Automated IC Card Dispenser System for Residential College

By

Zhirong Chen (zhirong4@illinois.edu)

Xiaoyang Chu (xzhu458@illinois.edu)

Zicheng Ma (zma17@illinois.edu)

Dongshen Ye (dye7@illinois.edu)

Sponsored by

Asst. Prof. Meng Zhang

Project #1

Project Proposal for ECE445/ME470, SP2024

Mar 10th, 2024

Abstract

There is a significant number of cases where students accidentally left their room and locked the door while forgetting to bring their cards with them. Moreover, should it happen after the office hours of the RC office, the student will need to call the security personals to handle the situation. The current issuance procedure also failed to fully identify the student which introduces potential security implications. In response to the demand, we proposed and developed a system that automatically identifies an authorized user and issues a temporal access IC card.

Contents

1 Introduction

In the recent analysis of the access card system at our residential college, it was found that between February 25th and March 3rd, 2024, there were 287 incidents where residents requested temporary access cards due to misplacing their original cards. This number excludes requests outside of office hours, which involve a cumbersome process requiring intervention from security personnel. This procedure not only poses challenges due to the diverse linguistic backgrounds of our student population, but also raises significant safety concerns due to the risk of unauthorized access, as the current method of issuing temporary cards lacks rigorous identity verification.

To address these issues, we propose the implementation of an automated system that enhances both security and convenience. This system includes a server and multiple kiosk terminals strategically placed within the college. As depicted in Fig. 1, this system is designed to integrate two robust methods of authentication: QR code verification and facial recognition.

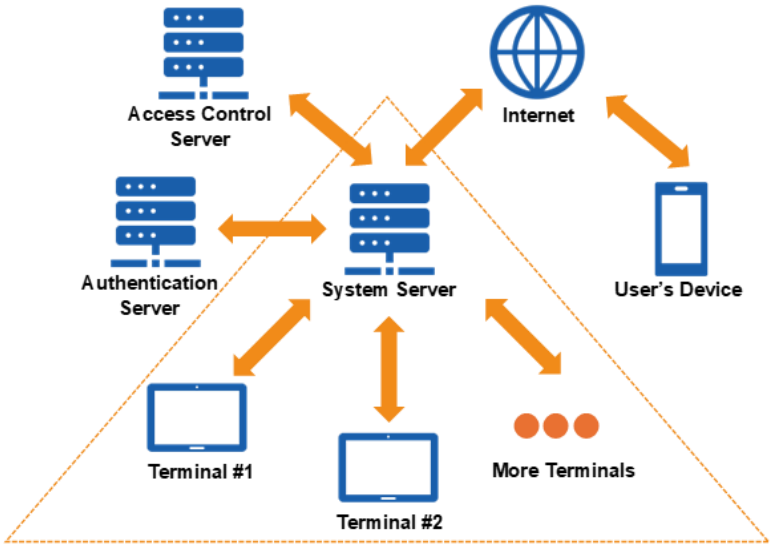

Fig. 1 Solution Structure

The division of the project into distinct blocks—server management, kiosk terminal interface, Kiosk mechanical subsystem, QR code generation, and facial recognition—allows for focused development and troubleshooting in each area, ensuring that each component meets its specific performance criteria. In general, the following criteria should be met:

- Reliable, robust, and convenient authentication methods should be adopted to keep the issuance process secure. The system should be invulnerable to conventional cyber-attacks. A successful issuance process should take less than 60 seconds.
- The mechanical card dispenser should have a failure rate less than 1/500.
- The system should support multiple languages and can be easily maintained and managed.

The mechanical subsystem is supposed to dispense cards automatically. Originally the mechanical subsystem was designed to be hang on the shell of the card dispenser. But during the designing process, the mechanical subsystem is set up on the ground with two pieces of board supporting its bottom to keep its balance so that the mechanical subsystem can operate stably.

Throughout the development of this system, we adhered strictly to performance requirements that mandate a seamless user experience and robust security measures.

2 Design

The entire project consists of two major parts: the KIOSK terminal and the server.

2.1 Design Procedure

2.1.1 KIOSK Terminal

The KIOSK Terminal consists of two subsystems: the user interaction subsystem and the mechanical subsystem. The composition of the KIOSK terminal is shown in Fig. 2 Block Diagram of KIOSK Terminal Hardware.

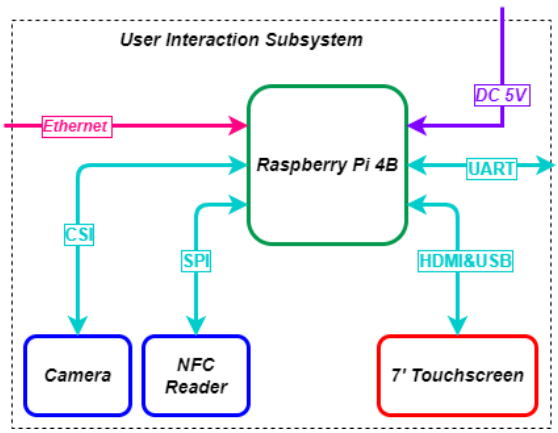

Fig. 2 Block Diagram of KIOSK Terminal Hardware

The client-side software will run on Raspberry Pi 4B which interacts with the user and controls the mechanical subsystem. The hierarchy of the client-side software is shown in Fig. 3.

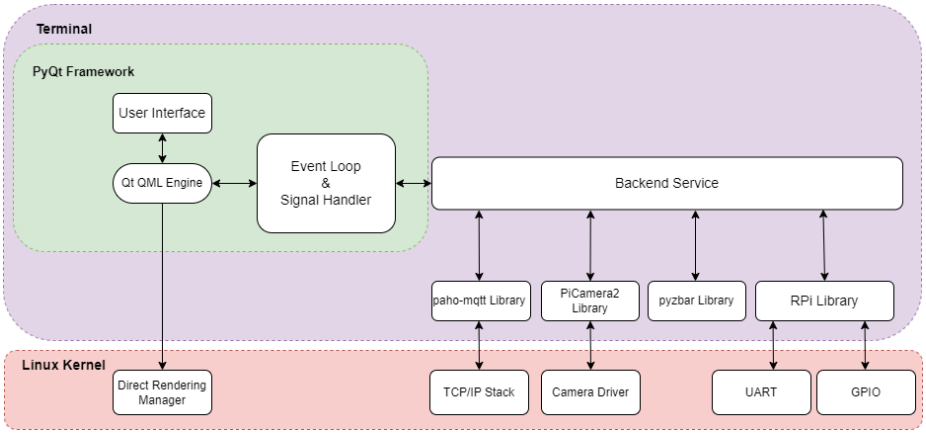

Fig. 3 Block Diagram of Client-side Software

The design of the KIOSK terminal emphasizes human-computer interaction and robustness of the automated card dispensing process, as well as the secure processing of the card information.

