as in low-light conditions with user-induced motion blur. In the worst case, simply reading a QR code for authentication can also pose cybersecurity threats.

To address these issues, this project integrates two key components: (1) enhanced QR code detection to improve robustness in real-world conditions; and (2) a time-based system that ensures QR codes dynamically refresh to prevent reuse or replay.

This approach provides a secure and user-friendly authentication solution that can be deployed in various environments, offering improved protection while maintaining ease of use.

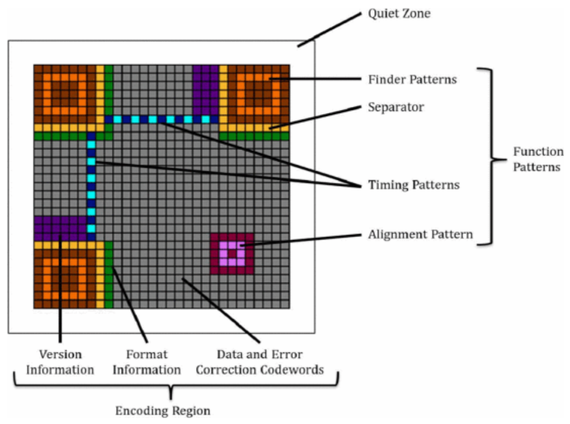
Introduction

"Quick Response" codes, otherwise known as "QR" codes, give users a two-dimensional matrix of white and black pixels (*see* Figure below) [1]. These codes – unlike more standardized one-dimensional barcodes – are capable of storing a larger amount of data, offer a much faster recognition, and can be read omni-directionally.

The Structure of QR Codes

QR Code Patterns

QR code functionality is simple. These codes contain four types of patterns, giving it structural integrity and enabling accurate data retrieval [1]. The *finder pattern* encompasses three of the four corners of the QR code and correctly orients the decoder with its perfect ratio of a 7 x 7 outer dark square, 5 x 5 inner light square, and 3 x 3 inner dark square. *Separators* encompass the white space around each of the three finder patterns. The *timing patterns* are alternating black and white pixels that form horizontal and vertical lines in 6th row and 6th column, respectively, to ensure proper alignment. Finally, *alignment patterns* consist of a 5 x 5 dark module surrounding a 3 x 3 light module, and 1-pixel dark module (*see* Figure below) [1]. These patterns distinguish QR codes from other visual encoding methods.



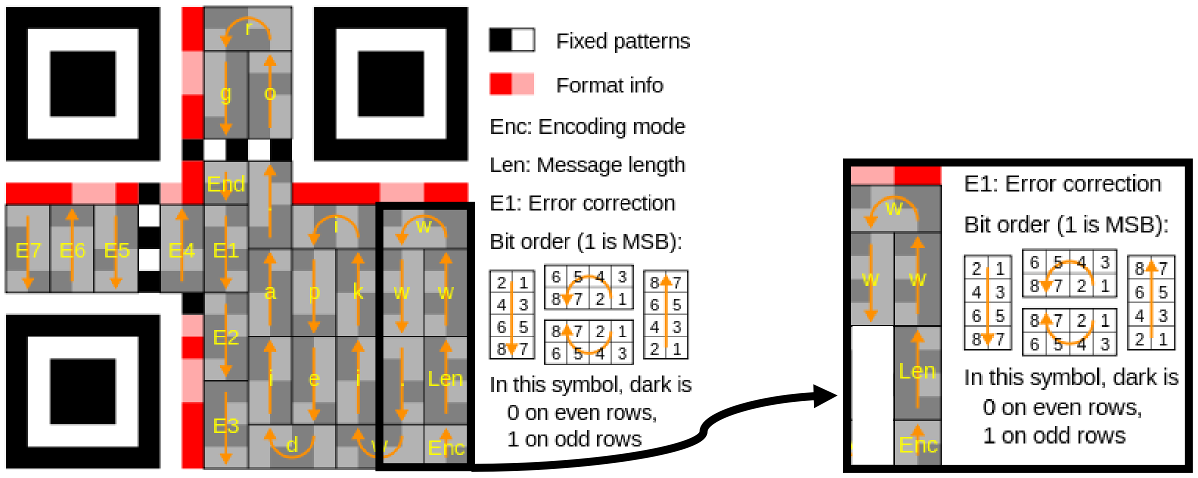
The Basics of Reading QR Code Data

To fundamentally understand QR codes, one must understand how the QR code is read from the camera:

1. The QR code image is scanned by the camera.
2. Each pixel[1] of the QR code is interpreted as a bit. A black pixel is interpreted as a **1** and a white pixel is interpreted as a **`0`**.
3. Bits are read following a zigzag pattern and grouped into 8-bit chunks – or bytes. For example, if the camera scans in an alternating set of black and white bits (`10101010`), this data is converted as the byte `170`.
4. These bytes are then interpreted as a message. If some data is corrupted or missing, *Reed-Solomon* algorithms use the mathematical relationship between bytes to detect and correct the errors before the message is displayed [2]

[1] A *pixel* can be more appropriately referred to as a *module*

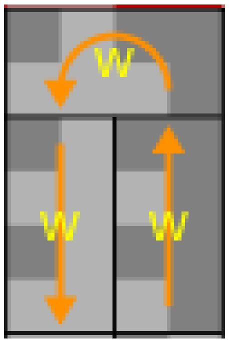
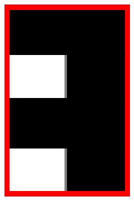
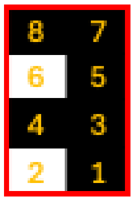
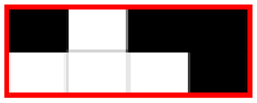
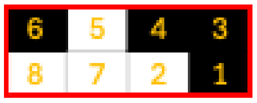
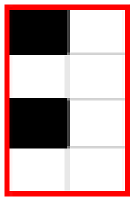
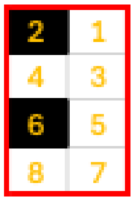
A simple example can be seen in below, a QR code holding a text-based URL [3]. Decoding a QR code starts from the bottom right corner, continuing up and around as per the orange arrows shown in a zigzag pattern. The first four bits, pictured as `Enc` below, gives the "encoding mode," of which there are 10 different modes as of this publication.[2] The second 8 bits, pictured as `Len` below, indicate how many characters should be read from the data stream.



