PROJECT REPORT

Project Semester 4

Face Recognition with Micro:bit Robot

Student Name: Xuanru Guo

NUIST Number: 202283890028

SETU Number: W20109677

Lab ID: RobotProject1

Email: xuanru.guo.echo@gmail.com

Github URL: https://github.com/Ataraxiaii/ProjectRobot1_2025.git

Video URL: Bilibili Video URL

June 29, 2025

Contents

1 The Agile Process **2**

1.1 Agile Framework with User Story 2

1.2 Scrum & Sprint Summary 3

1.3 Burndown Table & Chart 5

2 Project Introduction **8**

2.1 Project Overview 8

2.2 Project Objective 8

2.3 Project Management 8

2.4 Project Methodology 9

3 Hardware Connection **11**

3.1 Overview of Hardware Component 11

3.2 Micro:bit V2 Module 11

3.3 K210 Visual Identification Module 12

3.4 Component Connection 13

4 Software Development Methodology **15**

4.1 Implementation Overview 15

4.2 Display Images on LED Matrix 15

4.3 Recognize Faces with Number 16

4.3.1 Micro:bit V2 Part Code 17

4.3.2 K210 Visual Module Part Code 18

4.3.3 System Testing and Results 20

4.4 Hash Recognized Face Data 22

A Commit History **25**

B Source Code **27**

List of Figures

1.1 Sprint 1 Burndown Table 5
1.2 Sprint 1 Burndown Chart 6
1.3 Sprint 2 Burndown Table 6
1.4 Sprint 2 Burndown Chart 7

2.1 Github Repository Management 9
2.2 MakeCode Platform 10
2.3 Mu Editor 10

3.1 Hardware Components 11
3.2 Micro:bit V2 Module 12
3.3 K210 Visual Module 13
3.4 First Step Building Car 13
3.5 Second Step Building Car 13
3.6 Final Assemble Step 14

4.1 Display Images on LED Matrix 16
4.2 Exercise 1 Git Push 16
4.3 MakeCode Block Code 18
4.4 Scroll Display READY 20
4.5 Camera Setup 20
4.6 No Human Face 21
4.7 Undefined Human Face 21
4.8 Registered Human Face 22
4.9 Exercise 2 Git Push 22
4.10 Exercise 3 Git Push 23
4.11 Commit Merge Code 24

1 The Agile Process

This project follows the Scrum Agile framework, employing iterative development as its core methodology. The design and implementation of the robot face recognition project were divided into three sprint cycles. Through the creation of user stories, we ensured responsiveness to changes and maintained continuous progress. Task completion was effectively managed using burndown charts, while GitHub was utilized for version control and collaboration.

1.1 Agile Framework with User Story

During the initial planning phase, I first divide the project into three short sprint cycles. By defining project user stories, I create an actionable task list to guide development from both technical implementation and user experience perspectives. As shown in the following Sprint 1 summary, this phase includes three core user stories and four specific development tasks.

Sprint 1 Duration: 2 Days

User Stories::

- User Story 1: As a project manager, I want to define clear user stories so that the tasks are well defined.
- User Story 2: As a Scrum master, I want to establish a sprint plan so that progress can be tracked efficiently.
- User Story 3: As a project manager, I want to create burndown chart and table so that I can track the remaining work.

Tasks::

- Task 1: Plan Project Scope and Sprint Timeline
- Task 2: Define Project User Stories
- Task 3: Create Burndown Chart Template
- Task 4: Finalize Agile Process Report

Sprint 2 marks the official transition into the technical implementation phase, where I will primarily focus on assembling the robotic vehicle hardware and implementing core functionalities.

Sprint 2 Duration: 5 Days

User Stories::

- User Story 1: As a developer, I want to be able to build a micro:bit robot so that I have a physical platform for face recognition experiments.
- User Story 2: As a developer, I want to be able to show numbers on micro:bit LED matrix so that the system can provide basic visual feedback to users.
- User Story 3: As a developer, I want to be able to recognize different faces using micro:bit robot.
- User Story 4: As a developer, I want to be able to link different face with different number image so that each user gets personalized visual feedback.
- User Story 5: As a security developer, I want to hash face data using SHA-256 so that biometric information is stored securely.
- User Story 6: As a user, I want to register my face by pressing button so that the binding process is intuitive.
- User Story 7: As a tester, I want the LED to display "X" when recognition fails so that users clearly know when to retry positioning.

Tasks::

- Task 1: Create Github Repository
- Task 2: Build Micro:bit Robot
- Task 3: Display Images on Led Matrix
- Task 4: Recognize Different Faces
- Task 5: Link Faces with Related Numbers
- Task 6: Hash Recognized Faces
- Task 7: Add Error Handling
- Task 8: Finish Burndown Chart
- Task 9: Manage Code Version Control
- Task 10: Compile Final Report
- Task 11: Record Project Demo Video

1.2 Scrum & Sprint Summary

According to Section 1.1, I have designed the tasks to do in this two sprints.

SCRUM (Plan Micro:bit Robot Project) Duration: 7 Days

SPRINT 1 - Duration = 2 Days:

- Task 1: Plan Project Scope and Sprint Timeline
- Task 2: Define Project User Stories
- Task 3: Create Burndown Chart Template
- Task 4: Finalize Agile Process Report

SPRINT 2 - Duration = 5 Days:

- Task 1: Create Github Repository
- Task 2: Build Micro:bit Robot
- Task 3: Display Images on Led Matrix
- Task 4: Recognize Different Faces
- Task 5: Link Faces with Related Numbers
- Task 6: Hash Recognized Faces
- Task 7: Add Error Handling
- Task 8: Finish Burndown Chart
- Task 9: Manage Code Version Control
- Task 10: Compile Final Report
- Task 11: Record Project Demo Video

The project follows a structured Scrum framework divided into two key sprints totaling 7 days. Sprint 1 (2 days) establishes the Agile foundation, focusing on project planning and documentation. During this initial phase, critical groundwork is laid including defining the project scope through user stories, creating a sprint timeline with clear milestones, developing a burndown chart template for progress tracking, and finalizing the Agile methodology report. This preparatory work ensures all subsequent technical implementation has clear objectives and measurable criteria.

Sprint 2 (5 days) transitions into technical execution, where the team builds the physical Micro:bit robot while implementing its core functionalities. The sprint begins with repository setup and hardware assembly, then progresses through programming the LED matrix display, implementing face recognition algorithms, and establishing face-number associations. Security features like face data hashing are integrated, followed by robust error handling implementation. The sprint concludes with project management activities including finalizing the burndown chart, managing code versions through Git, compiling the comprehensive final report, and recording a demonstration video that showcases all functional requirements. This

phased approach ensures each technical component is properly tested before integration while maintaining Agile principles of iterative development.

1.3 Burndown Table & Chart

A burndown chart is a graph that represents the work left to do versus the time it takes to complete it. It can be especially useful for teams working in sprints, as it can effectively show whether deadlines are able to be met along the way.

This burn-down chart clearly documents the progress of the 2-day Sprint #1, presenting in detail the four tasks (including planning project scope, defining user stories, creating a burn-down chart template, and finalizing the Agile report) along with their completion status on June 21st and 22nd. Each task is labeled with initial time estimates and actual completion status, while the bottom section summarizes the remaining workload and compares it with the ideal trend. The use of different colors effectively distinguishes between tasks and progress data, making the well-structured table visually clear and intuitive.