We decided to use Raspberry Pi 4B as the main platform for terminal software development. There are other alternatives such as LubanCat, Orange Pi and Jetson Nano. We chose Raspberry Pi 4B for its relatively low price and large group of users. Its broad community largely facilitated the development process. Domestic brands such as LubanCat and Orange Pi are more economical options but there are issues with hardware compatibility and supportive software utilities.

In terms of the KIOSK software design, we considered a range of GUI libraries and frameworks. We've considered the following options: Flutter, Qt Desktop, Qt EGLFS, and PyQt. We initially considered allowing Raspberry Pi to boot into a desktop environment and then start software from there. But the approach risks malicious user takeover through the OS managed response to user input. We wish to keep the system secure and allow as few user inputs as possible. Hence, we decided to directly manage the frame buffer and user input without an OS managed window manager, such as Wayland or X-11. We explored a range of existing frameworks for this purpose. We initially tried Flutter for Embedded Devices, yet we discovered that the framework was in its early stage and not easily compatible with existing libraries

interacting with cameras. We then tried to develop the software with Qt with EGLFS. EGLFS is a Qt application mode that directly interacts with the Direct Rendering Manager (DRM) embedded in Linux kernel and perfectly matches our demand. However, despite its high performance, developing with C++ also comes with its shortages. Firstly, since Raspberry Pi 4B is equipped with Arm-based processors, a cross-compilation toolchain and remote debugging are necessary. The process was relatively slow, and it takes a lot of effort to achieve successful compilation. Secondly, the memory management in C++ was an error-prone task and it is critical for a 24x7 running software. Hence, we again, migrated our software to PyQt framework which preserves most features of Qt while providing the flexibility and translatability of Python. It turns out that the task is not too heavy, and we didn't sense tangible performance degradation after the migration. We also preserved the user interface design as it is written with QML language and dynamically loaded to the QML engine when the software initializes.

We chose the existing NXP RC522 module as the card reader mainly because of their support for the most popular communication protocols and their reliability. Alternatively, we could design our own reader and integrate it into our PCB, but there are a few issues to consider. The most prominent issue is that the design includes an antenna that is to be embedded in the PCB, and such design requires efforts on EMI analysis and fabrication considerations for the high frequency of the signals, which will be too time-consuming for the purpose of the project.

Initially we planned to install a stand-alone bar code scanner for QR code scan, but later we changed the design as we realized that the camera for facial recognition is capable of the task. In this way, the cost of the project is reduced. We chose to interact with users via a 7' touchscreen as it's highly efficient in terms of human-computer interaction, compared to keypads or mouses.

To control the mechanical system, we used a standalone Raspberry Pi Pico. We've also considered alternatives such as STM32 based microcontrollers. The major reason for the choice is that Pico is economic and supports MicroPython, which is a reduced version of Python and allows easy programming and testing. Also, it uses the same voltage level as the Raspberry Pi 4B for communication, which obviates extra voltage level conversion circuits.

Design Tools & Libraries:

- Visual Studio Code: General purpose IDE for software development
- Qt Design Studio: User interface designer for Qt Quick application.

- Qt Linguists: Multilanguage support for the user interface.
- Altium Designer: Circuit & PCB Design

The libraries included in our software are listed in **Error! Reference source not found.**.

Table 1 List of Libraries Used

Library	License	Description
PyQt5[7]	GPL v3	This framework is used to render user interface and manage the events and signals
paho-mqtt[8]	OSI Approved (EPL-2.0 OR BSD-3-Clause)	The library provided a Python implementation of the MQTT protocol, allows us to communicate with the broker easily.
PiCamera2[9]	BSD-2-Clause	The library interfaces with the new open-source camera stack “libcamera” and provide video feed from the camera
Pyzbar[10]	MIT	The library allows us to detect and decode QR codes in the frame.
RPi.GPIO[11]	MIT	The library provides a simple interface to manipulate the GPIO interface on Raspberry.
Spidev[12]	MIT	The library provides an easy way to interact with SPI devices
mfrc522-python[13]	GPL v3	Based on spidev library, the library provide interface to interact with the RFID module
Pyserial[14]	BSD	The library provides an interface to communicate via UART.
opencv-python[15]	Apache 2.0	The library provides a pretrained model for face detection.

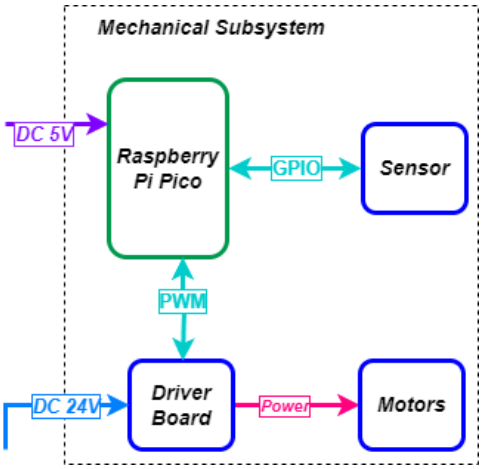

Fig. 4 Block Diagram of Mechanical Subsystem

In terms of the mechanical design, we adopt a vacuum suction system to grab and carry IC cards. Beside the vacuum suction system, the mechanical subsystem includes two linear screw actuators driven by step motors. One linear actuator is placed horizontally while the other is placed vertically so that the mechanical subsystem can move cards in the plane where card storage box and exit of the card dispenser locate. During operation, the mechanical subsystem picks up cards from the card storage box and moves the cards to the exit to send cards to users. When the users return the cards, the mechanical subsystem picks cards at the exit and moves them back to the same card storage box.

There exist other options to dispense cards. The simplest design might be to use an actuator to push out the cards. But this design is not suitable for the situation discussed in this report as the pushing mechanism cannot satisfy the requirement of getting back the returned cards. And the cards must be placed with a distance between adjacent cards to leave space corresponding to the width of the pushing actuator so that only one card will be pushed out at one time, which dramatically reduce the number of cards can be stored in the card dispenser. An alternative method is using a friction wheel to roll the cards out from the storage box to the exit. However, we did not adopt a friction wheel because we were worried that a friction wheel would send out more than one card or cause the card to be stuck in the machine if the size of the opening is not designed properly. In addition, it is difficult to return cards back to the storage box using a friction wheel since the friction wheel must press on the cards to produce the friction force needed to send out cards. Another option is to replace the suction cup with a clamp. But clamps will also suffer from the issue of sending out two or more cards for one user. By contrast, vacuum suction system can relatively easily move cards back to the storage box. Moreover, the vacuum suction system is robust at picking up exactly one card from the storage box, which can be a challenging task for other card dispensing options, including using a friction wheel or using a clamp to pick up cards.

To ensure that the suction cup can provide sufficient pressure to pick up an IC card, the following equation can be used.

$$F = \frac{\pi d^2}{4} P$$

where F is the suction force, d is the diameter of the suction cup, P is the vacuum pressure provided by the pump and π is circumference ratio.