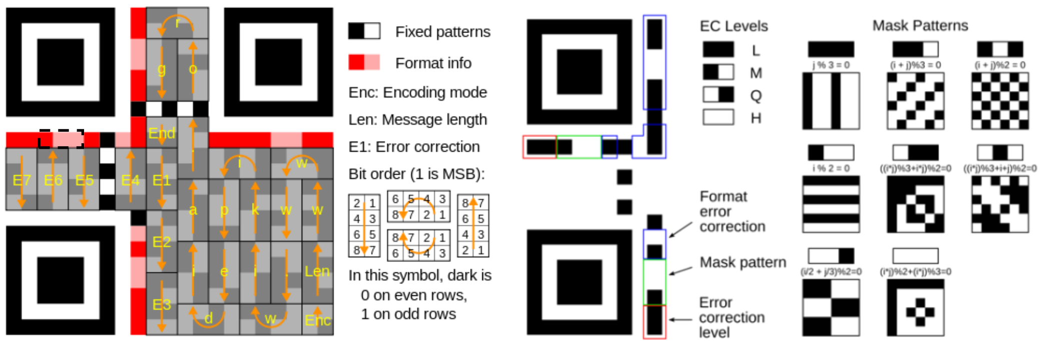
[2] Modes include `0001` for numeric encoding, `0010` for alphanumeric encoding, `0100` for byte encoding, `1000` for Kanji encoding, etc. [3]

Data Masking and Reversing

In analyzing the first 3 bytes (decoded as `w`, `w`, and `w`) from the figures below, one may notice that these bytes seemingly do not match despite all bytes supposed to be representing the same data: `w`. From a cursory glance, the bytes are as follows:[3]

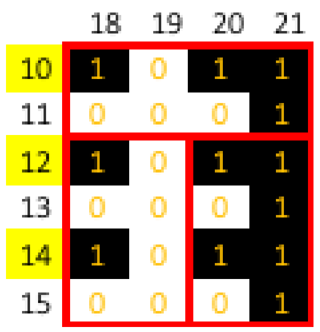
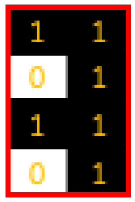
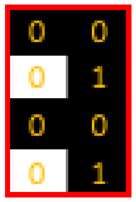
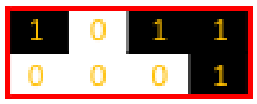
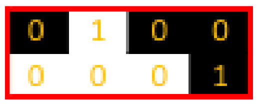
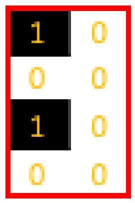
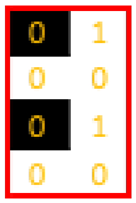
Original	Segments		
	Original Byte	Interpretation Order	Interpretation
			`10111011` (187)
			`10110100` (180)
			`01000100` (68)

This is because the *mask pattern* has not yet been applied. Masking occurs, "when an object (called the target) is affected by the presence of a second object (called the mask)" [4]. In this context, the *target* is the original byte of data while the mask is the predefined binary pattern applied to those bits (*see* Figure below). This masking process alters the visual layout the QR code to prevent problematic patterns – such as large areas of uniform bits or sequences that may resemble finder patterns. In this QR code, the mask pattern in the figure below is `001`, and can be found from the format information in the QR code (*see* Figure below).



[3] The three figures are recreations of the QR code for "www.wikipedia.com." Each figure is a "w" in "www".

Thus, for every even-numbered row (starting from the top left corner of the QR code), the bit will be flipped. The even-numbered rows are highlighted below and the corresponding encodings are also given.

Original	Segments		
	Original Byte	Update	Interpretation
 18 19 20 21 10 1 0 1 1 11 0 0 0 1 12 1 0 1 1 13 0 0 0 1 14 1 0 1 1 15 0 0 0 1			10001000 (136)
			10001000 (136)
			10001000 (136)

Though while each byte now matches `136`, the ASCII value for `w` is `119`. The discrepancy lies in that, after a QR code has been interpreted from its *masked* version, the reader must thereby *unmask* it – flipping the outputs once more. This means that each byte of `10001000` becomes flipped, and the reader finally interprets the byte as `01110111`, or `119 (w)`.

An issue arises when a QR code is not read perfectly: perhaps the screen holding the QR code has a smudge or water droplet, or perhaps the user has a finger over a portion of the QR code. When there is an issue with the QR code, *Reed-Solomon Codes* - which fall outside the scope of this paper – alleviate this by providing redundancy.

Comparing Traditional Barcodes to QR Codes

The previous analysis underscores how much more sophisticated that QR codes are versus traditional barcodes (*see* Figure below). Some features merit even further discussion.

2 34567 89999 3

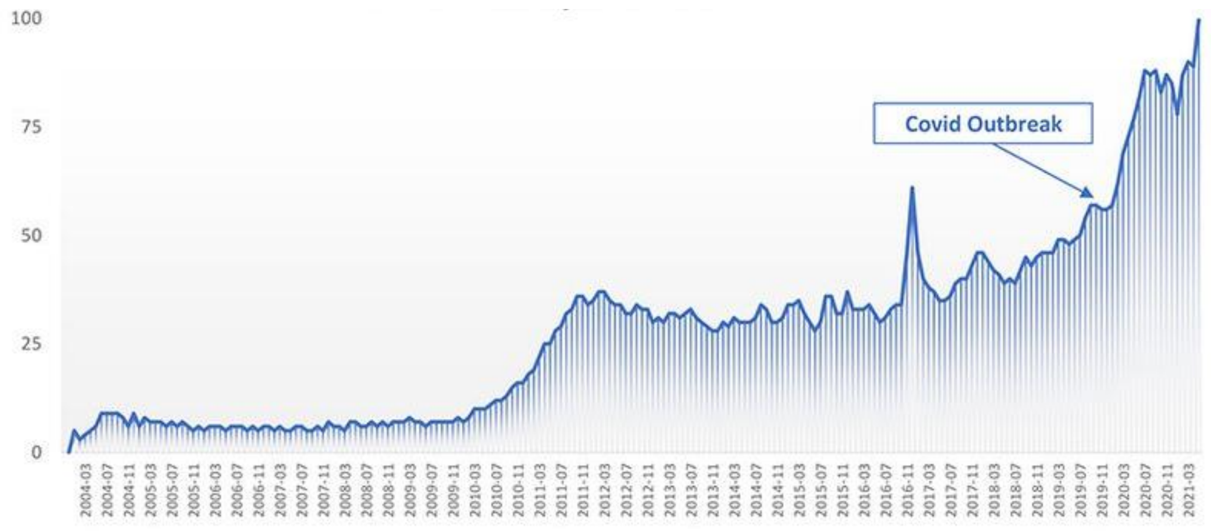
Traditional barcodes (e.g. a *universal product code* or *UPC*) can store only up to 20 digits. QR codes can store over 7,000 characters of data, including numeric data, alphanumeric data, binary data, and Kanji data. In addition, QR codes can encode URLs, e-mail addresses, and multimedia. As such, QR codes can encode 10 times more data than a barcode of the same size [5].

Furthermore, QR codes are "rapidly readable" from any direction, all made possible from the detection patterns in the 3 corners of the code. The structure of these codes also evades background interference. UPCs do not have these features. [5]

The Rising Popularity of QR Codes

While QR codes were created in 1994 as a means of tracking inventory, they become much more popular with the increasing use of smartphones.

According to a 2011 study [6], 20.1 million American smartphone owners used their device to scan a QR code in a 3-month average period. Among those tested, 44% scanned from a retail store and 59.4% scanned from home. This study hypothesized that the QR code was becoming the retailer's "secret weapon." For example, electronics retailer Best Buy adopted QR code integration on product tags to view and compare key features of the product.



But the popularity of QR codes in 2011 pales in comparison to what occurred during the COVID-19 outbreak (*see* Figure below). Between March 2020 and December 2020, 8.74 million users began using QR codes for mobile payment, and QR code payments represented 85% of all mobile payments in 2020 [7]. In addition, there was a 110% increase in QR code usage in general from 2019 to 2020, and a 28% increase from 2020 to 2021. Overall, the pandemic was a catalyst for the widespread adoption of QR codes, transforming them from a technology used in specific contexts to one that was much more ubiquitous.