Sprint #1 Burndown Chart				
Duration: 2 days				
Task ID	User Story	Initial Estimate(Hours)	21-Jun	22-Jun
		Day 0	Day 1	Day 2
1	Plan Project Scope and Sprint Timeline	0.5	0.5	0
2	Define Project User Stories	0.5	0.5	0
3	Create Burndown Chart Template	0.5	0	0.5
4	Finalize Agile Process Report	1	0	1
Remaining Effort		2.5	1.5	0
Ideal Trend		2.5	1.25	0

Figure 1.1. Sprint 1 Burndown Table

Figure 1.2 shows the ideal trend and remaining effort of completing the tasks in sprint 1. The burn-down chart for Sprint 1 clearly illustrates the two-day progress tracking, showing both the planned (orange dashed line) and actual (blue solid line) remaining effort starting from approximately 2.5 hours and gradually decreasing to zero by the end of the sprint period. The visual comparison between these two trend lines demonstrates how the team's work completion pace aligned with initial estimates throughout this short development cycle.

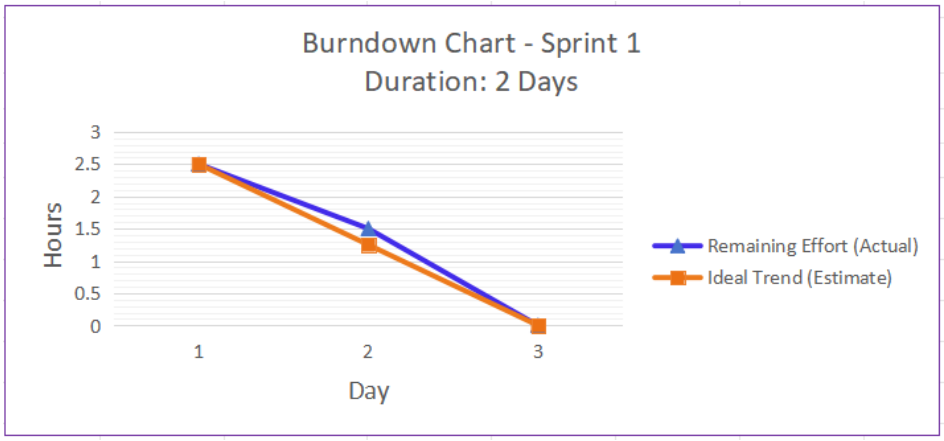

Figure 1.2. Sprint 1 Burndown Chart

The Sprint #2 burn-down chart presents a comprehensive 5-day progress overview (June 23-27) tracking 11 development tasks including GitHub repository setup, Micro:bit robot construction, LED matrix display implementation, face recognition system development, and final reporting activities. The white-background table displays each task's initial time estimates (ranging from 0.5 to 3.5 hours) alongside daily completion status, with color-coded rows differentiating between technical implementation tasks (like face hashing and error handling) and documentation work (final report and demo video). The bottom section captures the cumulative remaining effort starting from 14 total hours, showing gradual reduction across the sprint period while comparing actual progress against an ideal completion trajectory.

Sprint #2 Burndown Chart							
Duration: 5 days							
Task ID	User Story	Initial Estimate(Hours)	23-Jun	24-Jun	25-Jun	26-Jun	27-Jun
		Day 0	Day 1	Day 2	Day 3	Day 4	Day 5
1	Create Github Repository	0.5	0.5	0	0	0	0
2	Build Micro:bit Robot	0.5	0.5	0	0	0	0
3	Display Images on Led Matrix	1.5	1.5	0	0	0	0
4	Recognize Different Faces	1	0	1	0	0	0
5	Link Faces with Related Numbers	1	0	1	0	0	0
6	Hash Recognized Faces	2	0	0.5	1.5	0	0
7	Add Error Handling	1	0	0	1	0	0
8	Finish Burndown Chart	0.5	0	0	0	0.5	0
9	Manage Code Version Control	0.5	0	0	0	0.5	0
10	Compile Final Report	3.5	0	0	0	1	2.5
11	Record Project Demo Video	2	0	0	0	0	2
Remaining Effort		14	11.5	9	6.5	4.5	0
Ideal Trend		14	11.2	8.4	5.6	2.8	0

Figure 1.3. Sprint 2 Burndown Table

Figure 1.3 shows the ideal trend and remaining effort of completing the tasks. The burn-down chart for Sprint 2 tracks progress across five days, with the blue solid line (Actual Remaining Effort) starting at 14 hours and gradually declining, while the orange dashed line (Ideal Trend)

shows the projected linear decrease. The actual progress closely follows but slightly lags behind the ideal trajectory, particularly noticeable between Days 3 and 4 where the pace accelerates, ultimately converging near zero by the sprint's end, demonstrating effective planning execution.

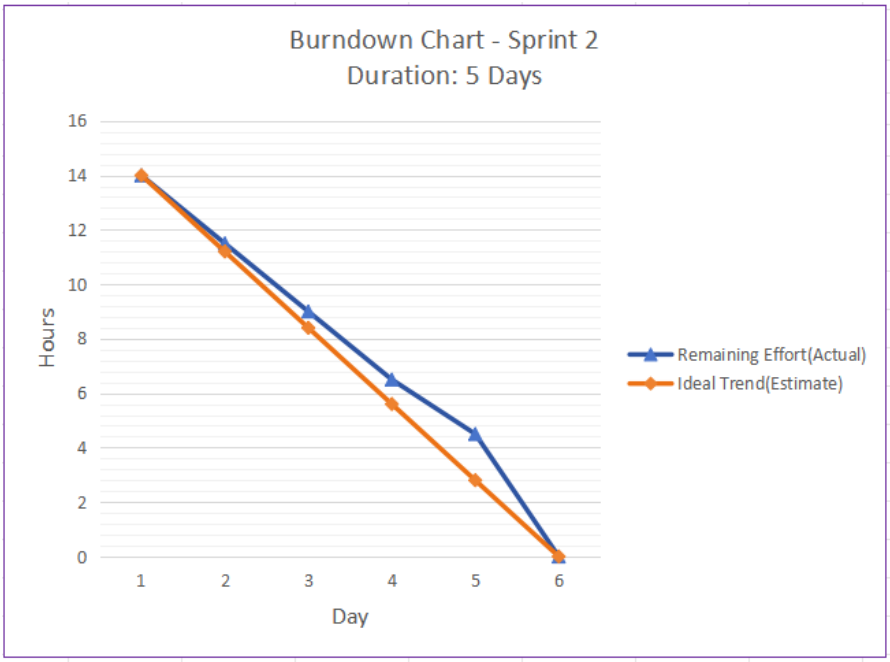

Figure 1.4. Sprint 2 Burndown Chart

The analysis of burn-down charts across two sprint cycles validates the effective application of Agile methodology in this project, demonstrating both reasonable task breakdown and the team's strong progress control.

2 Project Introduction

2.1 Project Overview

The RobotProject1 innovatively integrates hardware development with Agile methodology, requiring the development of a Micro:bit robot with facial recognition capabilities through iterative sprints. This project combines microcontroller programming (Micro:bit components), Python scripting, and the Scrum framework, requiring us to implement core functionalities ranging from LED matrix displays to facial data encryption while maintaining rigorous version control via GitHub. As a hands-on Agile learning project, it bridges theoretical concepts with engineering implementation, ultimately delivering comprehensive outcomes including a functional robot prototype and demonstration video.