Design Tools:

- Autodesk Fusion 360: CAD modeling

2.1.2 Server-side Software – Authentication Part

The design of the authentication system was guided by two main objectives: securing the transmission of information and verifying user identity efficiently and securely. In considering the architectural framework, we explored several alternatives:

1. Information propagation - *HTTP vs. HTTPS*: Initially, HTTP was considered for simplicity; however, the need for secure transmission made HTTPS the preferable choice. HTTPS provides an encrypted channel, thus enhancing security against interception and unauthorized access. The way we authenticate HTTPS data segments is using certificate signed by CA(Certificate Authority).
2. User Identity Verification - *AD vs. Custom Database*: We had the option to develop a custom database for user credentials or utilize an existing system. A custom database would have required users to register and maintain another set of credentials, but this requires user to register before they use the login functionality. Instead, we opted to leverage the existing Windows Active Directory (AD) system. This choice was driven by AD's widespread use on campus, because AD is already in use across campus services, providing a consistent user experience. And AD has a robust security model that has been battle-tested over the years. We can seamlessly integrate with Microsoft Azure Active Directory what our campus is currently using after we tested our application.
3. Request Verification - *Token-based vs. PIN-based*: PIN-based is an approach where each user is assigned a personal identification number (PIN) that they use to authenticate requests. This method is commonly used in banking and access control systems. However, it requires users to remember an additional set of credentials and is susceptible to security risks if the PIN is compromised. We selected a token-based verification system for its simplicity and effectiveness. This method integrates seamlessly with our existing infrastructure and leverages the secure environment provided by HTTPS, ensuring that tokens are transmitted securely. Our token will be expired in 10 minutes so the risks of PIN compromission is also lowered.

Design Tools and Equations:

- Windows Cloud Server: We used a Windows Cloud Server to boot up AD.
- LDAP Queries: To interact with AD, Lightweight Directory Access Protocol (LDAP) queries were used. These queries allow for efficient retrieval of user data from the AD servers.

- SSL/TLS Protocols: For securing data transmissions, SSL/TLS protocols were employed to establish encrypted links between the client and the server.

2.1.3 Server-side Software – Backend Server and Web Server Part

In the development of the backend server for our project, the selection of server architecture was crucial to accommodate high traffic and maintain efficient performance. Our primary objectives were to ensure high concurrency, efficient resource utilization, and scalability. We considered several architectural frameworks and technologies before finalizing our choices.

1. **Web Server - Nginx vs. Apache**: We initially evaluated traditional web servers like Apache; however, Nginx emerged as the superior option due to its ability to handle significantly higher volumes of concurrent connections, which is pivotal during peak periods such as the start of a new semester or exam times. Nginx's event-driven model allows it to process many requests concurrently, making it highly efficient in environments requiring quick response times with minimal memory usage. This selection was supported by numerous benchmarks showing that Nginx could handle approximately four times more concurrent connections than Apache while using fewer resources.
2. **Database System Comparison – MySQL vs MongoDB**: For the database management system of our project, the decision to use MySQL over MongoDB was guided primarily by ease of use and suitability to our specific needs. Here's why MySQL was favored:
    - SQL vs NoSQL: MySQL, a relational database management system (RDBMS), uses structured query language (SQL) which is beneficial for our application that requires complex queries and precise data integrity. MySQL's ability to handle structured data and support complex transactions is crucial for the operational demands of our server.
    - Ease of Use: MySQL is generally considered easier to use for our requirements, especially when dealing with relational data structures. Its widespread adoption makes it easier to find resources and community support.
    - Schema Management: The structured schema in MySQL is advantageous for our application, which depends on well-defined data formats and relationships. This ensures data consistency and integrity across different parts of our application.
3. Web Server Framework – Python Flask & Gunicorn v.s. React

In the web server stage, the choice of the Flask framework over React was driven by its compatibility with our system requirements and ease of integration with other features such as face recognition technologies: Python Flask: Chosen for its flexibility and ease of use, Flask supports rapid development, which is crucial in a fast-paced project environment. It allows for efficient handling of dynamic content requests, such as user authentication and QR code generation.

- Integration Flexibility: Flask provides more flexibility in integrating various backend functionalities, which is crucial for incorporating advanced features like face recognition into our system seamlessly.
- Ease of Use with Python: Flask, being a Python framework, naturally integrates with the Python programming environment. This integration is beneficial because Python offers extensive libraries and tools that simplify the implementation of additional functionalities, such as data processing and machine learning, which are integral to our project.
- Simplicity and Rapid Development: Flask is designed to be simple and easy to use, enabling quick development and deployment of web applications. This is particularly useful in a project with tight deadlines and the need for frequent updates and iterations.

These design choices form the foundation of our backend server architecture, ensuring it is robust, scalable, and capable of handling our specific needs efficiently.

Design Tools and Equations:

- Nginx: We utilized specific Nginx configurations to optimize handling of concurrent connections and resource allocation. These configurations help in tuning the server according to the expected load and traffic behavior.
- Docker: Used to boot up Nginx and MySQL in a more lightweight way.
- MySQL: Used to boot up the database and manage the database. Chosen for its robust transactional support and ease of use in managing structured data and complex transactions. MySQL excels in environments where data integrity and precise query handling are necessary.
- Python Flask: Utilized for its simplicity and flexibility, enabling rapid development and easy integration of various functionalities, such as face recognition. Flask's compatibility with Python also facilitates the use of extensive libraries for additional features.

2.1.4 Face Recognition – Local Model or Online Model

We tried two different ways to implement the face recognition function. The first one is to deploy the model in our own server, the other is to use the online model from Hikvision, which is broadly used in the campus. We first tried to deploy the face recognition on our server, but we met two main problems:

a. Since the financial support from residential college is not enough, we don't have enough money to buy the server with GPU. Without GPU, it's hard to accelerate the machine learning model. In our experiment, mapping a face picture in a dataset of 10 images takes 49.8 seconds with the model from https://github.com/ageitgey/face_recognition.
b. Another problem is dataset collection. If we deploy the model in our own server, we need to collect the face images of all students, which would cause two problems: 1. Collecting the images from all students needs many human resources. 2. We need to guarantee the data security of face dataset.

Because of the above problems, we begin to consider the second method: use the online model from Hikvision. This online model is broadly used in several buildings in campus: Library, Residential College, Campus Gate, etc. The average latency of detecting a face image using the Hikvision online model is 0.82 seconds, which is acceptable. Additionally, the server from Hikvision has the face datasets from students, so we don't have to collect the images ourselves. We use the Hmac-SHA256 algorithm to secure the communication between the backend server and Hikvision server, so the security of dataset is guaranteed.