QR Codes for Access Control

Most think about QR codes in the context of the user scanning a QR code and receiving data (a website, an e-mail address, a restaurant menu, etc.). However, the use of QR codes has greatly expanded even past this. QR codes have been widely adopted for access control contexts such as ticketing and building entry, offering an efficient and contactless method for verifying user credentials. In addition, unlike a physical fob or key card, QR codes only exist digitally and can be deactivated and revoked without the need to recover an object. This streamlines management and administrative overhead.

This same principle of secure, digital access control is increasingly relevant in academic environments, where schools have traditionally relied on physical fob systems. These physical fob systems are a common method of access control in schools, allowing students to unlock doors or check in to secure areas. One researcher, Pak Satanasaowapak, found that QR codes were a solution to access control, citing the following problems with other methods:

> The security of access to buildings or residences using technology can be guaranteed in various methods. Using a password is the easiest method, but it has the lowest security. Using radio frequency identification (RFID) is convenient for access, but it needs to be carried like a key. Biometrics is another method that has high security, but its limitation is the inability to access the security system remotely. Moreover, sending a password directly to a user by using a wireless network, such as Wi-Fi or Bluetooth, may run the risk of the data being stolen by hackers. Currently, many studies have proposed solutions to these issues. [8]

As such, QR code authentication for access control will be discussed in detail in this paper.

Problem Statement

Physical fob systems are often ineffective for access control as students frequently forget or misplace their fobs. In addition, building access is sometimes needed to be provided for one-time use and in circumstances in which providing a physical fob is difficult. This makes access control unreliable and induces administrative burdens to continually issue and replace fobs.

However, students almost certainly carry their smartphones and thus QR codes for access control seems to pose a solution.

Yet even if a fob-based system were to be replaced by a more robust QR code system for access control, this would also have shortcomings. Traditional QR code detection sometimes fails in real-world environments, specifically in low light conditions or under motion blur. Static QR codes also present a security risk, as they can be easily copied, reused, or shared.

Objective

The primary objective of this project is to develop a secure, robust QR code authentication system that can replace a fob system while addressing detection and security challenges. To achieve this, the system will integrate two features.

First, this project implements an enhanced pipeline to improve QR code readability under the real-world conditions including poor lighting and motion blur. Second, this project dynamically refreshes QR codes, allowing each QR code to be used only once.

Together, these improvements deliver a more secure, user-friendly, and resilient QR code authentication solution, suitable for use in schools and other environments needing access control.

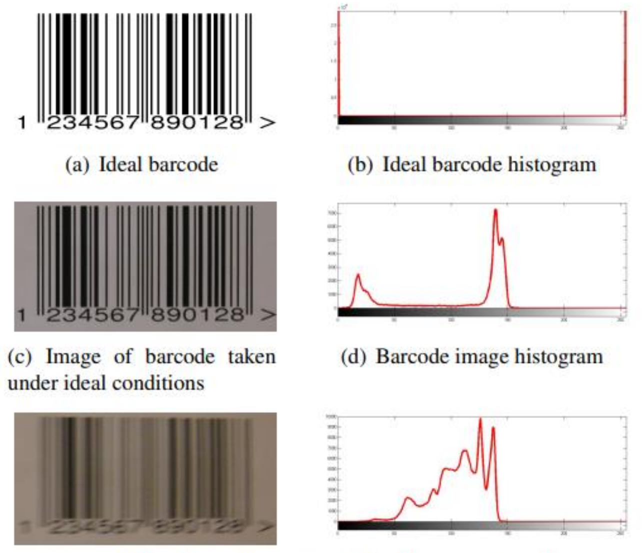
Literature Review

Decoding Techniques in QR-Based Authentication

A perhaps surprising amount of research has been conducted in making QR code and barcode reading more robust and reli.able.

Yahyanejad, et al. (2010)

In an Institute of Electrical and Electronics Engineers study conducted in 2010 by Yahyanejad, et al., researchers proposed methods to remove motion blur from UPC images specifically [9]. To understand their methods, it is import to examine precisely the problem they aimed to fix. In the figure to the right, three UPCs are presented: (a) is the ideal barcode with perfect black and white modules. These barcodes are unrealistic in real-world applications. (c) is an actual image produced by a camera without any motion blur. This is how UPCs are typically read: there is varying illumination, differences in light reflection, imperfect black and white pixels, etc. (e) is the image obtained from motion blur. As one can see, a significant difference between image (c) and (e) are exposed in the histogram: (e)'s histogram is significantly less consistent.


(a) Ideal barcode (b) Ideal barcode histogram

(c) Image of barcode taken under ideal conditions (d) Barcode image histogram

(e) Motion-blurred barcode (f) Blurred barcode image histogram

The deblurring method proposed is to basically guess the kernel that was used to "smear" the barcode, deconvolute the image using that same kernel and mathematically reverse the effect of the blur to get a sharp image back, determine if that kernel is the same as the original barcode, and repeat with different kernels. This results in a simple 4-step process.

First, the UPC was converted from a 2-dimensional image to a 1-dimensional image. Because UPCs are simply a series of vertical white and black lines, whether or not the top, middle, or bottom of the vertical line is obtained is irrelevant. Therefore, researchers averaged the pixels of each vertical column to one pixel.[4]

$$b(y) = \frac{1}{N}\sum_{x=1}^{N} b(x,y)$$

[4] More specifically, researchers averaged approximately 5% of each vertical line, in the middle of the UPC.

As a consequence, the kernel used will also become 1-dimensional.

$$k(y) = \sum_{x=1}^{L} k(x, y)$$

Notably, this is an optional step and, because QR codes are indeed 2-dimensional, this step would not need to be replicated if applied to a QR code.

Second, a kernel – which consists of the length of the blur – is guessed. Assuming a length of L in the kernel, the researchers started with a uniform kernel, assuming that the blur was spread over evenly across L pixels.

$$\mathbf{k}_0 = [\underbrace{\frac{1}{L} \quad \frac{1}{L} \quad \dots \quad \frac{1}{L}}_{\text{L times}}]$$

Third, how close this 1-dimensional image is to a real UPC is evaluated using a target function. Luckily, UPCs have specific conventions that make this possible. The histogram of a barcode without motion blur has a clear bimodal shape with one intensity modeling black and the other intensity modeling white. Though an ideal UPC will have a perfect bimodal distribution with one intensity being perfect white and the other intensity being perfect black, a UPC in the real-world will essentially never have this distribution and, instead, will have much more gray and an intensity distribution much closer to the mean. The target function should be a histogram with two peaks where the variance of each peak is small while the distance between them is large. The researchers present the following Bivar equation, which is designed to evaluate how similar a deblurred 1-dimensional signal is to a clean barcode.

$$Bivar(\mathbf{o}) = \begin{cases} \frac{Var(\mathbf{o}_1)+Var(\mathbf{o}_2)}{[mean(\mathbf{o}_1)-mean(\mathbf{o}_2)]^2}, & |mean(\mathbf{o}_1)-mean(\mathbf{o}_2)| \leq 1 \\ \infty, & |mean(\mathbf{o}_1)-mean(\mathbf{o}_2)| > 1 \end{cases}$$

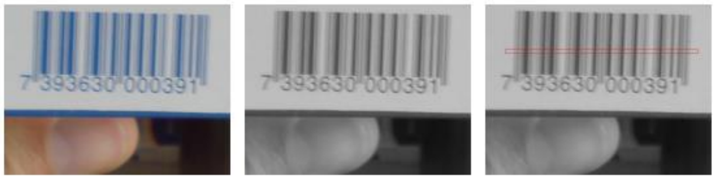
$$\text{where} \quad \begin{cases} \mathbf{o}_1 = \{\mathbf{o} < mean(\mathbf{o})\} \\ \mathbf{o}_2 = \{\mathbf{o} > mean(\mathbf{o})\} \end{cases},$$

o_1 contains all values less than the mean (the darkest pixels) and o_2 contains all values greater than the mean (the lightest pixels). $var(o_1) + var(o_2)$ thereby calculates the sum of the variances between these two groups, demonstrating how tightly values are clustered around each peak. This sum is divided by the square of the differences between their means, to measure how far apart these peaks are. This is called a *Bivar score*. A lower score means that the barcode has a tight, well-separated set of bimodal peaks. A higher score means that the proposed kernel is invalid.

