# Lesson 13 Intelligent Access Control with Facial Recognition

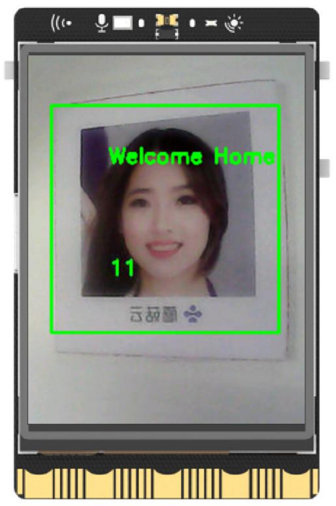
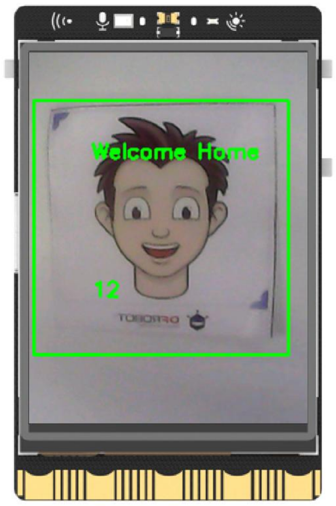
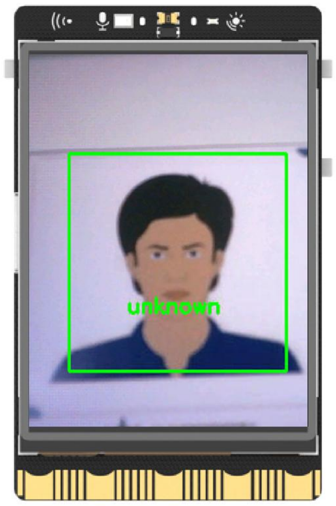
Have you ever experienced the frustration of forgetting your keys and being unable to enter your home, or the need to return to double-check if you locked the door after leaving?

Nowadays, facial recognition technology is widely used in various scenarios such as facial scanning for station access, facial payment, campus security management, and facial attendance. If our home door locks could also be opened by recognizing our faces, it would eliminate the inconvenience of losing or forgetting keys.

In this lesson, let's combine the concept of "UNIHIKER" and a camera to design an intelligent access control system that can recognize the owner's face.

## Task Objectives

Using a USB camera, we can display the real-time environmental footage. When a recognized face is detected, the corresponding identification (ID) marker will be displayed on the screen, and a servo motor will be controlled to simulate the opening of the door.

| | | |
|---|---|---|
|  |  |  |

## Knowledge points

1. Familiarize yourself with servo motors.

2. Learn how to control servo motor rotation using the Pinpong library.

3. Understand facial recognition.

4. Learn how to perform facial recognition using the OpenCV library.

5. Learn how to draw text and rectangles on images using the OpenCV library.

6. Learn how to convert image formats using the PIL library's Image module.

7. Learn file handling methods using the os library.

## Material List

### Hardware List:

| | | |
|---|---|---|
| 行空板 x1 | Type-C&Micro 二合一 USB 线 x1 | 免驱 USB 摄像头 x1 |
| 舵机 x1 | | |

**Software Preparation:** Mind+ Programming Softwarex1

## Knowledge background

1. What is a servo motor?

A servo motor is an actuator that can control the rotation of an object to a specific position (angle). There are two common types of servo motors: 180° and 360°. In this case, we are using a 180° servo motor.

2. Controlling Servo Rotation with the angle() Method in the Pinpong Library's Servo Class

The angle() method in the Servo class of the Pinpong library allows us to control the rotation of a servo motor to a specific position. Before using this method, we need to import the Pinpong library, initialize the board, and instantiate the Servo class to create an object.

```
from pinpong.board import Board, Pin, Servo  # Import the Pinpong libra
ry


Board("UNIHIKER").begin()  # Initialize and select the board type, if n
ot specified, it will be automatically recognized


s1 = Servo(Pin(Pin.P23))  # Initialize the servo pin by passing it to t
he Servo class

s1.angle(10)  # Control the servo to rotate to the initial position of
10 degrees (closed door)
```

In this case, the Servo module is similar to the Board module, both of which are modules under the pinpong.board package. "P23" refers to the pin to which the servo is connected on the board. "Pin(Pin.P23)" creates a pin object, which is then passed into the Servo class for instantiation, resulting in the creation of a servo object. "10" refers to the 10° position.