2.2 Design Details

2.2.1 KIOSK Terminal – User Interface

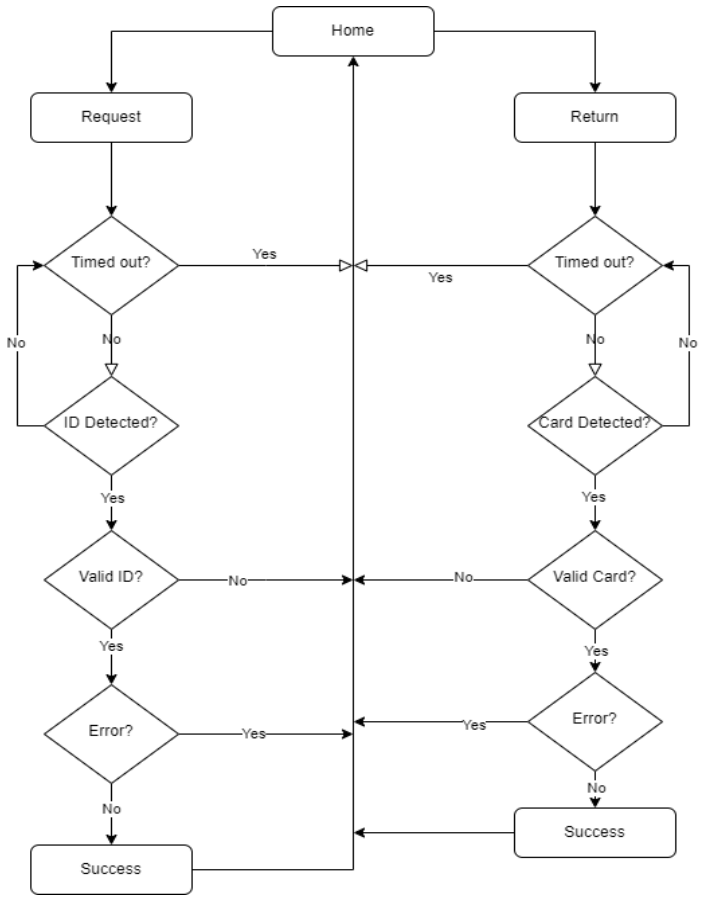

Fig. 5 KIOSK Terminal Software Flow Chart

Component Specifications:

- Support for multiple languages: As described previously, the user interface supports multiple languages, as shown in Fig. 6 below.

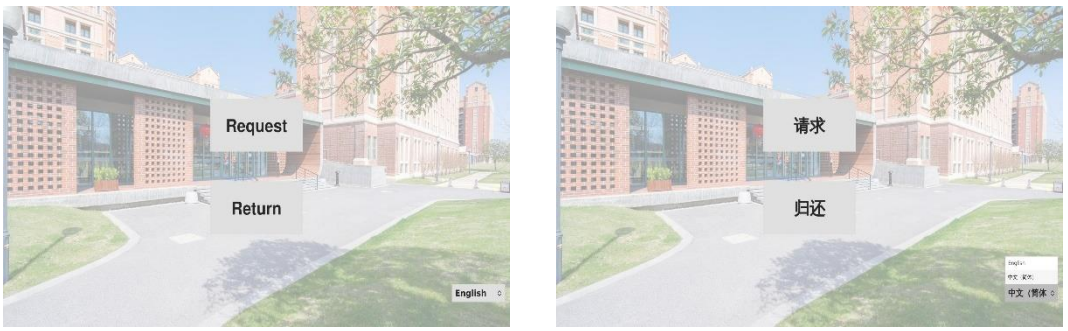

Fig. 6 User Interface

- Facial / Barcode Detection: Once the user pushes the “Request” button, the

camera starts, and a preview of the camera feed is shown on the screen. Our software is responsible for forwarding the video frames from the camera stack to the buffer, while employing OpenCV and pyzbar libraries to detect any faces or bar codes. Once detected, the software will automatically switch to a “In-Process” tab and informs the user of the status. Meanwhile, the software will validate the information with the server and, should the information be valid, the software will instruct the peripherals to issue a new card to the user.

- RFID Tag Processing: The software will decode the keys and write the information to a registered blank card. When the user try to return a card, the reader will also check its validity and reject the card if the card is not valid.
- Mechanical Control: The Raspberry Pi 4B communicates with the microcontroller via UART. The communication is in a request-response manner. Specifically, when Pi asks the mechanical subsystem to move, the microcontroller will immediately acknowledge the request, and later an “in-position” response is returned. Should there be any unexpected mechanical error or a time-out exception, the microcontroller will respond with an error signal, in which case the software should notify the server of the incident.

2.2.2 KIOSK Terminal – Mechanical Subsystem

The vacuum suction system consists of a 5 Volts vacuum pump, a 30-mm diameter suction cup, and a 12 Volts solenoid valve. The vacuum pump provides a 55 KPa vacuum pressure by pumping out air, resulting in a suction force of 38.87 N, which is much larger than the weight of an 8-gram IC card, also ensuring that the card will not fall from the suction cup during motion even though the card might suffer from collision or friction. The solenoid valve is adopted to destroy vacuum situation to speed up card releasing process.

Several sensors are added to the mechanical subsystem to enhance its robustness.

- Tact sensor: used to detect the height of the card stack so that the suction cup can stop at the proper height where the suction cup can grab the card and the suction cup will not run into the card stack.
- SN04-N proximity sensor: Define the origin position of two linear actuators. Based on the origin of the linear actuators, the actual positions of two linear actuators can be determined. By defining the origin position, the motion of two linear actuators can be limited to a certain range, preventing collisions with other parts in the system

- Optoelectronic sensor: Used to detect whether there is a card at the exit. This function is added to ensure that the card has been sent to a proper position when the user returns the card so that the suction cup can get the card back.

2.2.3 Server-side Software – Authentication Part

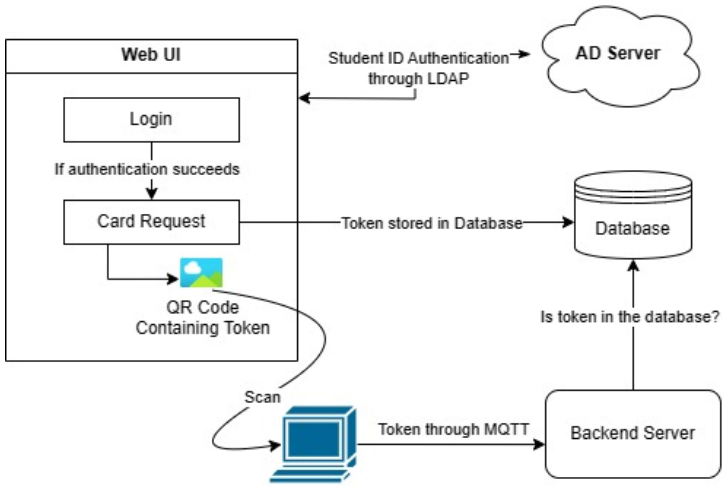

Fig. 7 Authentication Procedure

Component Specifications:

- Database Configuration: In MySQL database, we created a table called “tokens” and an action which will delete all the record generated 10 minutes ago.