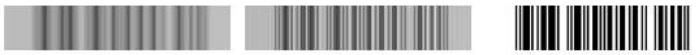
Fourth, a small, random noise vector[5] is added to a new kernel. The new kernel's Bivar score is compared to the previous kernel's score, and the better of the two is kept. This process is repeated many times; the researchers completed this process 1,000 times in fact.

[5] This vector is controlled by a variable that shrinks over time, so the guesses become better with each attempt.

The results of this algorithm were largely successful and an example can be seen in the figure below. 138 images taken with a 3.2 megapixel smartphone camera; 45 images (32.6%) were decoded successfully without any deblurring applied, leaving 93 images unsuccessfully decoded without any deblurring applied. Of those 93 images, 44 were successfully decoded after deblurring. Moreover, the deblurring algorithm had no effect on images that would have otherwise successfully been decoded without deblurring, so the deblurring algorithm led to almost a 50% increase in successful decoding [9].[6]



(a) Blurred image acquired by camera (b) Grayscale conversion (c) One-dimensional conversion by averaging or cropping a cross section



(d) Perpendicular replication of one-dimensional barcode (e) After deblurring by our algorithm (f) Fine tuning by converting to exact black and white

Rioux, et al. (2021)

In 2021, Rioux et al. explored more advanced deblurring techniques that rely on taking what we already know about QR codes, including the strict patterns and structure rules [10]. In other words, while many techniques such as those demonstrated in Yahyanejad guess the kernel somewhat randomly until it approximates correctness and then use it to unblur the image, Rioux proposes to augment this strategy by treating the code as a probabilistic object. In other words, it is a much more sophisticated trial and error process: rather than randomly selecting a fixed blur kernel, Rioux et al.'s algorithm uses a structured, iterative process — estimating the most likely QR code via probability, and refining the kernel through optimization.

One key conceptual shift is that Rioux's algorithm does not attempt to uncover a single most likely QR code. Instead, it analyzes many valid QR codes that follow the symbology rules and assigns a probability to each one. This mitigates the risk of committing too early to a potentially incorrect guess and allows the algorithm to remain flexible. This algorithm is a four-step process.

First, the image must be modeled because, in order to understand how to reverse a blur, one must understand how it occurred in the first place. This modeling is by the following equation:

$$\boldsymbol{b} = C\boldsymbol{x} \equiv \boldsymbol{c} * U\boldsymbol{x}$$

[6] 45 images were decoded successfully without deblurring and an additional 44 images were decoded successfully with deblurring (without any effect on the original 45). Thus, 89 of the 138 images (64.4%) could be decoded with deblurring versus only 45 (32.6%) being able to be decoded without deblurring.

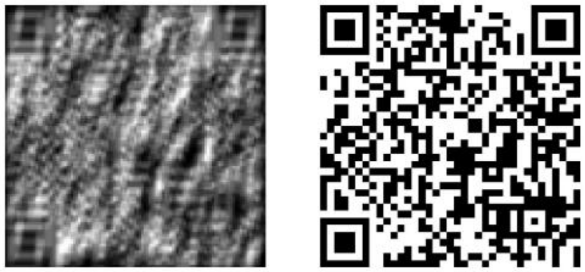
In this equation, x is the original QR code, U is the upsampling matrix, c is the blur kernel that describes how the camera shook or how the image got smeared, $*$ is the convolution operation that spreads to each pixel to represent the smearing, and b is the resulting blurry image.

Second, the algorithm defines a probability function over all possible QR codes and estimates the most likely QR code as a distribution. In simplest terms, the algorithm considers all possible QR codes that follow the QR code rules and figures out how likely each one is. This is formalized using *Kullback-Leibler divergence*[7] to stay close to known QR code structure.

Third, and because the algorithm cannot feasibly iterate through every possibility in general, the algorithm will solve a *dual problem* with a shortcut known as the *Fenchel-Rockafellar* duality. This mathematical tool elicits that one can solve a huge problem by solving a much smaller, easier version of the problem that leads to the same answer [11]. Here, the larger problem is iterating through every QR code possible in existence; the smaller problem that is *actually* solved in the algorithm is iterating over the possibilities for a single pixel. The problem goes from trillions of possibilities of QR codes to 2 possibilities for each pixel, repeated over however many pixels are in the QR code – from over a trillion problems to only a few thousand problems.

Fourth, once the probabilities of each pixel are uncovered, the entire QR code is thereby reconstructed. The algorithm reconstructs the entire QR code by combining these probabilities into a full image. This grayscale image is then thresholded — turning modules with values greater than 0.5 into white, and others into black — to produce a final binary QR code for scanning [10].

The results of this algorithm are impressive, as can be seen in the figures below which represent a blurred image and the unblurred result.

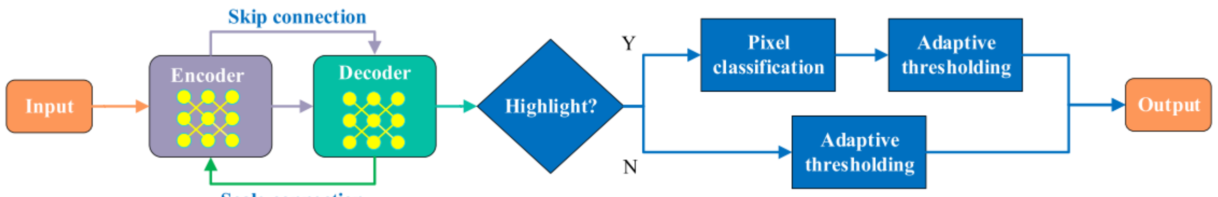


Li, et al. (2021)

Li, et al. cited the previous papers and was critical of their lack of focus on speed [12]. In the words of the authors: "Traditional algorithms are not robust enough when dealing with images severely affected by non-uniform motion blur and usually have poor performance in realistic scenes. Furthermore, they are so time-consuming due to their complex physical models that sometimes they are unacceptable for practical use." Thus, Li thereby proposed a new algorithm that combines feature extraction based on deep learning and an improved adaptive thresholding when light is uneven.

[7] In simple terms, Kullback-Leibler divergence is a way to measure how different two probability distributions are from each other. In the context of QR codes, this is used to ensure that the probabilities uncovered in the algorithm approximate what we know about QR code structure. This ensures that the probabilities are not unrealistic.