2.2 Project Objective

This project aims to develop a Micro:bit-based robot capable of face recognition through three core functions - all implemented using Python within an Agile development framework:

- Display numbers on an LED matrix;
- Recognize and matching faces to specific numbers;
- Secure facial data through hashing;

2.3 Project Management

Figure 2.1 shows the public GitHub repository **ProjectRobot1_2025** created for managing the Face Recognition System project. The repository demonstrates clear management of structured Agile project practices and code program version. The repository includes multiple Python scripts and trained YOLO models, each reflecting different stages of the system's evolution:

- Exercise 1 document for displaying images on led matrix;

- Exercise 2 document provides hex file for microbit and python script to recognize face through K210;
- Exercise 3 document offers python script to hash data on the basis of Exercise 2;
- Project Images document shows pictures of hardware components.

The README.md file at the root of the repository provides a well-documented project description, including its introduction, hardware requirements, steps to run the script, and software requirements.

Figure 2.1. Github Repository Management

2.4 Project Methodology

In this project, I designed and implemented a face recognition system with hashed data encryption. After conducting preliminary background research, I adopted Agile development methodology for system planning, including defining user stories and preparing a burndown chart. Subsequently, I completed hardware implementation tasks such as device assembly and visual recognition module calibration.

For software development, I programmed in Python through Microsoft MakeCode for micro:bit's

official website (Figure 2.2) - a graphical programming platform developed by Microsoft to support BBC micro:bit programming. After coding, I flashed the program to the micro:bit via USB. Alternatively using Mu Editor (Figure 2.3) for Python programming, I primarily implemented face recognition and data encryption functions on the K210 vision module. Ultimately, I successfully achieved face recognition and lightweight hash encryption on the K210, with data transmission to the micro:bit to display corresponding face IDs on its LED matrix and trigger music playback.

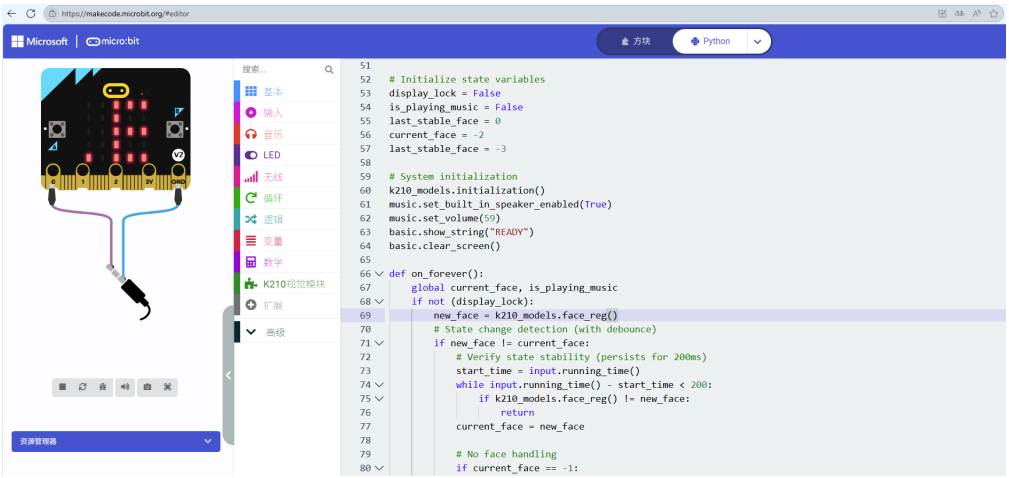

Figure 2.2. MakeCode Platform

Figure 2.3. Mu Editor

A major technical challenge emerged during development: Due to the K210 module's limited memory, pursuing highly secure hash encryption caused memory overflow, resulting in camera failure and image loss. Therefore, for our hash algorithm implementation, we had to balance security requirements with system availability and performance considerations.

3 Hardware Connection

3.1 Overview of Hardware Component

The hardware system of this project is built around the micro:bit V2 development board as the core controller, integrated with a K210 vision module equipped with a dual-core RISC-V processor. Stable communication between the micro:bit and K210 module is achieved through I2C interface, while the onboard 5x5 LED matrix displays real-time face recognition results. The system retains GPIO pins for external sensor expansion and utilizes lithium battery power supply, delivering stable 5V output through the micro:bit's power interface to ensure reliable operation of the vision processing unit.

Figure 3.1. Hardware Components

3.2 Micro:bit V2 Module

The micro:bit serves as the central control unit in this face recognition system, leveraging its compact size and versatile I/O capabilities to bridge hardware components. This credit-card-

sized development board processes signals from the K210 vision module through its I2C interface while driving the 5Œ5 LED matrix to display recognition results. Its GPIO pins facilitate real-time data exchange with external sensors, and the built-in accelerometer enables potential motion-based interactions.

Powered by a lithium battery, the micro:bit's energy-efficient design ensures prolonged operation while maintaining stable 5V output for peripheral components. The board's straightforward programming environment lowers the development barrier, allowing rapid prototyping of the face recognition applications through either block-based or Python coding approaches.

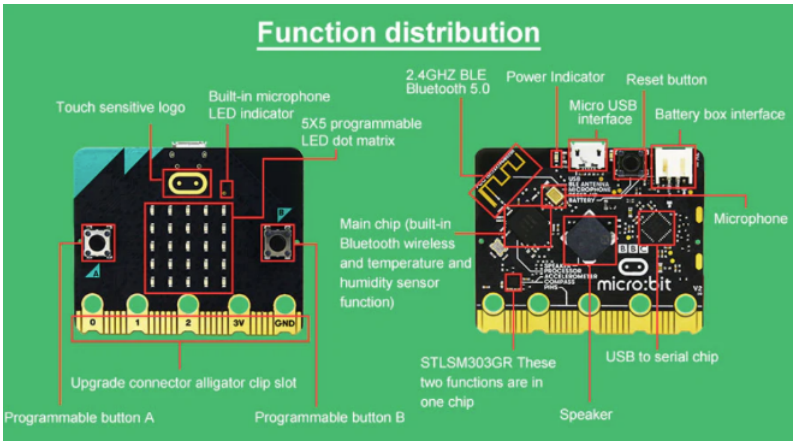

Figure 3.2. Micro:bit V2 Module

3.3 K210 Visual Identification Module

The K210 vision module serves as the AI processing core in this system, featuring a dual-core RISC-V processor with neural network acceleration capabilities that enable real-time face detection at 30 FPS. Equipped with an OV2640 camera (2MP resolution), it captures facial images and executes the trained face recognition model, then transmits processed results to the micro:bit via I2C communication.

The module operates on 5V power supplied by the micro:bit, while its compact size (35mmŒ20mm) and low power consumption (300mA@5V) make it ideal for embedded vision applications. The onboard KPU (Neural Network Processor) accelerates the pre-trained face recognition algorithm, achieving 0.2s response time per recognition cycle.

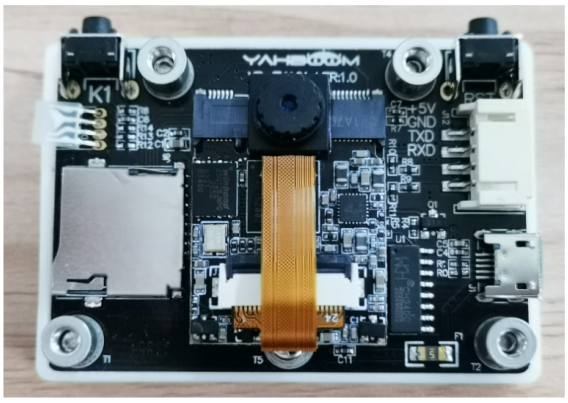

Figure 3.3. K210 Visual Module

3.4 Component Connection

We first install tires, universal wheel and lithium battery.