Field	Type	Null	Key	Default	Extra
token	varchar(32)	NO	PRI	NULL	
student_id	bigint(20)	YES	UNI	NULL	
created_at	timestamp	NO		CURRENT_TIMESTAMP	
request_id	int(10) unsigned	YES		NULL	

Fig. 8 Database Content

- Security Protocols: The certificate we used is signed by CA.
- AD Server we used: Booted up on 10.105.5.97 named as izju.

2.2.4 Server-side Software – Backend and Web Server Part

Here, we want to mainly discuss the workflow about how to give a card and return a card in our backend server.

The workflow about receiving a request sending a card is shown in Fig. 9 below,

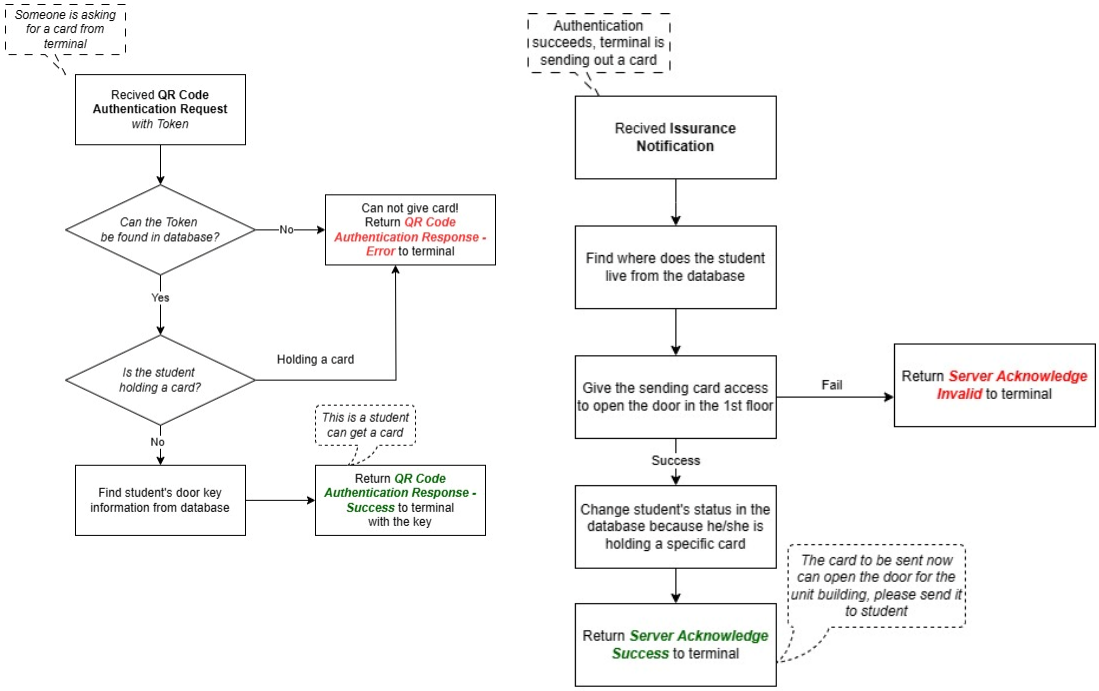

Fig. 9 Workflow about Receiving a Request

2.2.5 Server-side Software – Facial Recognition Part

The facial recognition server is responsible for receiving images of faces from a terminal, analyzing them using a machine learning (ML) model, and comparing them against a predefined facial dataset to identify the student in the image. This system primarily employs advanced ML models such as Deepface, FaceNet, and VGGFace. If the server cannot find a match in the dataset, it immediately sends an error message back to the client. However, if a match is found, the server sends a request with the student's information to the RC server to facilitate the issuance of a temporary card.

The workflow for this facial recognition subsystem is as follows: First, a terminal captures an image of a student using a camera and sends it to the backend server. This server converts the image into binary data and encodes it in Base64 format before transmitting it to the facial recognition server. Upon receiving the data, the facial recognition server decodes it back into an image format. It then uses a computer vision model to compare this image with the stored images in the student database, looking for the highest similarity. The server then sends the student's information along with the computed similarity score back to the backend server. For security purposes, the authentication process is deemed successful only if the similarity score, which ranges from 0 to 100, exceeds a set threshold of 80.

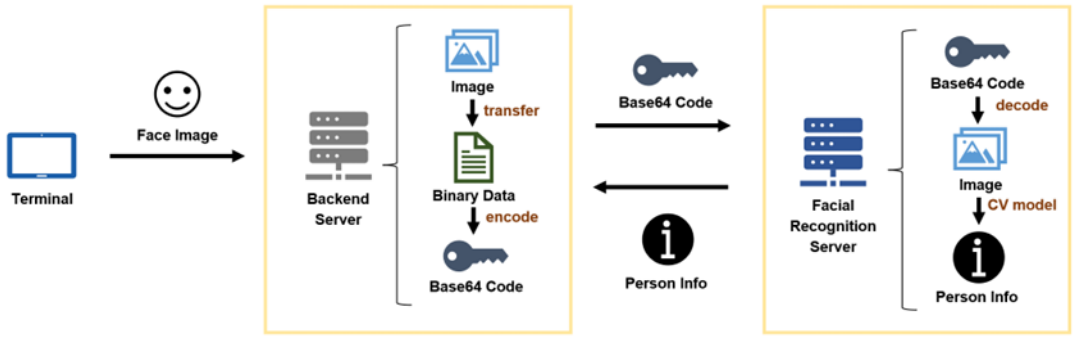

Fig. 10 Workflow for Facial Recognition Authentication

2.2.6 Server-side Software – Access Control Part

The door access system comprises three distinct parts: dorm room access, block access, and residential college door access. Each segment is controlled by different entities, necessitating varied approaches.

For dorm room access, we face a limitation as the company managing this part, along with the workers at the residential college, have declined to share their dataset with us. Therefore, we can only duplicate the door access key from the student ID card into our database system during the student registration process. Team members Zicheng Ma and Xiaoyang Chu are tasked with overseeing this component.

For the block and residential college door access, we are supported by the Hikvision platform, with the residential college staff granting us the necessary permissions to manage access control. The access control subsystem operates as follows:

Initially, we must authenticate using the Hikvision platform, which employs an Access Key/Secret Key (AK/SK) system for client identity verification. Upon user registration, the platform assigns an AppKey and an AppSecret to each user. The authentication process involves generating a signature that incorporates the Http Method, headers, and Url (including path, query, and bodyForm). This signature, using the AppSecret as a cryptographic key, is hashed with the Hmac-SHA256 algorithm to create a message digest, which is then encoded in BASE64 format using UTF-8. The resultant signature is included in the HTTPS request header, enabling the platform to verify the client's identity using the AppKey.