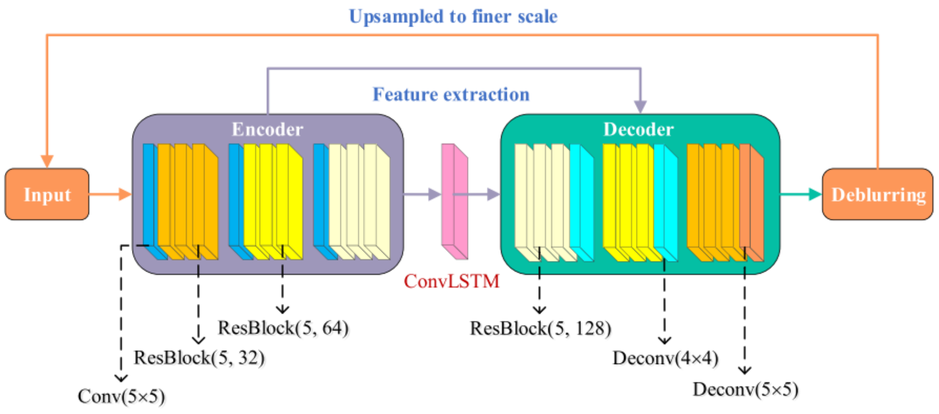
Perhaps the greatest difference between Li and the previous methods evaluated is the deep learning architecture. This is demonstrated below [12].



(a) The proposed algorithm

A blurry or otherwise low-quality QR code is given as the input. Then, a deep learning model performs feature extraction and deblurring. This step is complicated, and thus another figure (*see* Figure below) has been provided for a more in-depth evaluation.

Once the input is provided, the encoder analyzes the blurry, low-quality image and pulls out the most important details. This encoder includes a 5 x 5 convultion filter that scans over the image and looks for patterns. Then, two residual blocks – one of 32 filters and one of 64 filters - further detect features.[8] These layers compress the image into a compact representation of key features. After this, a Convolutional Long Short-Term Memory[9] will analyze all the features and fill in missing details. The decoder will attempt to reconstruct a clean image. Another residual block of 128 filters is utilized. During deconvolution, the image is upsampled and stretched back to normal size. The result is a deblurred image.

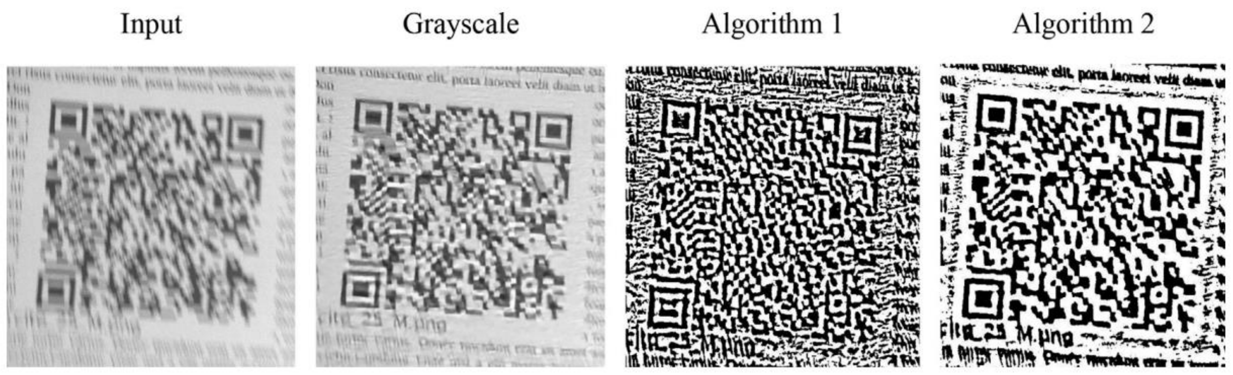


(b) The feature extraction framework

[88] A residual block (also known as a "ResBlock") was introduced in 2015 by Kaiming He as a "key architectural component in modern deep learning models" that enables the training of deep neural networks by incorporating skip connections. A deeper analysis of ResBlocks can be found in Choudhary [16].

[9] A Convolutional Long Short-Term Memory (also known as "ConvLSTM") is a convolutional neural network that not only analyzes an image, but also "remembers" what is saw before. A deeper analysis of ConvLSTM can be found in Xavier [17].

After deep learning is performed, the highlight detection step checks if an image contains areas that are overexposed. Importantly, depending on whether or not the image is overexposed, one of two different algorithms will execute. If no overexposed regions are detected, adaptive thresholding is applied directly by “Algorithm 1” and an output image is created. If there are overexposed regions, “Algorithm 2” is used with enhanced thresholding and pixel classification. Algorithm 2 is an improved, more powerful, and more accurate version of Algorithm 1, which is much more robust in difficult conditions. Because of this, the author recommends Algorithm 2.



Discussion of Enhanced QR Decoding Techniques

While extensive research has been conducted, issues arise when utilizing previous techniques for this project specifically.

An issue with the Yahyanejad research is its speed. The computation time for a UPC with a width of 1,000 pixels was approximately 8 seconds for 1,000 iterations and 2 seconds with 10 iterations. In fact, “*No attempts* to optimize the code for speed were made” [9]. Rioux suffers similar problems for its application to the problem at hand. According to the authors themselves, the algorithm is computationally heavy and “any implementation would require a significant amount of preprocessing” [10]. Moreover, both authors assume a uniform blur. While Li’s algorithm performs better than the other two, there are also concerns about speed. In Li’s algorithm, the reported runtime was still 23.24 seconds [12].

For the purposes of my project, speed is essential and the algorithms provided simply are not fast enough. Moreover, the amount of processing power available is relatively low and the likelihood of a uniform blur is small. This project is meant to be tested when perhaps an entire line of students is waiting to gain access to a building. In addition, this project will not be used with a high-performing computer, but rather a consumer-grade computer available on the budget of a school.

Overall, it is clear that – while current research is certainly insightful – it is certainly not definitive or universally adaptable to the real-time, resource-constrained environments in which the QR authentication system I propose must often operate.

Security Vulnerabilities in QR-Based Authentication

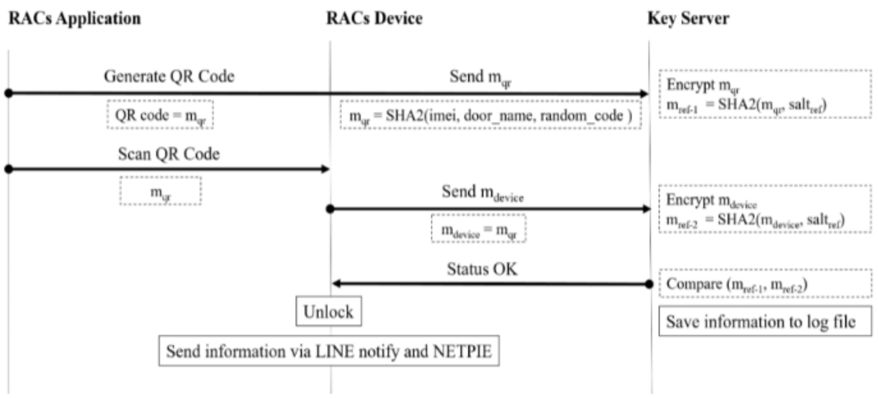
Before discussing ways in which QR code-based access control can be made more secure, it is important to understand why some adopters of such a system would be skeptical.