3. What is Facial Recognition?

Facial recognition is a biometric technology that involves identifying individuals based on their facial features. It utilizes cameras or webcams to capture images or video streams containing faces and automatically detects and tracks faces within the images. This technology then performs a series of related techniques to recognize and identify the detected faces. Facial recognition is also commonly referred to as face recognition or facial identification.

In simple terms, facial recognition is the process of analyzing and identifying a person's face image to determine their identity. It addresses the question of "whose face is this?"

Facial recognition generally involves the following four steps:

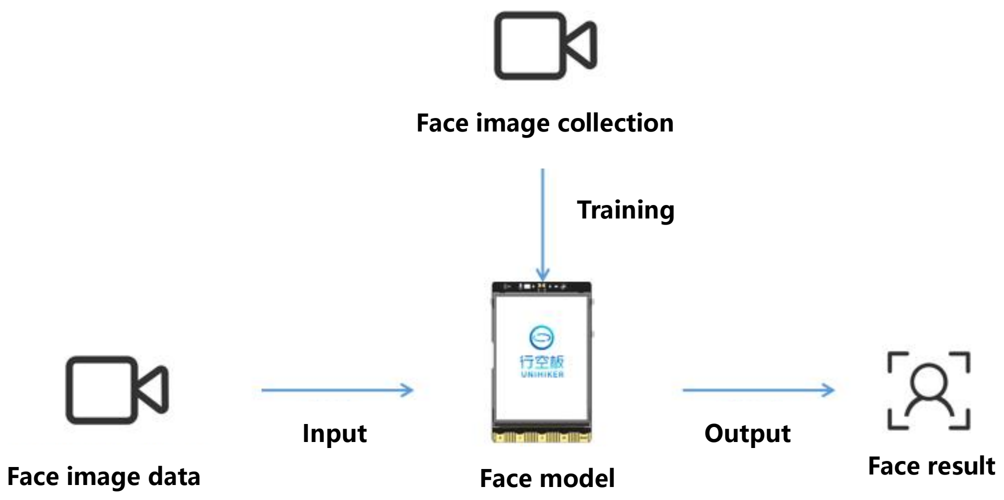
1) Face detection: This is the process of locating the regions of faces in an image. (i.e., determining whether it is a face or not)

2) Face preprocessing: Adjusting the face image to make it appear clearer and more suitable for analysis.

3) Face data collection and learning: Collecting the detected faces that have undergone preprocessing and then learning and memorizing them.

4) Face recognition: Searching among the collected and learned faces to find the one that is most similar to the face being identified.

Among them, the first three steps lay the foundation for the final fourth step of facial recognition. When designing a program for facial recognition, we first need to collect a set of face images from the detected and preprocessed faces. Then, we train these image sets to create a facial model. Finally, during the recognition phase, we capture real-time image data and input it into the generated facial model to obtain a facial result, as shown in the diagram below.



4. What is LBPH Facial Recognition?

Facial detection is the first step in facial recognition. During facial detection, we can utilize cascaded classifiers. Similarly, during facial recognition, we can employ various methods.

OpenCV provides three facial recognition methods: LBPH (Local Binary Patterns Histograms), Eigenfaces, and Fisherfaces. In this lesson, we are using the LBPH method. We will train a facial model and detect real-time faces using the LBPH recognizer provided by OpenCV. Therefore, in this project, we will divide the process into three steps: "Face Data Collection," "Facial Model Training," and "Real-time Facial Recognition."

LBPH (Local Binary Patterns Histograms) is a facial recognition algorithm used to identify faces. The basic idea of the facial recognition method based on LBPH is as follows: first, each pixel is considered as the center, and the relationship between its grayscale value and the surrounding pixels is determined. The pixel is then encoded into a binary pattern, resulting in an LBP (Local

Binary Patterns) encoded image of the entire image. Next, the LBP image is divided into regions, and the LBP histogram is obtained for each region. Finally, the LBP histograms of different facial images are compared to achieve facial recognition. The advantage of this method is that it is not affected by lighting conditions, scaling, rotation, or translation.

Here, we only need a basic understanding of LBPH.

5. Using the LBPH Recognizer in the OpenCV Library

(1) Creating the LBPH Recognizer

In OpenCV, you can create an instance of the LBPH recognizer using the function "cv2.face.LBPHFaceRecognizer_create()".

```
recognizer = cv2.face.LBPHFaceRecognizer_create()  # Create an instance
 of the LBPH recognizer model
```

In this case, "recognizer" is a variable used to store the generated instance of the recognizer model.