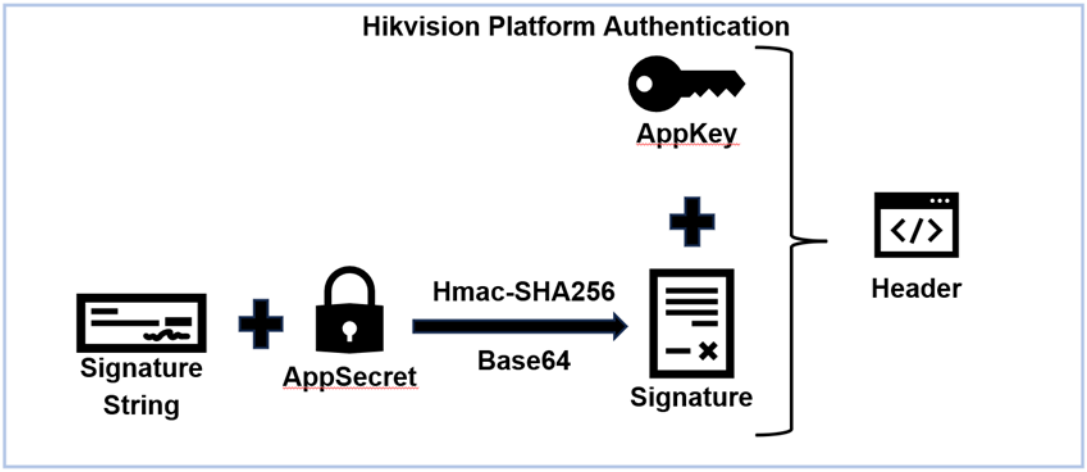

Fig. 11 Workflow for Facial Recognition Authentication

Post-identity verification, we utilize the student's name and ID to fetch their dorm room number from our SQL database. This allows us to ascertain the corresponding block ID and residential college ID. We then request the Hikvision platform to assign a personID (distinct from the student ID) for the student, using their name and student ID. We also retrieve the ID for the relevant devices (door locks) using their names. With both the personID and device ID, we set up the access permissions for the student by uploading the student ID, card ID, and device ID to the platform, thereby enabling access. The same process is employed to revoke access once the student returns their card.

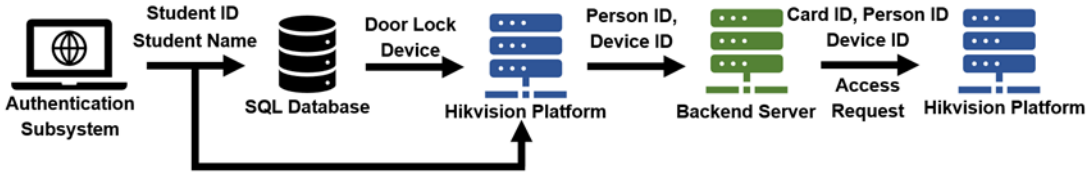

Fig. 12 Workflow for Facial Recognition Authentication

3 Verification

3.1 QR Code Generation Test

Objective: Ensure that the QR codes generated by the system are accurate and reliable for authentication purposes. Besides, the QR code really carries the 32 bytes token we want.

Procedure: QR codes were dynamically generated on a web server accessible via

ZJUWLAN at port 8002. The generated codes were then tested at kiosk terminals to authenticate user access, assessing the correctness of the encoded information and the scanning system's reliability. The robustness of QR codes was also evaluated under varying physical conditions, including different lighting levels and angles.

Outcome: All QR codes were successfully recognized and authenticated by the kiosk scanners without any discrepancies, confirming the effectiveness of the encoding and decoding processes. The tests also verified the resilience of QR codes, maintaining readability under diverse environmental conditions.

3.2 Request Card and Return Card Test

Objective: To validate the functionality of the card request and return system, using QR codes for card issuance and authentication, and ensuring backend and terminal communication for successful key writing and door access control.

Procedure:

Card Issuance: QR codes generated by the backend system were scanned at the terminal using an integrated camera setup. The terminal established communication with the backend to request card issuance.

Key Writing: Upon successful QR code verification, a unique key was written to the card, which was then used to unlock a test door lock, demonstrating the operational effectiveness of the card in a real-world scenario.

Card Return: After use, the card was returned to the terminal, which triggered the backend system to rewrite the database entries associated with the card, ensuring that all information was updated and consistent with system requirements.

Outcome:

Issuance and Authentication: The QR code scanning and verification process was flawless, with no errors in recognition or decoding. The backend efficiently handled the card issuance request and communicated the necessary data to the terminal.

Key Functionality: The key written to the card successfully activated the door

lock, confirming the system's ability to handle secure transactions and control access effectively.

3.3 Facial Recognition Test

We test the similarity scores and the latency with four resolution sizes and 20 images. For the image dataset, we include all angles of the face photos, including left face, right face and the middle face.

Based on the latency and similarity score result shown below, we finally decided to resize the input image into 384 x 512 pixels, which can maximize the similarity score and minimize the latency as much as possible.

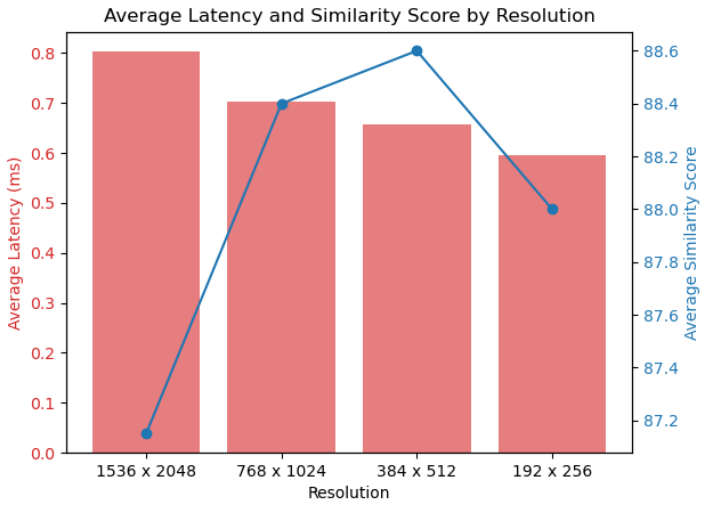

Fig. 13 Average Latency and Similarity Score by Resolution

3.4 Mechanical Test

To test the stability of the mechanical subsystem, the mechanical subsystem is run to simulate the scenario of borrowing and returning cards for 20 times. And it turns out that the mechanical subsystem operates properly every time without human intervention. Although the number of tests is not sufficient to prove that the failure rate of the mechanical subsystem is below 0.2%, it still provides an evidence about the mechanical subsystem's robustness.