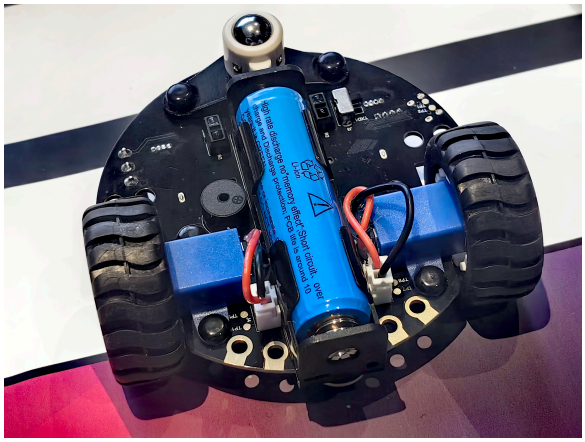
Figure 3.4. First Step Building Car

After that, we install Micro:bit V2 board.


Figure 3.5. Second Step Building Car

We also use M3 lock nut and screw to build K210 visual module bracket. Then we can install K210 module bracket on the car. Finally we use M3*6mm screw to install K210 visual module. Then our car is totally completed.

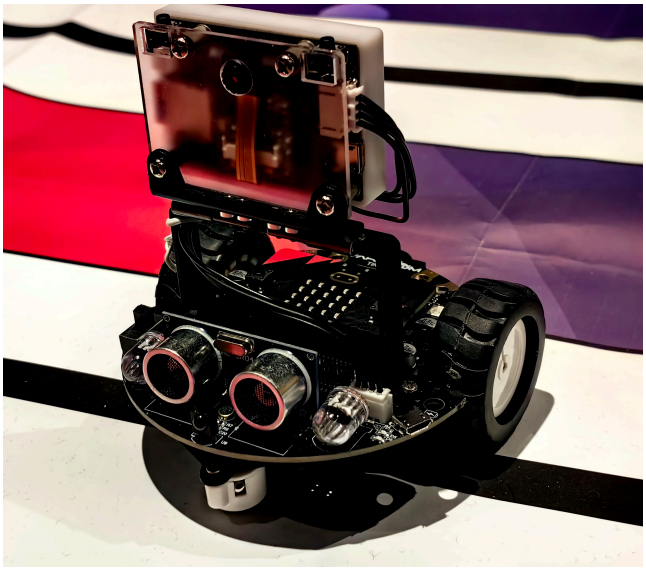

Figure 3.6. Final Assemble Step

For the communication between K210 and Micro:bit V2 module, we also need to use four pin cable to connect them together. TXD pin in the K210 is connected to RXD pin in Micro:bit. RXD of K210 is connected to TXD in Micro:bit.

4 Software Development Methodology

4.1 Implementation Overview

This chapter documents the iterative development process of the face recognition and encryption system, beginning with basic numeric displays on LEDs, progressively implementing face detection and model training, and culminating in comprehensive improvements to data hashing encryption. The complete source code is provided in Appendix B.

4.2 Display Images on LED Matrix

This section implements a simple program for displaying numbers 1-9 on the micro:bit development board, primarily designed for LED matrix visualization output. We employ a predefined dictionary of numeric patterns to encode each digit (1 through 9) as 5x5 pixel Image objects.

Listing 1 Display Image on LED Matrix

```
from microbit import *

# define numbers
NUMBERS = {
    1: Image("00900:"
             "00900:"
             "00900:"
             "00900:"
             "00900"),
    2: ...
    }

# show number images on led matrix
while True:
    for number in range(1, 10):  # 1-9
        display.show(NUMBERS[number])
        sleep(1000)  # show 1 second for every number
```

During execution, the program enters an infinite loop that sequentially retrieves digit patterns 1 through 9 from the NUMBERS dictionary and displays them on the micro:bit's 5Œ5 LED matrix via the display.show() method, with each number maintained for a 1-second display duration (controlled by sleep(1000)).

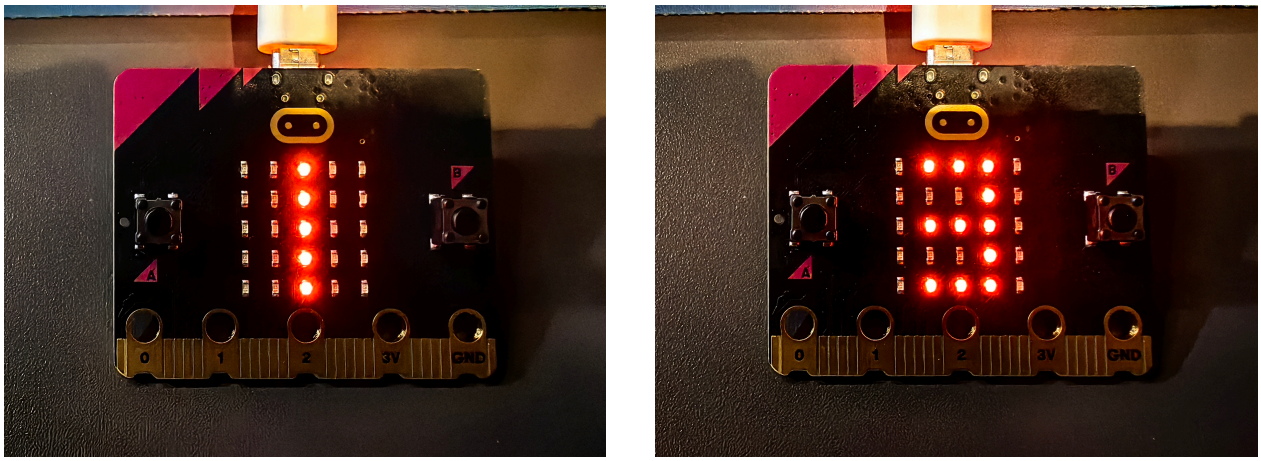

Figure 4.1. Display Images on LED Matrix

After executing the git push command, the terminal displayed a message indicating that all changes were successfully uploaded to the remote repository on GitHub, as shown in Figure 4.2.

```
27224@Guo MINGW64 /e/Desktop/robot/ProjectRobot1_2025/Exercise 1 (guo)
$ git push --set-upstream origin guo
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 662 bytes | 662.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'guo' on GitHub by visiting:
remote:      https://github.com/Ataraxiaii/ProjectRobot1_2025/pull/new/guo
remote:
To https://github.com/Ataraxiaii/ProjectRobot1_2025.git
 * [new branch]      guo -> guo
branch 'guo' set up to track 'origin/guo'.
```
Figure 4.2. Exercise 1 Git Push

4.3 Recognize Faces with Number

This section implements a face recognition interactive system based on the K210 chip, running on embedded devices through MicroPython. The core functionality includes real-time face detection and multimodal feedback, capable of distinguishing three registered users (ID 0/1/2), unregistered users, and no-face states. The system displays corresponding patterns on a 5Œ5 LED matrix: numbers 0/1/2 represent registered users, an "Œ" symbol indicates the no-face state, and a "U" shape identifies unregistered users. Simultaneously, the system plays unique

sound effects for different registered users, creating an audiovisual coordinated feedback mechanism.

4.3.1 Micro:bit V2 Part Code

For this task, I programmed on the official MakeCode website. In the implementation mechanism on micro:bit, the system loads K210's pre-trained model and configures audio parameters during the initialization phase. The main loop acquires face IDs through face_reg(), first performing a 200ms state stability verification to eliminate false recognition before triggering corresponding display and audio outputs based on the state type. To prevent conflicts between display refresh and music playback, a display_lock mutex is implemented while employing a non-blocking design to ensure real-time system responsiveness.