(2) Training the LBPH Recognizer Model using the "train()" Method of the Recognizer Object

After generating the LBPH recognizer, you can train the model using the "train()" method. During training, you need to provide the facial image samples and their corresponding ID numbers to the empty model file.

```
recognizer.train(faces, np.array(Ids))  # Pass the face samples list an
d ID samples list to the empty LBPH recognizer model to get a complete
face model
```

In this case, "faces" is a list that stores the facial images used for training, and "np.array(Ids)" refers to the ID labels corresponding to the facial images.

(3) Saving the LBPH Recognizer Model using the "save()" Method of the Recognizer Object

After generating the LBPH recognizer, you can use the "save()" method to save the specified model to a specified path.

```
img_src = '/root/image/project14'  # Define the save path (Specify to a
 fixed location in Mind+)

recognizer.save(img_src+'/model.yml')  # Save the generated model
```

In this case, "img_src" refers to the specified path for storing the model, and "model.yml" refers to the filename of the saved model file.

(4) Reading the LBPH Recognizer Model using the "read()" Method of the Recognizer Object

After generating the LBPH recognizer, you can use the "read()" method to read the model.

```
recognizer.read(img_src + '/model.yml')  # Read the trained face model
```

```
from the specified path
```

In this case, "img_src" refers to the path where the model is located, and "model.yml" refers to the filename of the model being read.

(5) Predicting Facial Images using the "predict()" Method of the LBPH Recognizer Object

After reading the facial model, you can use the "predict()" method to recognize and predict facial images.

```
img_id, confidence = recognizer.predict(gray[y:y + h, x:x + w])  # Pred
ict on the grayscale image within the specified rectangular region (use
d for face presentation)
```

In this case, "gray[y:y + h, x:x + w]" refers to the grayscale image of the facial region used for prediction, "img_id" refers to the returned label of the recognition result, and "confidence" refers to the confidence score returned. The confidence score is used to measure the distance between the recognition result and the original model. A score of 0 indicates a perfect match, and typically, values below 50 are considered acceptable. If the value is greater than 80, it is considered a significant difference.

6. Drawing Text on Images using the "putText()" Function in the OpenCV Library

The "putText()" function in the OpenCV library allows us to draw text on images. To use this function, we need to first import the library and set the desired font type.

```
import cv2  # Import the OpenCV library

font = cv2.FONT_HERSHEY_SIMPLEX  # Set the font type to normal-sized sa
ns-serif

cv2.putText(img, 'shooting', (10, 50), font, 0.6, (0, 255, 0), 2)  # Di
splay the text "shooting" on the image, indicating that the face is bei
ng captured
```

In this case, "cv2.FONT_HERSHEY_SIMPLEX" is the chosen font type, stored in the variable "font". "img" refers to the image on which the text will be displayed. "shooting" represents the specific text content to be displayed. "(10, 50)" indicates the position where the text will be drawn, with the bottom left corner of the text as the starting point. "0.6" corresponds to the size of the text. "(0, 255, 0)" represents the color of the text, in this case, green, using RGB values. "2" corresponds to the thickness of the line used for drawing the text.

7. Drawing Rectangles on Images using the "rectangle()" Function in the OpenCV Library

The "rectangle()" function in the OpenCV library allows us to draw rectangular boxes on images to highlight the detected faces.

```
cv2.putText(img, 'Done, Please quit', (10, 50), font, 0.6, (0, 255, 0),
 2)  # Display the text "Done, Please quit" to indicate completion
```

In this case, "img" refers to the original image on which the rectangle will be drawn. "(x, y)" corresponds to the top-left coordinates of the rectangle, while "(x + w, y + h)" refers to the bottom-right coordinates of the rectangle. "(0, 255, 0)" represents the color of the rectangle's border, specified using RGB values, with green being used in this case. "2" indicates the thickness of the rectangle's border.

8. Saving Images using the "imwrite()" Function in the OpenCV Library

The "imwrite()" function in the OpenCV library can be used to save a specified image to a specified path.

```
lena = cv2.imread("lena.bmp")  # Read the image "lena.bmp"

r = cv2.imwrite("result.bmp", lena)  # Save it as "result.bmp" image
```

In this case, "lena.bmp" is an image located in the current directory. After reading and creating a copy of this image, it will be saved in the current directory with the name "result.bmp".