In *QR Code Security: A Survey of Attacks and Challenges for Usable Security*, researcher Katharina Krombholz elucidates some potential issues with QR codes' security. While much of this paper was focused on a user *reading* QR codes for data (rather than my proposed application, in which a scanner would read a QR code for access control), Krombholz does propose some issues that would be relevant:

> *If a scanning application uses a database to store scanned entries, it is possible to execute a sql injection by scanning values such as 1' OR 1=1 – in order to circumvent authentication mechanisms. Furthermore, [there is] a threat of browser-based exploits, XSS attacks and command injections via QR codes.* [13]

In other words, when access control systems forward QR data to a database, attackers can inject malicious code to exploit the database. In one example, Krombholz reiterated a 2012 study in which a QR code was encoded to "to trigger the execution of the MMI [Man-Machine Interface] code which erases all data from the mobile device."

Regardless of any potential security vulnerabilities, studies have also shown the success of a QR code-based access control system. In fact, Satanasaowapak et al. implements a system in which QR codes are dynamically generated, secured, and immediately invalidated after each use [8]. The architecture is essentially as follows (*see* Figure below).


RACs Application
RACs Device
Key Server
Generate QR Code
Send m_{qr}
QR code = m_{qr}
m_{qr} = SHA2(imei, door_name, random_code)
Encrypt m_{qr}
m_{ref-1} = SHA2(m_{qr}, $salt_{ref}$)
Scan QR Code
m_{qr}
Send m_{device}
m_{device} = m_{qr}
Encrypt m_{device}
m_{ref-2} = SHA2(m_{device}, $salt_{ref}$)
Status OK
Compare (m_{ref-1}, m_{ref-2})
Unlock
Save information to log file
Send information via LINE notify and NETPIE

First, the smartphone app generates a QR code, which encodes a message (`m_qr`) that includes the user's device IMEI, the door name, and a randomly generated one-time code. This is encoded into a QR code that is valid for 5 minutes. The QR code is scanned by the device and is sent to validation.

Second, the Residential Access System Device installed at the door sends the QR code to the key server and awaits the decision.

Third, the server validates whether the QR code is legitimate via hashes and provides an access decision. If the QR code is confirmed as valid, a signal is sent back to the RAC device. Regardless, the attempt is logged in a secure file.

Finally, the Residential Access System Device installed at the door Is either locked or unlocked as determined by Step #3.

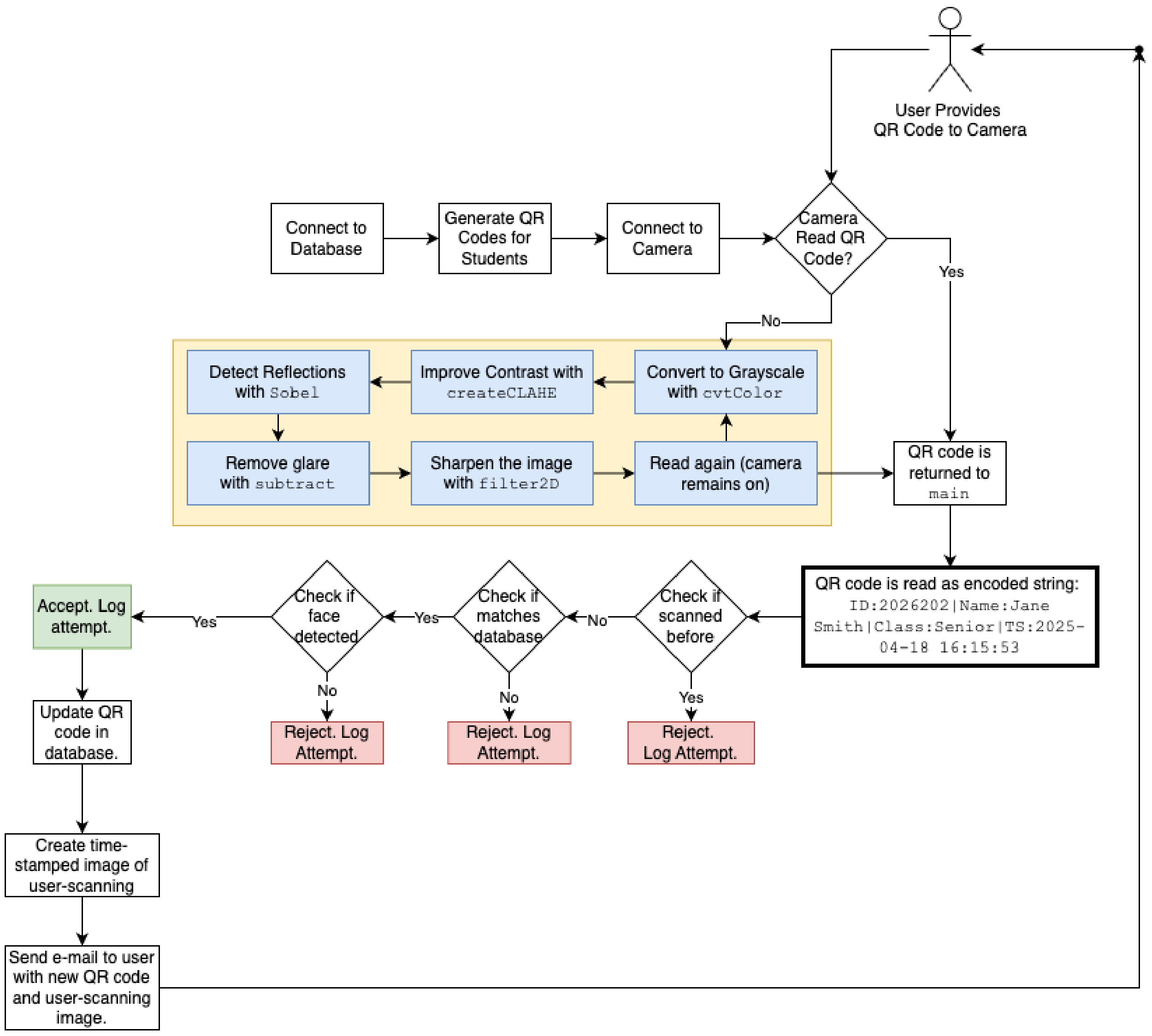
Overall, this presented a secure method of access control: the flow of data prevented attackers from successfully spoofing or reusing QR codes - disabling a previous QR code after authentication and only checking a QR code to a database. From a usability standpoint, the entire process took an average time of only 5.63 seconds to authenticate, proving itself to be a stable, accurate, and reliable method of access control.

Methodology and Implementation

Given the limitations and trade-offs identified in the literature, I developed a lightweight, real-time QR authentication system that more reasonably balances reliability with efficiency.

System Overview

The final system designed is the following figure:


User Provides QR Code to Camera
Connect to Database
Generate QR Codes for Students
Connect to Camera
Camera Read QR Code?
Yes
No
Detect Reflections with Sobel
Improve Contrast with createCLAHE
Convert to Grayscale with cvtColor
Remove glare with subtract
Sharpen the image with filter2D
Read again (camera remains on)
QR code is returned to main
QR code is read as encoded string: ID:2026202|Name:Jane Smith|Class:Senior|TS:2025-04-18 16:15:53
Check if scanned before
Yes
Reject. Log Attempt.
No
Check if matches database
No
Reject. Log Attempt.
Yes
Check if face detected
No
Reject. Log Attempt.
Yes
Accept. Log attempt.
Update QR code in database.
Create time-stamped image of user-scanning
Send e-mail to user with new QR code and user-scanning image.