Listing 2 Face Recognition Key Code

```
def on_forever():
    global current_face, is_playing_music
    if not (display_lock):
        new_face = k210_models.face_reg()
        if new_face != current_face:
            start_time = input.running_time()
            while input.running_time() - start_time < 200:
                if k210_models.face_reg() != new_face:
                    return
            current_face = new_face
            if current_face == -1:
                update_display(-1)
            elif current_face == 0 or current_face == 1 or current_face == 2:
                update_display(current_face)
                is_playing_music = True
                if current_face == 0:
                    music.play(music.built_in_playable_melody(Melodies.POWER_UP),
                        music.PlaybackMode.UNTIL_DONE)
                ...
                elif current_face == 2:
                    music.play(music.built_in_playable_melody(Melodies.JUMP_DOWN),
                        music.PlaybackMode.UNTIL_DONE)
                is_playing_music = False
                update_display(current_face)
            else:
                update_display(current_face)
```

The programming on this website can be displayed in block form. Below is my complete code

design.

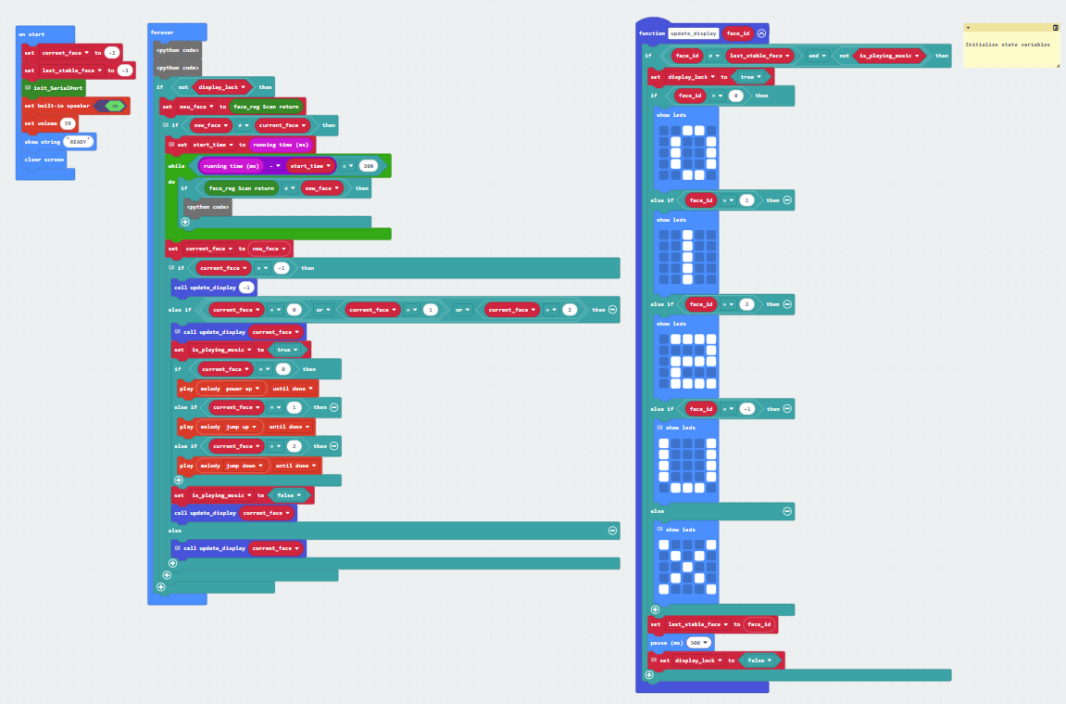

Figure 4.3. MakeCode Block Code

4.3.2 K210 Visual Module Part Code

The vision module code was primarily developed using the Mu Editor IDE. In terms of hardware architecture, the system employs the K210 as the main processor, integrated with a camera module and LCD display to construct the visual processing pipeline. During initialization, three critical models are loaded: a YOLOv2-based face detection model (face_detect_320x240.kmodel), a 5-point facial landmark detection model (ld5.kmodel), and a feature extraction model (feature_extraction.kmodel). Hardware interrupts are managed via fpioa_manager, with the BOOT button configured as the trigger signal for face registration functionality.

The main loop implements a multi-stage processing workflow: First, the YOLOv2 model detects facial regions within the video feed. Each detected face undergoes five-point landmark localization, followed by affine transformation to standardize alignment at 64Œ64 pixel resolution. The feature extraction model then converts the aligned facial image into a feature vector for similarity comparison against registered templates (threshold set at 80.5 points). Recognition results are annotated with color-coded bounding boxes - green indicates registered users (displaying user ID and matching score), while white frames denote unregistered states. The system supports real-time registration of new facial features through physical button activation, with all recognition results simultaneously output via serial communication protocol.

Listing 3 Face Model Key Code

```
while True:
    gc.collect()
    clock.tick()
    img = sensor.snapshot()
    kpu.run_with_output(img)
    dect = kpu.regionlayer_yolo2()# Detect faces
    fps = clock.fps()
    if len(dect) > 0:
        for l in dect :
            # Process detected face
            x1, y1, cut_img_w, cut_img_h= extend_box(l[0], l[1], l[2], l[3])
            face_cut = img.cut(x1, y1, cut_img_w, cut_img_h)
            face_cut_128 = face_cut.resize(128, 128)
            face_cut_128.pix_to_ai()
            # Get facial landmarks
            out = ld5_kpu.run_with_output(face_cut_128, getlist=True)
            face_key_point = []
            for j in range(5):
                x = int(KPU.sigmoid(out[2 * j])*cut_img_w + x1)
                y = int(KPU.sigmoid(out[2 * j + 1])*cut_img_h + y1)
                face_key_point.append((x,y))
            T = image.get_affine_transform(face_key_point, dst_point)
            image.warp_affine_ai(img, feature_img, T)
            feature = fea_kpu.run_with_output(feature_img, get_feature = True)
            del face_key_point
            scores = []
            for j in range(len(record_ftrs)):
                score = kpu.feature_compare(record_ftrs[j], feature)
                scores.append(score)

            # Compare with registered faces
            if len(scores):
                max_score = max(scores)
                index = scores.index(max_score)
                if max_score > THRESHOLD:
                    img.draw_string(0, 195, "persion:%d,score:%2.1f" %(index,
                    max_score), color=(0, 255, 0), scale=2)
                    recog_flag = True
                else:
                    img.draw_string(0, 195, "unregistered,score:%2.1f" %(max_score),
                    color=(255, 0, 0), scale=2)
            del scores
            ...
```

4.3.3 System Testing and Results

This section focuses on system testing and performance demonstration. First, we power on the car - upon activation, the micro:bit display will scroll the "READY" message.