9. Image Processing with the PIL Library's Image Module

(1) Opening an Image with the "open()" Function

The "open()" function in the Image module of the PIL library can be used to open an image located at a specified path.

```
    image = Image.open(image_path)
```

In this case, "image_path" refers to the path of the image.

(2) Converting the Mode of an Image with the "convert()" Function

The "convert()" function in the Image module of the PIL library can be used to convert the mode of an image.

For color images, regardless of their image format (PNG, BMP, or JPG), when opened using the "open()" function in the Image module of PIL, the returned image object's mode will be "RGB". On the other hand, for grayscale images, regardless of their image format, when opened, the mode will be "L". PIL supports 9 different image modes, with "L" representing grayscale images where each pixel is represented by 8 bits, with 0 representing black and 255 representing white. Other modes represent different color spaces.

```
image = Image.open(image_path).convert('L')# Convert the color image to
 grayscale
```

In this case, "image_path" refers to the path of the image.

## Hands-on practice

### Task Description 1: Collecting Facial Images

In this task, we will capture 3 different faces using a camera. Each face will be captured with 50 images and labeled accordingly, so that later during recognition, the corresponding face labels can be displayed.

#### 1. Hardware setup

**STEP1:** Connect the UNIHIKER to the computer via a USB cable.

**STEP2:** Connect the USB camera to the UNIHIKER.

#### 2. program coding

**STEP 1:** Creating and Saving Project Files

Launch Mind+ and save the project as "013 Intelligent Access Control with Facial Recognition

".

**STEP 2:** Creating and Saving Python Files

Create a Python program file named "main1.py" and double-click to open it.

**STEP 3:** Programming

(1) Importing Required Libraries

In this task, we will need to import the OpenCV library to access the camera and display the video stream. We will also need to import the os library to create a folder for saving the captured images.

```
import cv2  # Import the OpenCV library

import os  # Import the os library
```

(2) Defining the Path and Folder for Image Saving

In this task, we will be capturing facial images and saving them. Therefore, it is necessary to define the path where the images will be saved and create an empty folder to store the images later.

```
img_src = '/root/image/project14'  # Define the image path (Specify to a fixed location in Mind+)

os.system('mkdir -p ' + img_src + '/new/')  # Create a folder named "new" in the specified path (The folder needs to be created first before storing the images)

# img_src = os.getcwd()  # Specify the current working directory if running directly
```

(3) Setting up the Camera

In order to display the real-time camera feed on the UNIHIKER screen, we need to perform four steps. First, we need to open the camera. Then, we set a buffer of 1 frame for the video stream to avoid any lag or high delay. After that, we create a window, and finally, we set the window to fullscreen mode to display the complete video feed on the window.

```
cap = cv2.VideoCapture(0)  # Open and initialize the camera

cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)  # Set the buffer size to 1 frame to reduce latency

cv2.namedWindow('frame', cv2.WND_PROP_FULLSCREEN)  # Create a window named 'frame' with the default property of being able to go fullscreen

cv2.setWindowProperty('frame', cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)  # Set the 'frame' window to fullscreen
```

(4) Setting the Font Type

In order to display some prompt text on the screen during facial image capture, it is necessary to define the font type beforehand.

```
font = cv2.FONT_HERSHEY_SIMPLEX  # Set the font type to a normal-sized sans-serif font
```

(5) Loading the Face Detection Classifier

To detect faces and capture images, we need to use the appropriate classifier tool. Here, we load the classifier in advance.

```
detector = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')  # Load the face detection classifier
```

(6) Defining the Initial Image Sample Count

To keep track of the number of images captured later on, we define the initial image sample count as 0.

```
sampleNum = 0  # Initialize the sample number to 0
```

(7) Labeling the Captured Facial Images

During the facial image capture process, in order to differentiate between different faces, we need to assign an ID number to each face. Here, we achieve this by inputting numerical values through the terminal before capturing the images.

```
ID = input('enter your id: ')  # Enter the ID number for the face image data
```