4. Cost Analysis

Table 2 Human Labor Cost Analysis

Labor	Quant.	Amount (CNY)
Develop the face recognition subsystem	50 h	2000
KIOSK Software Development	75 h	3000
Build a QR code web server	40h	1200
Build database and import RC data	30h	1000
Build mechanical subsystem	40h	1200

Table 3 Parts Cost Analysis

Parts	Quant.	Amount (CNY)
Raspberry Pi 4B Development Kit	1	775.90
RFID Analyzing Tool	1	345.92
Vacuum pump and suction cup	1	30.39
XY actuator	1	480
Raspberry Pi Pico Microcontroller	1	39
Camera Module	1	29.9
Miscellaneous Cables & Components		276.43

5 Conclusion

5.1 Accomplishments

We finish the implementation and verification of the backend subsystem, facial recognition subsystem, access control subsystem, Kiosk Terminal Subsystem, and

Mechanical Subsystem.

5.2 Uncertainties

Since some of the students don’t register their face images in the Hikvision system, we are not sure how many students can pass the face recognition authentication.

5.3 Future Work

For now, the backend software and the mechanical part haven’t been assembled. Next week we will finish the assembly.

5.4 Ethical Consideration

5.4.1 Safety of Authentication and Access control System

The safety concerns for the software system primarily relate to data protection, system reliability, and user privacy. Ensuring the safety of these aspects is critical, as breaches could lead to identity theft, unauthorized access, or service disruptions.

- User Privacy and Encryption: The system must maintain user privacy by ensuring that personal data is not exposed to unauthorized entities. This requires all data stored and transmitted to be encrypted using industry-standard cryptographic protocols to prevent unauthorized access. Therefore, we plan to adopt SHA-256, which is a relatively secure, difficult, and costly encryption method in the current industry.
- System Reliability: The RC Server must offer high availability and fault tolerance to avoid service interruptions, which could lead to safety issues in systems that rely on constant connectivity for critical functions. Although the traffic and demand on campus may not be large, the stability and load capacity of the server are also to be considered.

5.4.2 Reliability of Facial Recognition

It is guaranteed that the face dataset of students is safely stored in the facial recognition system. We will also make sure that when the camera take a picture of student, the picture will only be used for facial recognition, but not for other purpose.

5.4.3 Safety Concerns of Mechanical Systems

- Since mechanical system uses vacuum suction cup to grab IC cards, which involves the production and use of compressed air flow. It should be checked that the compressed air is processed properly so there will be no leak or

explosions.

- It should be ensured that the mechanical subsystem will not hurt the users' hands when the user puts his or her hand at the exit of the card dispenser to pick up the card or return the card. When a user picks up the card, the user opens the door at the exit of the card dispenser to get the card. The suction cup and the linear actuators cannot move when the user opens the door.
- The mechanical subsystem should have the ability to recycle IC cards at the exit back if the users do not pick up the cards in 5 minutes after the cards are sent out. That function can prevent the risk that someone else taking away the cards, in order "to protect safety of others" [5].
- The mechanical subsystem should wear the card as little as possible to " comply with ethical design and sustainable development practices" [5] and reduce the frequency of replacement of cards. Suction cup is adopted since it will do little wear to the cards.

Reference

[1] “MQTT - the standard for IoT messaging.” https://mqtt.org/

[2] Taigman, Y., Yang, M., Ranzato, M. A., & Wolf, L. (2014). DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[3] Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[4] Parkhi, O. M., Vedaldi, A., & Zisserman, A. (2015). Deep Face Recognition. In British Machine Vision Conference (BMVC).

[5] IEEE, “IEEE Code of Ethics,” ieee.org, Jun. 2020. https://www.ieee.org/about/corporate/governance/p7-8.html

[6] National Institute of Standards and Technology, “Secure Hash Standard (SHS),” U.S. Department of Commerce, Aug. 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[7] Riverbank Computing Limited, The Qt Company, "PyQt5 Reference Guide," [Online]. Available: https://www.riverbankcomputing.com/static/Docs/PyQt5/. [Accessed 15 April 2024].

[8] Eclipse Foundation AISBL, "Paho," [Online]. Available: https://eclipse.dev/paho/. [Accessed 15 April 2024].

[9] Raspberry Pi Ltd, "The Picamera2 Library," 27 November 2023. [Online]. Available: https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf. [Accessed 15 April 2024].

[10] D. Ferens, Alex, I. Bento and @jaant, "pyzbar," [Online]. Available: https://github.com/NaturalHistoryMuseum/pyzbar/. [Accessed 15 April 2024].

[11] B. Croston, "raspberry-gpio-python," [Online]. Available: http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/. [Accessed 15 April 2024].

[12] S. Caudle, "py-spidev," [Online]. Available: https://github.com/doceme/py-spidev. [Accessed 15 April 2024].

[13] Aditya and N. Downing, "MFRC522-python," [Online]. Available: https://github.com/1AdityaX/mfrc522-python. [Accessed 15 April 2024].

[14] C. Liechti, "pySerial," [Online]. Available: https://pyserial.readthedocs.io/en/latest/pyserial.html. [Accessed 15 April 2024].

[15] OpenCV Team, " OpenCV Team," [Online]. Available: https://github.com/opencv/opencv-python. [Accessed 10 May 2024].