The software architecture of this system is as follows:

`main.py`	The main script that runs the QR authentication system. Captures video, detects QR codes and faces, verifies the scan, logs data, updates the database, regenerates QR codes, and displays results.
`config.py`	Stores configuration variables. Centralizes constants used across the project.
`db_utils.py`	Handles all database-related operations: initializing tables, inserting student records from a CSV file, generating initial QR codes, and logging scan events to a CSV log file.
`email_utils.py`	Sends emails when a QR code is updated. Attaches both the new QR code image and optionally the image captured at scan time. Uses Gmail via SMTP.
`image_utils.py`	Contains image-processing logic: detecting faces, saving scan images, and decoding QR codes with preprocessing to improve reliability in low-quality frames.
`qr_utils.py`	Generates and updates QR codes for each student. Embeds metadata into the QR, saves the image, converts it to binary, updates the database, and calls `send_email`.
`sound_utils.py`	Provides audio feedback to the user by playing different sound effects (success or failure) using `afplay` on macOS.
`webcam.py`	Lets you choose between the internal or external webcam. Simplifies webcam management with a small helper function.
`secure.py`	Stores email credentials and recipient details. Sensitive information used by `email_utils.py` for sending email notifications.
`fake_data.csv`	CSV file used to initially populate the student database. Includes student IDs, names, and class names. Only used during setup if the table doesn't already exist.

Technologies Used

In creating this project, the following technologies were used:

Python 3	Core programming language used to build the application.
OpenCV (`cv2`)	Used for video capture, image preprocessing, face detection, grayscale conversion, and drawing rectangles on frames.
Pyzbar (`pyzbar`)	Decodes QR codes from camera frames.
`qrcode` library	Generates QR code images from student data.
SQLite3	Lightweight embedded database used to store student info and QR metadata.
`csv` library	Reads studenxt data from `.csv` files and logs scan events to a scan history log.
`numpy` library	Used for matrix operations in image processing (e.g., sharpening, gradients).
`EmailMessage` / `smtplib` / `mimetypes`	Built-in Python libraries for sending QR code update emails with attachments.
`afplay` (macOS)	Used to play success/failure sounds via system audio (macOS-specific).
Haar Cascades (`cv2`)	Detects faces in the camera feed using pre-trained models.

System Libraries (`os`, `sys`, `datetime`, `subprocess`, `time`)	Standard Python libraries for file handling, timestamping, subprocess control, and general utilities.

Discussion and Analysis

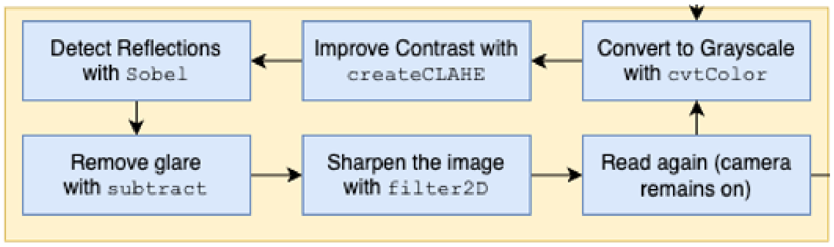
To ensure robust authentication in real-world environments, this application employs an enhanced QR code detection, combined with verification safeguards designed to resist spoofing and unauthorized access.

Enhanced QR Code Detection Analysis

While traditional QR scanners often fail in challenging conditions such as motion blur, uneven lighting, or glare, this implementation uses a multi-step enhancement strategy to improve detection reliability before attempting to decode a QR code. The figure below shows one of the central components of the algorithm utilized in this application.

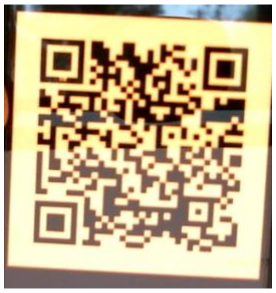
Advanced Detection

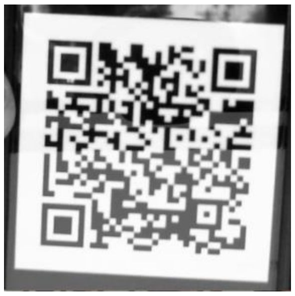
The core QR code detection algorithm from OpenCV and pyzbar is augmented with additional image transformations to mitigate issues that commonly affect QR code readability. The detection process begins with grayscale conversion to reduce complexity, filtering the QR code structure to the minimum amount of values. Following this, *Contrast Limited Adaptive Histogram Equalization (CLAHE)* is applied to enhance contrast. It does so by dividing the image into small regions (typically 8×8 pixels), and performing histogram equalization independently within region. Unlike more traditional histogram equalization, which may amplify noise or overcorrect lighting inconsistencies, CLAHE clips each local histogram at a predefined threshold to prevent over-amplification. The clipped excess is then redistributed across the intensity range to maintain smooth tonal transitions. This localized approach ensures that subtle variations in black-and-white module boundaries are enhanced without distorting well-exposed regions. As a result, QR code features become more distinguishable, increasing the likelihood of successful decoding under real-world conditions.

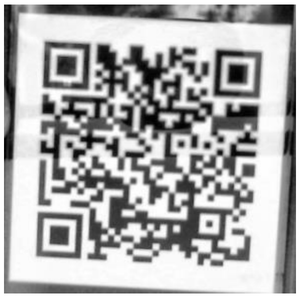

Detect Reflections with Sobel
Improve Contrast with createCLAHE
Convert to Grayscale with cvtColor
Remove glare with subtract
Sharpen the image with filter2D
Read again (camera remains on)

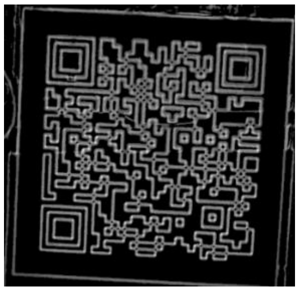
To further address glare and motion blur, the application calculates the gradient magnitude of the image using Sobel filters. A Sobel filter emphasizes regions of an image where there are significant changes in intensity, highlighting edges. It works through two 3×3 kernels: one detects horizontal changes (x), and the other detects vertical changes (y). The resulting gradient from the Sobel filter is subtracted from the enhanced grayscale image to suppress these glares. Subsequently, a sharpening kernel is applied to reinforce edge clarity and module boundaries, increasing the likelihood that the QR code will be detected successfully [14].

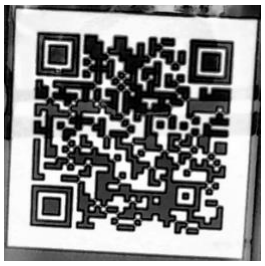
This full enhancement pipeline is wrapped in `safe_decode()`,a function that attempts basic decoding first, and only falls back to advanced preprocessing if the initial attempt fails. This ensures that frames without distortion are processed quickly, while frames with noise or blur are handled more robustly without further unnecessary, computation.

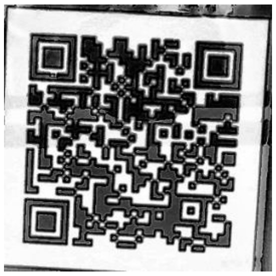

Original QR Code


Grayscale


CLAHE


Subtraction


Reflection Removal


Final QR Code

Face-Dependent Validation

To further strengthen the authentication process, the system necessitates facial presence detection using *Haar cascades*. Quite simply, A Haar cascade is a tool that helps the computer recognize specific things in an image—in this case, a human face. It works by scanning the image for patterns it has learned from thousands of examples, such as the shapes and shading commonly found in faces. Instead of examining every part of the image from the start, it quickly skips over areas that clearly don't look like a face and takes a closer look at areas that might. This makes it both fast and accurate. [15] In this program, I use the function `cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')` to load an openCV built-in file that already knows how to detect faces. A QR code is then only considered valid if a human face is present in the same frame. Notably, *the application is not matching a face to a database*, avoiding legal ramifications of holding this data.