(8) Facial Image Capture

Next, we need to capture facial images using the webcam. This process can be divided into four steps. Firstly, we display the real-time video stream from the camera on the UNIHIKER screen in proportion. Then, we detect and retrieve facial data. After that, we set the desired number of detected faces. Once we have collected 50 images, we display a completion message on the screen. Otherwise, we show a prompt indicating that the images are being captured and save them into the previously defined folder. Finally, we set the program to exit when the "b" key is pressed.

```
while True:
    ret, img = cap.read()  # Read the image frame by frame
    # img = cv2.flip(img, 1)  # Mirror the image (horizontally flip the img image)
    if ret:  # If an image is successfully read
        h, w, c = img.shape  # Record the shape of the image, which includes the height, width, and channels
        w1 = h * 240 // 320
        x1 = (w - w1) // 2
        img = img[:, x1:x1 + w1]  # Crop the image
        img = cv2.resize(img, (240, 320))  # Resize the image to match the dimensions of the PinPong board
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convert the image to grayscale
        faces = detector.detectMultiScale(gray, 1.3, 5)  # Detect and obtain face recognition data
        for (x, y, w, h) in faces:  # If a face is detected
            cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

```
 # Draw a rectangle box around the face
            sampleNum = sampleNum + 1  # Increment the sample number
            if sampleNum <= 50:  # If the number of samples is less than or equal to 50
                cv2.putText(img, 'shooting', (10, 50), font, 0.6, (0, 255, 0), 2)  # Display the text "shooting" on the image, indicating that the face is being captured
                # Save the cropped face image with a name format of sampleNum.UserID.jpg
                cv2.imwrite(img_src + '/new/' + str(sampleNum) + '.' + str(ID) + ".jpg", gray[y:y + h, x:x + w])
            else:  # If the number of images exceeds 50
                cv2.putText(img, 'Done, Please quit', (10, 50), font, 0.6, (0, 255, 0), 2)  # Display the text "Done, Please quit" to indicate completion
        cv2.imshow('frame', img)  # Display the image on the 'frame' window

        key = cv2.waitKey(1)  # Delay each frame by 1ms, delay cannot be 0, otherwise the result will be a static frame
        if key & 0xFF == ord('b'):  # Press 'b' to exit
            break
```

(9) Closing the Camera and Windows

After pressing the "b" key to exit, we release the camera resources and close all image windows.

```
cap.release()  # Release the camera
cv2.destroyAllWindows()  # Close all windows
```

**Tips:** The complete example program is as follows:

```
'''
Run the program, enter the ID in the terminal and press Enter. The screen will display the camera image. Adjust the position until a green box appears, indicating that the photo is being taken. When "done" is d
```

```
isplayed, the process is complete.

The captured images will be saved in the "/root/image/project14/new" f
older. This code captures 50 images, but you can modify the code param
eters to capture more images.

'''


import cv2  # Import the OpenCV library

import os  # Import the os library


img_src = '/root/image/project14'  # Define the image path (Specify to
 a fixed location in Mind+)

os.system('mkdir -p ' + img_src + '/new/')  # Create a folder named "n
ew" in the specified path (The folder needs to be created first before
 storing the images)

# img_src = os.getcwd()  # Specify the current working directory if ru
nning directly


cap = cv2.VideoCapture(0)  # Open and initialize the camera

cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)  # Set the buffer size to 1 frame
to reduce latency

cv2.namedWindow('frame', cv2.WND_PROP_FULLSCREEN)  # Create a window n
amed 'frame' with the default property of being able to go fullscreen

cv2.setWindowProperty('frame', cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FUL
LSCREEN)  # Set the 'frame' window to fullscreen


font = cv2.FONT_HERSHEY_SIMPLEX  # Set the font type to a normal-sized
 sans-serif font

detector = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_
frontalface_default.xml')  # Load the face detection classifier