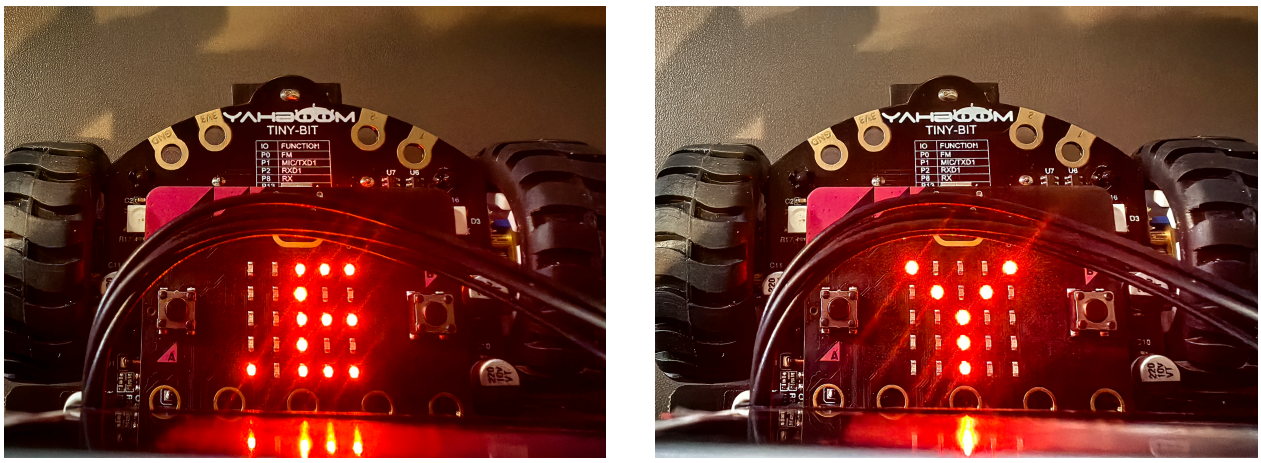

Figure 4.4. Scroll Display READY

When the vision module initializes successfully, the camera activates and begins streaming live footage. The display simultaneously shows real-time FPS metrics and operational instructions for face registration.

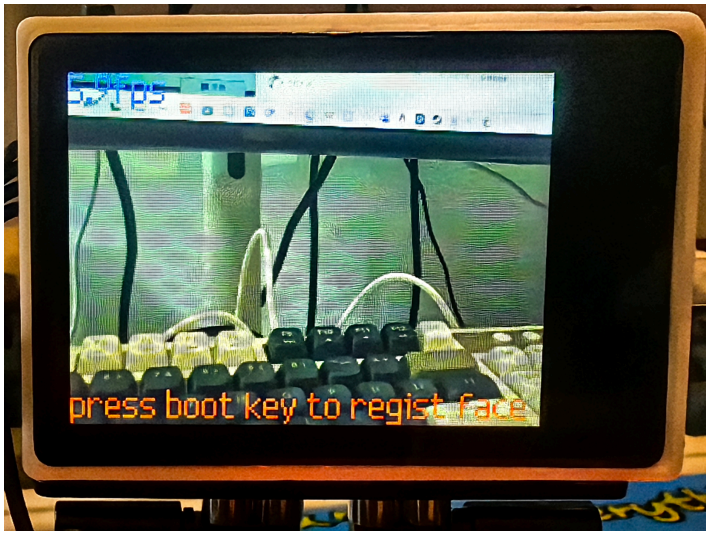

Figure 4.5. Camera Setup

When no faces are detected by the camera, the micro:bit displays an "x" symbol.

Figure 4.6. No Human Face

When an unregistered face is detected, the micro:bit displays a "U" character, indicating an undefined/unknown user.

Figure 4.7. Undefined Human Face

Pressing the BOOT button on the right side of the K210 module successfully registers a face. When a registered face is detected, the camera screen displays the matching confidence score, while the micro:bit simultaneously shows the corresponding face ID number (e.g., "0" for the first registered face, "1" for the second, and so on).

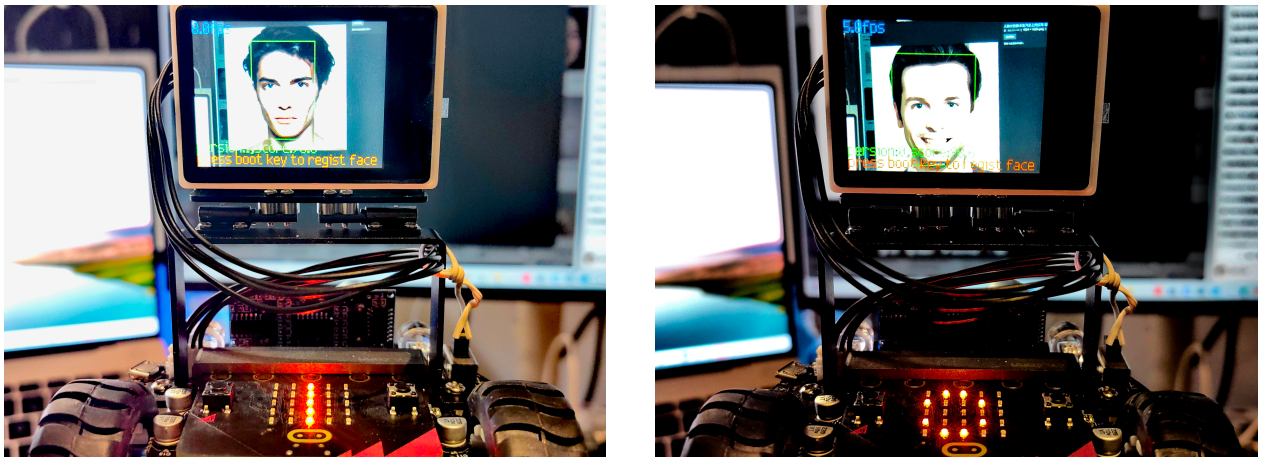

Figure 4.8. Registered Human Face

After executing the git push command, the terminal displayed a message indicating that all changes were successfully uploaded to the remote repository on GitHub, as shown in Figure 4.9.

```
27224@Guo MINGW64 /e/Desktop/robot/ProjectRobot1_2025 (guo)
$ git push git@github.com:Ataraxiaii/ProjectRobot1_2025.git
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 16 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (11/11), 2.09 MiB | 1.02 MiB/s, done.
Total 11 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:Ataraxiaii/ProjectRobot1_2025.git
   3567f91..919d1af  guo -> guo
```

Figure 4.9. Exercise 2 Git Push

4.4 Hash Recognized Face Data

This code implements facial feature data protection through cryptographic hashing. The system employs PBKDF2-HMAC-SHA256 algorithm to encrypt extracted 128-dimensional facial feature vectors, enhanced with predefined salt (SECURITY_SALT) and 50 hashing iterations (HASH_ITERATIONS) for security reinforcement. The hardware architecture integrates three core models: a YOLOv2 face detection model for facial region localization, a 5-point landmark model (ld5.kmodel) for precise facial alignment, and a feature extraction model (fea_kpu) that transforms aligned faces into encryptable feature vectors.

The encryption workflow specifically: (1) quantizes raw feature vectors to 0-255 range, (2) extracts the first 16 critical dimensions, then (3) processes them with salt through PBKDF2 algorithm. Feature dimensionality compression (128D16D) accommodates K210's memory constraints. During operation, the registration phase - triggered via BOOT button - stores hashes

in the record_ftrs list, while the recognition phase computes real-time hashes for encrypted comparison with the database, completely eliminating storage/transmission of raw biometric data.

Listing 4 Hash Face Key Code

```
def secure_hash(feature):
    """Generate secure hash of face features using PBKDF2-HMAC-SHA256"""
    quantized = bytes(int(min(max(x, 0), 1) * 255) for x in feature[:16])
    return uhashlib.pbkdf2_hmac('sha256', quantized +
    SECURITY_SALT, SECURITY_SALT, HASH_ITERATIONS)
while True:
    gc.collect()
    clock.tick()
    ...
    if start_processing:
        hashed = secure_hash(feature)
        record_ftrs.append(hashed)
        print("Registered face #%d" % len(record_ftrs))
        start_processing = False
        img.draw_rectangle(l[0], l[1], l[2], l[3], color=(255, 255, 255))
        msg_ = "R"
    current_hash = secure_hash(feature)
    matched = False
    for i, saved_hash in enumerate(record_ftrs):
        if saved_hash == current_hash:
            max_score = THRESHOLD + 1  # Show as recognized
            index = i
            recog_flag = True
            matched = True
            break
    ...
```

After executing the git push command, the terminal displayed a message indicating that all changes were successfully uploaded to the remote repository on GitHub.