As such, this layer of validation provides two key benefits: (1) it ensures that the code is being presented by a real person and not simply displayed on a device or printed artifact, and (2) it allows the system to associate a scan with a visual confirmation. This is especially relevant for access control.

Security Analysis

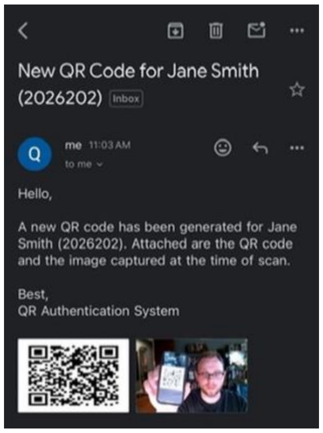
Several features of the system contribute to its suitability for secure access control deployment:

First, each QR code encodes a timestamp and is updated dynamically. Once scanned, the QR is invalidated and replaced. Any reuse attempt of an old QR will be rejected. Every scan attempt—successful, outdated, or invalid—is logged with metadata (ID, name, class, time, and result). This log provides a traceable history for administrators and supports future analytics or incident review.

Second, the system maintains a per-user cooldown window that prevents multiple scans from being accepted in rapid succession, thereby mitigating the risk of double-entry or repeated spoof attempts.

Third, QR codes must correspond to records in an SQLite database, and the system validates that the associated student ID exists, the code has not expired, and the formatting is correct. Even more importantly, the application is not actual *reading* the QR code but rather *comparing* the QR code to the one in the system; this allows the system to not have the same vulnerabilities of those noted previously in Krombholz.

Fourth, upon successful scan and code regeneration, the system emails the updated QR code to the student (who is authenticated with two-factor authentication of the school's e-mail address) and attaches the captured webcam image. This offers an additional validation step and ensures that updates are traceable even if the device is later compromised.


New QR Code for Jane Smith (2026202)

me 11:03 AM
to me

Hello,

A new QR code has been generated for Jane Smith (2026202). Attached are the QR code and the image captured at the time of scan.

Best,
QR Authentication System

Overall, the system is designed with layered defenses, combining enhanced computer vision with real-time validation and robust backend logging. These features together make the solution well-suited for school use.

Challenges

While the QR code authentication system demonstrates strong potential for secure and user-friendly access control, several challenges remain that affect reliability and performance.

One key limitation continues to be the variability of real-world image conditions—QR codes presented at odd angles, in low light, or with glare can still fail to decode, even with preprocessing techniques like CLAHE and gradient suppression. Processing speed is another constraint, especially on consumer-grade hardware, where image enhancement, decoding, and database logging must occur in near real-time to avoid frustrating the user and ultimately leaving the application unused.

For the system to reach production, these areas will need continued refinement and performance optimization.

Conclusion

This project demonstrates a QR code–based authentication of quite reasonable practicality and security. It combines advanced image preprocessing techniques with real-time verification safeguards. Unlike standard systems that simply attempt to decode a QR code as-is, this application enhances the image using contrast boosting, glare reduction, and sharpening, all tailored to improve performance under challenging conditions such as low lighting, motion blur, or reflections. Additionally, requiring the presence of a human face in the frame adds a meaningful layer of protection against spoofing, ensuring that the system not only recognizes a valid code, but also confirms that it is being presented by a live user.

While the solution is not without its challenges—including sensitivity to lighting conditions and the limitations of real-time performance on modest hardware—it offers a clear improvement over basic QR scanning methods. With further refinements, this system can become even more robust and scalable for access control in schools.

Works Cited

[1] S. Tiwari, An Introduction to QR Code Technology, Bhubaneswar: 2016 International Conference on Information Technology (ICIT), 2016, pp. 39-44.

[2] B. S. Bajaj, "Reliability on QR Codes and Reed-Solomon Codes," Canary Islands, 2025.

[3] Wikipedia, "QR Code," 14 October 2011. [Online]. Available: https://commons.wikimedia.org/wiki/File:QR_Character_Placement.svg. [Accessed 27 March 2025].

[4] J. Zhang, D. Li, J. Jia, W. Sun and G. Zhai, "Protection and Hiding Algorithm of QR Code Based on Multi-channel Visual Masking," *2019 IEEE Visual Communications and Image Processing (VCIP),* pp. 1-4, 2019.

[5] J. H. Chang, "An introduction to using QR codes in scholarly journals," *Sci Ed,* vol. 1, no. 2, pp. 113-117, 2014.

[6] Retail TouchPoints, "QR Code Trends And Early Adopters," 14 February 2012. [Online]. Available: https://www.retailtouchpoints.com/topics/digital-marketing/mobile-marketing/qr-code-trends-and-early-adopters?utm_source=chatgpt.com. [Accessed 8 April 2025].

[7] M. Tu, L. Wu, H. Wan, Z. Din, Z. Guo and J. Chen, "The Adoption of QR Code Mobile Payment During COVID-19: A Social Learning Perspective," *Frontiers in Psychology,* 2022.

[8] P. Satanasaowapak, K. Witawat, S. Promlee and A. Vilamat, "Residential access control system using QR code and the IoT," *International Journal of Electrical and Computer Engineering (IJECE),* vol. 11, no. 4, pp. 3267-3274, 2021.

[9] S. Yahyanejad and J. Strom, "Removing motion blur from barcode images," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Francisco, c2010.

[10] G. Rioux, C. Scarvelis, R. Choksi, T. Hoheisel and P. Marechal, "Blind Deblurring of Barcodes via Kullback-Leibler Divergence," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE,* vol. 43, no. 1, pp. 77-78, 2021.

[11] E. Dohmatob, "Fenchel-Rockafellar duality theorem, one ring to rule'em all!," 31 October 2019. [Online]. Available: https://dohmatob.github.io/research/2019/10/31/duality.html. [Accessed 13 April 2025].

[12] J. Li, Z. Dong, M. Zhou and C. Zhengai, "A motion blur QR code identification algorithm based on feature," *Neurocomputing,* pp. 351-361, 2022.

[13] K. Krombholz, P. Fruehwirt, P. Kieseberg and I. Kapsalis, "QR Code Security: A Survey of Attacks and Challenges for Usable Security," *Lecture Notes in Computer Science,* 2014.

[14] ScienceDirect, "Sobel Filter," [Online]. Available: https://www.sciencedirect.com/topics/computer-science/sobel-filter. [Accessed 19 April 2025].

[15] A. Mittal, "Medium," December 2020. [Online]. Available: https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d. [Accessed 19 April 2025].

[16] N. S. Choudhary, "A Comprehensive Guide to Understanding and Implementing Bottleneck Residual Blocks," Medium, 2 April 2023. [Online]. Available: https://medium.com/@neetu.sigger/a-comprehensive-guide-to-understanding-and-implementing-bottleneck-residual-blocks-6b420706f66b. [Accessed 13 April 2025].

[17] A. Xavier, "An introduction to ConvLSTM," Medium, 25 March 2019. [Online]. Available: https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7. [Accessed 13 April 2025].