Appendix

Appendix A. Circuit Diagram for Mechanical Control

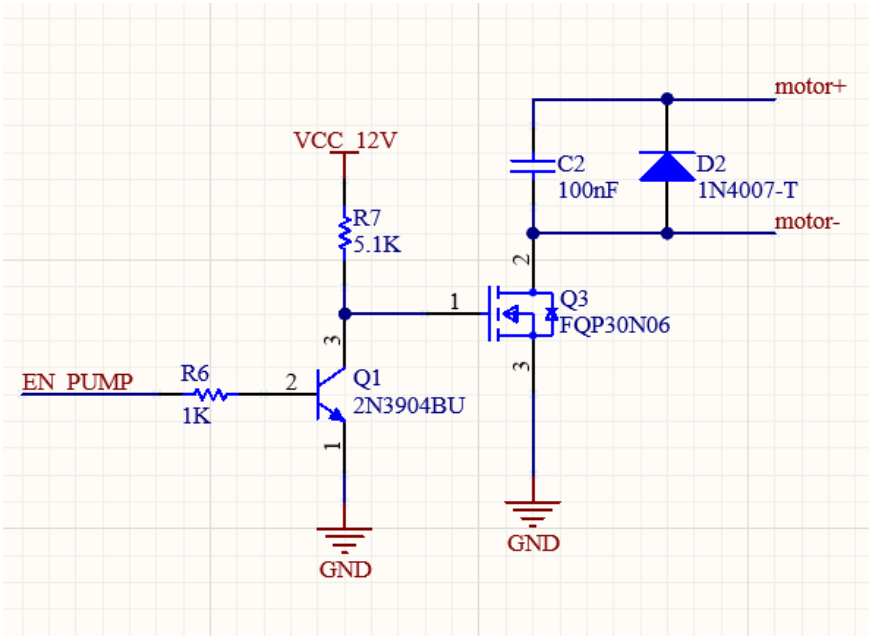

Fig. 14 Circuit Diagram for Vacuum Pump Control

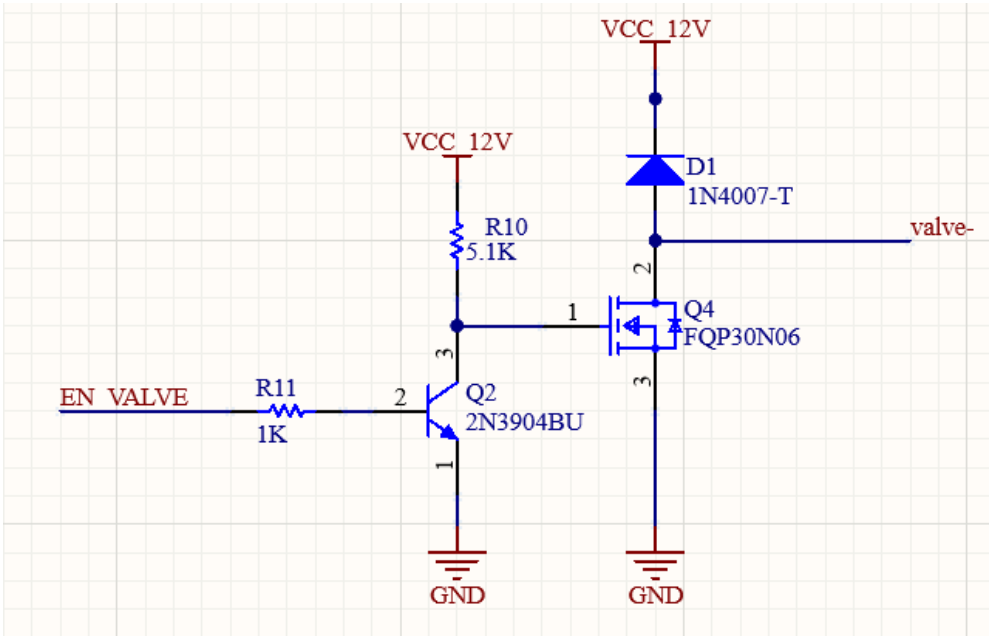

Fig. 15 Circuit Diagram for Solenoid Valve Control

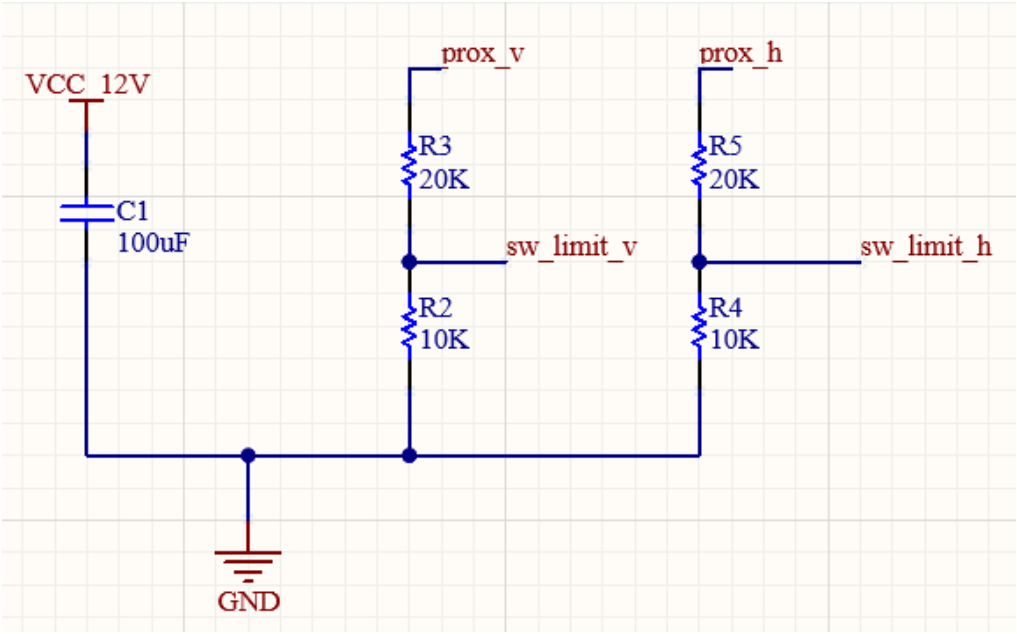

Fig. 16 Circuit Diagram for Proximity Sensor Signal Processing

Appendix B. Similarity Score and Latency of Facial Recognition

Table 4 Similarity scores with different image resolution

Image Index / Resolution	1536 x 2048	768 x 1024	384 x 512	192 x 256
1	89	88	89	86
2	89	88	90	88
3	90	92	91	89
4	88	87	90	87
5	84	87	87	86
6	87	86	89	91
7	87	91	91	90
8	86	89	90	91
9	85	86	86	88
10	94	85	86	86
11	88	88	90	90
12	89	89	90	87
13	83	89	86	86
14	85	87	86	87
15	88	89	87	85
16	88	89	88	88
17	89	91	90	89
18	86	88	88	87
19	82	90	89	91
20	86	89	89	88

Table 5 Latency of online model with different image resolution

Image Index/Resolution	1536 x 2048	768 x 1024	384 x 512	192 x 256
1	0.738295794	0.628719091	0.651124716	0.640042067
2	0.758849859	0.642990112	0.58182621	0.545042753
3	0.744408131	0.633938551	0.583513021	0.558713436
4	0.751494408	0.581557751	0.691009521	0.584634542
5	0.854304314	0.726917028	0.721786499	0.612939596
6	0.83710289	0.766220093	0.70773983	0.555555582
7	0.761623383	0.744357347	0.682974577	0.598637581
8	0.712453604	0.672476292	0.618635416	0.578127861
9	0.798819304	0.730727196	0.705238342	0.652128696
10	0.815337896	0.669465542	0.653531075	0.609490156
11	0.785992622	0.710080385	0.683963537	0.603325844
12	0.826137781	0.816992521	0.665312529	0.599835396
13	0.755683184	0.722836018	0.642542124	0.606658936
14	0.749848127	0.730558157	0.657264948	0.635754347
15	1.25428915	0.696682215	0.637785673	0.680992126
16	0.772652388	0.728740931	0.672860622	0.583869934
17	0.804905891	0.740184069	0.644750595	0.536188602
18	0.75211525	0.670586586	0.635467529	0.550262451
19	0.801511526	0.684829712	0.679459095	0.591224909
20	0.77336359	0.738835573	0.643973112	0.595976114