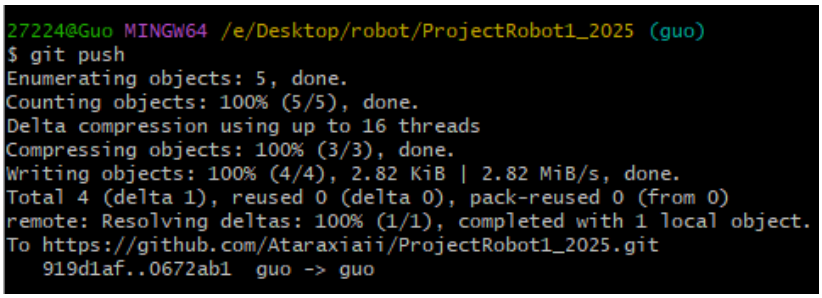

Figure 4.10. Exercise 3 Git Push

Finally, I submitted a pull request and, after verifying there were no conflicts, merged the branch code into the main default branch.

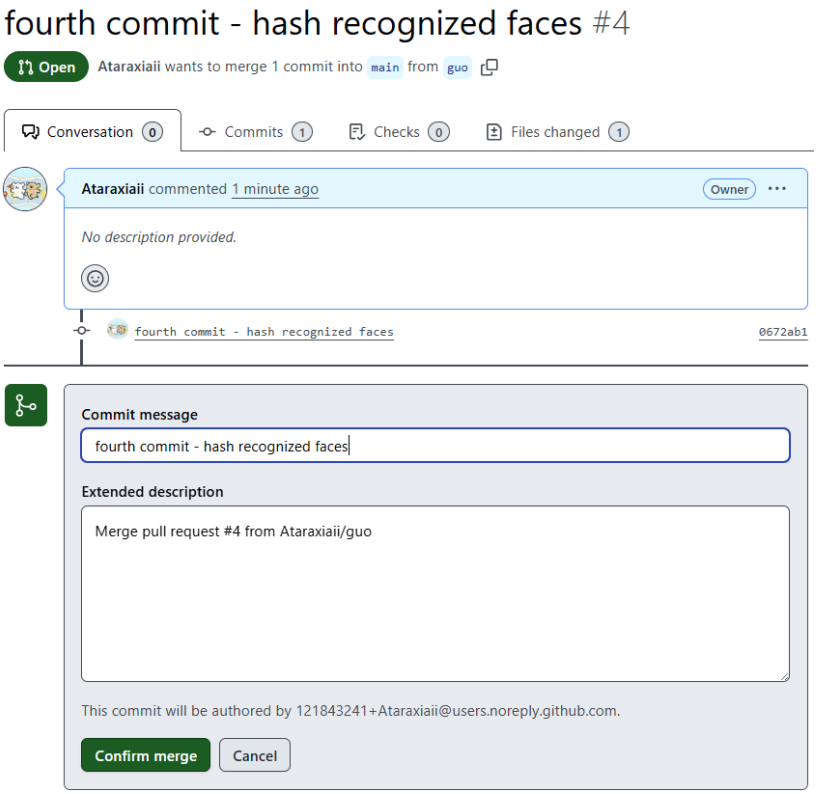

Figure 4.11. Commit Merge Code

A Commit History

This Git commit history documents the complete development cycle of a face recognition system, beginning with foundational LED matrix display functionality and progressively implementing core features including face detection with numeric binding and cryptographic hashing, culminating in hardware image uploads. The entire development occurred on the 'guo' branch with all commits originating from a single developer, pending final merge of these features into the main branch.

```
commit 5bc9554d23d3803f700b0239438b22927017ca6d (HEAD -> guo, origin/guo)
Author: Ataraxiaii <2722458837@qq.com>
Date: Fri Jun 27 22:31:43 2025 +0800
fifth commit - upload project hardware images

commit 0672ab1cc109b1654947ccc155672eb259fceb08
Author: Ataraxiaii <2722458837@qq.com>
Date: Fri Jun 27 21:13:42 2025 +0800
fourth commit - hash recognized faces

commit 919d1afb70d88ba911056b24e305c1cb304e649a
Author: Ataraxiaii <2722458837@qq.com>
Date: Fri Jun 27 21:00:37 2025 +0800
third commit - recognize faces with number showing

commit 3567f9176f306bfb34e92c070f9cafbfce3705f0
Author: Ataraxiaii <2722458837@qq.com>
Date: Thu Jun 26 22:28:06 2025 +0800
second commit - add exercise 1 comment
```

```
commit 218fab453334d34745467353c43d2b8f9ec738df
Author: Ataraxiaii <2722458837@qq.com>
Date: Wed Jun 25 23:24:26 2025 +0800
first commit - display images on led matrix

commit 25a16caba9dd334cea8882f883a5df0f42104b80 (origin/main, origin/HEAD,
main)
Author: Xuanru GUO <121843241+Ataraxiaii@users.noreply.github.com>
Date: Wed Jun 25 17:36:28 2025 +0800
Initial commit
```

B Source Code

Listing 5 Micro:bit Source Code

```
def update_display(face_id: number):
    global display_lock, last_stable_face
    if face_id != last_stable_face and not (is_playing_music):
        display_lock = True
        if face_id >= 0:
            basic.show_number(face_id)
        elif face_id == -1:
            # No face detected - show E
            basic.show_leds("""
                # . . . #
                # . . . #
                # . . . #
                # . . . #
                . # # # .
                """)
        else:
            # Unregistered face - show U
            basic.show_leds("""
                # . . . #
                . # . # .
                . . # . .
                . # . # .
                # . . . #
                """)
        last_stable_face = face_id
        basic.pause(300)
        # Display remains for 300ms
        display_lock = False
display_lock = False
is_playing_music = False
last_stable_face = 0
```

Listing 5 Micro:bit Source Code (Continued)

```
current_face = -2
last_stable_face = -3
# System initialization
k210_models.initialization()
music.set_built_in_speaker_enabled(True)
music.set_volume(59)
basic.show_string("READY")
basic.clear_screen()
def on_forever():
    global current_face, is_playing_music
    if not (display_lock):
        new_face = k210_models.face_reg()
        # State change detection (with debounce)
        if new_face != current_face:
            # Verify state stability (persists for 200ms)
            start_time = input.running_time()
            while input.running_time() - start_time < 200:
                if k210_models.face_reg() != new_face:
                    return
            current_face = new_face
            # No face handling
            if current_face == -1:
                update_display(-1)
            elif current_face == 0 or current_face == 1 or current_face == 2:
                # Registered face handling
                update_display(current_face)
                is_playing_music = True
                if current_face == 0:
                    music.play(music.built_in_playable_melody(Melodies.POWER_UP),
                        music.PlaybackMode.UNTIL_DONE)
                elif current_face == 1:
                    music.play(music.built_in_playable_melody(Melodies.JUMP_UP),
                        music.PlaybackMode.UNTIL_DONE)
                elif current_face == 2:
                    music.play(music.built_in_playable_melody(Melodies.JUMP_DOWN),
                        music.PlaybackMode.UNTIL_DONE)
                is_playing_music = False
                update_display(current_face)
            else:
                # Unregistered face
                update_display(current_face)
basic.forever(on_forever)
```

Listing 6 K210 Visual Final Code

```
# Author: Xuanru GUo
# Date: 2025-6-26
# Version: 1.0
# Description: Hash recognized face data

# Import necessary libraries for hardware interfacing and image processing
import sensor, image, time, lcd, gc
import uhashlib, ubinascii
from maix import KPU, GPIO, utils
from fpioa_manager import fm
from board import board_info
from modules import ybserial

# Security configuration constants
SECURITY_SALT = b"k210_secure_salt!"  # Unique salt value for hashing
HASH_ITERATIONS = 50  # Number of iterations for PBKDF2

# Initialize serial communication and display
serial = ybserial()
lcd.init()

# Camera setup
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time=200)  # Allow camera to stabilize
clock = time.clock()

feature_img = image.Image(size=(64,64), copy_to_fb=False)
feature_img.pix_to_ai()

# Face alignment reference points
FACE_PIC_SIZE = 64
NORM_FACTOR = FACE_PIC_SIZE / 112
dst_point = [
    (round(38.2946 * NORM_FACTOR), round(51.6963 * NORM_FACTOR)),
    (round(73.5318 * NORM_FACTOR), round(51.5014 * NORM_FACTOR)),
    (round(56.0252 * NORM_FACTOR), round(71.7366 * NORM_FACTOR)),
    (round(41.5493 * NORM_FACTOR), round(92.3655 * NORM_FACTOR)),
    (round(70.7299 * NORM_FACTOR), round(92.2041 * NORM_FACTOR))
]
# Load face detection model
kpu = KPU()
```

Listing 6 K210 Visual Final Code (Continued)

```
kpu.load_kmodel("/sd/KPU/yolo_face_detect/face_detect_320x240.kmodel")
anchor = (0.1075, 0.126875, 0.126875, 0.175, 0.1465625, 0.2246875,
          0.1953125, 0.25375, 0.2440625, 0.351875, 0.341875, 0.4721875,
          0.5078125, 0.6696875, 0.8984375, 1.099687, 2.129062, 2.425937)
kpu.init_yolo2(anchor, anchor_num=9, img_w=320, img_h=240, net_w=320, net_h=240,
               layer_w=10, layer_h=8, threshold=0.7, nms_value=0.2, classes=1)

# Load facial landmark detection model
ld5_kpu = KPU()
ld5_kpu.load_kmodel("/sd/KPU/face_recognization/ld5.kmodel")

# Load face feature extraction model
fea_kpu = KPU()
fea_kpu.load_kmodel("/sd/KPU/face_recognization/feature_extraction.kmodel")

def secure_hash(feature):
    """Generate secure hash of face features using PBKDF2-HMAC-SHA256"""
    quantized = bytes(int(min(max(x, 0), 1) * 255) for x in feature[:16])
    return uhashlib.pbkdf2_hmac('sha256', quantized +
    SECURITY_SALT, SECURITY_SALT, HASH_ITERATIONS)

# Button interrupt setup for face registration
start_processing = False
BOUNCE_PROTECTION = 50
fm.register(board_info.BOOT_KEY, fm.fpioa.GPIOHS0)
key_gpio = GPIO(GPIO.GPIOHS0, GPIO.IN)

def set_key_state(*_):
    """Interrupt handler for registration button"""
    global start_processing
    start_processing = True
    time.sleep_ms(BOUNCE_PROTECTION)

key_gpio.irq(set_key_state, GPIO.IRQ_RISING, GPIO.WAKEUP_NOT_SUPPORT)

# Face database and recognition parameters
record_ftrs = []
THRESHOLD = 80.5  # Minimum similarity score for recognition
recog_flag = False
def extend_box(x, y, w, h, scale):
    """Expand bounding box coordinates with safety checks"""
    x1_t = x - scale*w
    x2_t = x + w + scale*w
    y1_t = y - scale*h
    y2_t = y + h + scale*h
```

Listing 6 K210 Visual Final Code (Continued)

```
    x1 = int(x1_t) if x1_t>1 else 1
    x2 = int(x2_t) if x2_t<320 else 319
    y1 = int(y1_t) if y1_t>1 else 1
    y2 = int(y2_t) if y2_t<240 else 239
    return x1, y1, x2-x1+1, y2-y1+1

# Main processing loop
msg_ = ""
while True:
    gc.collect()
    clock.tick()

    # Capture and process image
    img = sensor.snapshot()
    kpu.run_with_output(img)
    dect = kpu.regionlayer_yolo2()  # Detect faces
    fps = clock.fps()

    if len(dect) > 0:
        for l in dect:
            # Extract face region
            x1, y1, cut_img_w, cut_img_h = extend_box(l[0], l[1], l[2], l[3])
            face_cut = img.cut(x1, y1, cut_img_w, cut_img_h)
            face_cut_128 = face_cut.resize(128, 128)
            face_cut_128.pix_to_ai()

            # Detect facial landmarks
            out = ld5_kpu.run_with_output(face_cut_128, getlist=True)

            # Calculate face alignment points
            face_key_point = []
            for j in range(5):
                x = int(KPU.sigmoid(out[2*j]) * cut_img_w + x1)
                y = int(KPU.sigmoid(out[2*j+1]) * cut_img_h + y1)
                face_key_point.append((x, y))
            # Align face and extract features
            T = image.get_affine_transform(face_key_point, dst_point)
            image.warp_affine_ai(img, feature_img, T)
            feature = fea_kpu.run_with_output(feature_img, get_feature=True)
            # Register new face if button pressed
            if start_processing:
                hashed = secure_hash(feature)
                record_ftrs.append(hashed)
                print("Registered face #%d" % len(record_ftrs))
                start_processing = False
                img.draw_rectangle(l[0], l[1], l[2], l[3], color=(255, 255, 255))
                msg_ = "R"
```

Listing 6 K210 Visual Final Code (Continued)

```
            # Compare with registered faces
            current_hash = secure_hash(feature)
            matched = False
            for i, saved_hash in enumerate(record_ftrs):
                if saved_hash == current_hash:
                    max_score = THRESHOLD + 1  # Show as recognized
                    index = i
                    recog_flag = True
                    matched = True
                    break

            # Display recognition results
            if matched:
                img.draw_string(0, 195, "person:%d,score:%2.1f" % (index+1,
                max_score), color=(0, 255, 0), scale=2)
                img.draw_rectangle(l[0], l[1], l[2], l[3], color=(0, 255, 0))
                msg_ = "Y%02d" % (index+1)
            else:
                img.draw_string(0, 195, "unregistered,score:0.0", color=(255, 0, 0),
                scale=2)
                img.draw_rectangle(l[0], l[1], l[2], l[3], color=(255, 255, 255))
                msg_ = "N"

            # Clean up temporary objects
            del face_cut_128, face_cut, feature, current_hash
            gc.collect()

    # Send recognition results via serial
    if len(dect) > 0:
        send_data = "$08" + msg_ + ",#"
        time.sleep_ms(5)
        serial.send(send_data)
    else:
        serial.send("#")

    # Display status information
    img.draw_string(0, 0, "%2.1ffps" % fps, color=(0, 60, 255), scale=2.0)
    img.draw_string(0, 215, "press boot key to register face",
    color=(255, 100, 0), scale=2.0)
    lcd.display(img)

# Clean up resources
kpu.deinit()
ld5_kpu.deinit()
fea_kpu.deinit()
```