sampleNum = 0  # Initialize the sample number to 0

ID = input('enter your id: ')  # Enter the ID number for the face imag
e data
```

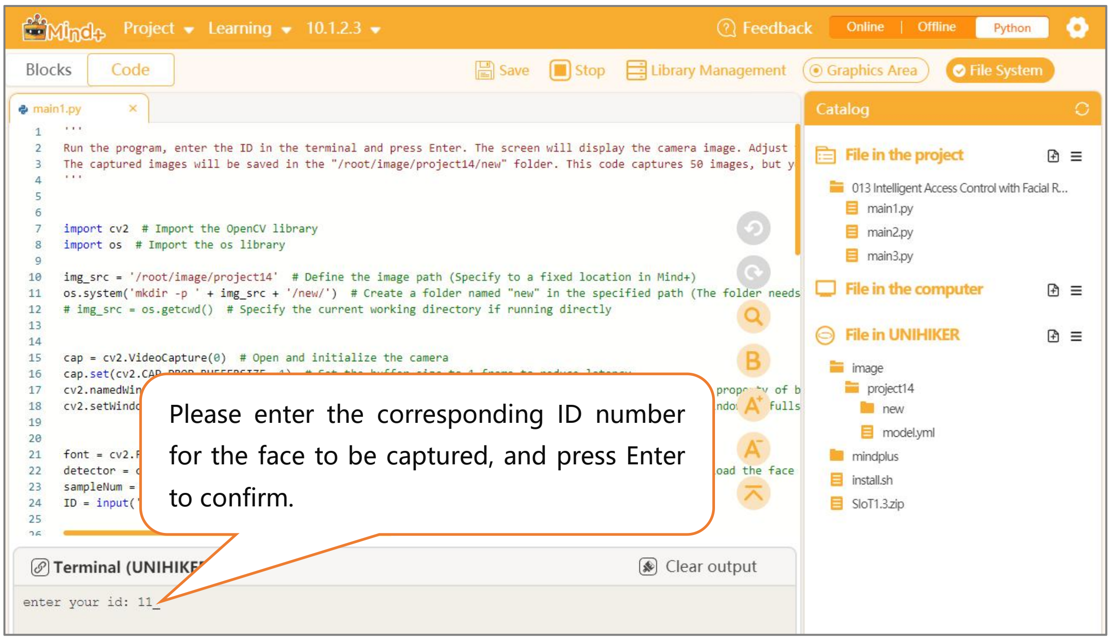
```
while True:
    ret, img = cap.read()  # Read the image frame by frame
    # img = cv2.flip(img, 1)  # Mirror the image (horizontally flip th
e img image)
    if ret:  # If an image is successfully read
        h, w, c = img.shape  # Record the shape of the image, which in
cludes the height, width, and channels
        w1 = h * 240 // 320
        x1 = (w - w1) // 2
        img = img[:, x1:x1 + w1]  # Crop the image
        img = cv2.resize(img, (240, 320))  # Resize the image to match
 the dimensions of the PinPong board
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convert the im
age to grayscale
        faces = detector.detectMultiScale(gray, 1.3, 5)  # Detect and
obtain face recognition data
        for (x, y, w, h) in faces:  # If a face is detected
            cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
  # Draw a rectangle box around the face
            sampleNum = sampleNum + 1  # Increment the sample number
            if sampleNum <= 50:  # If the number of samples is less th
an or equal to 50
                cv2.putText(img, 'shooting', (10, 50), font, 0.6, (0,
255, 0), 2)  # Display the text "shooting" on the image, indicating th
at the face is being captured
                # Save the cropped face image with a name format of sa
mpleNum.UserID.jpg
                cv2.imwrite(img_src + '/new/' + str(sampleNum) + '.' +
 str(ID) + ".jpg", gray[y:y + h, x:x + w])
            else:  # If the number of images exceeds 50
                cv2.putText(img, 'Done, Please quit', (10, 50), font,
0.6, (0, 255, 0), 2)  # Display the text "Done, Please quit" to indica
te completion
        cv2.imshow('frame', img)  # Display the image on the 'frame' w
```

```
indow


        key = cv2.waitKey(1)  # Delay each frame by 1ms, delay cannot
be 0, otherwise the result will be a static frame

        if key & 0xFF == ord('b'):  # Press 'b' to exit

            break


cap.release()  # Release the camera

cv2.destroyAllWindows()  # Close all windows
```

## 3. Running the Program

**STEP1:** Remote Connect to the UNIHIKER, Run the Program, and Observe the Results

Running Mind+, you will see a prompt in the terminal asking you to enter the ID number corresponding to the face you want to capture. Here, we input "11" and press Enter to confirm. Note: The ID number can be adjusted here.

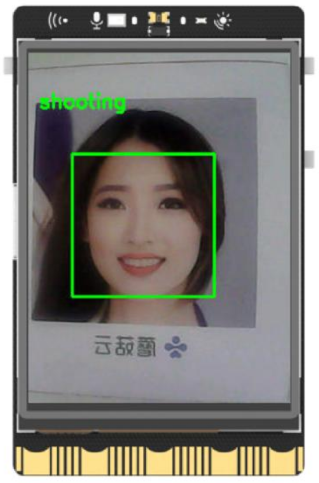
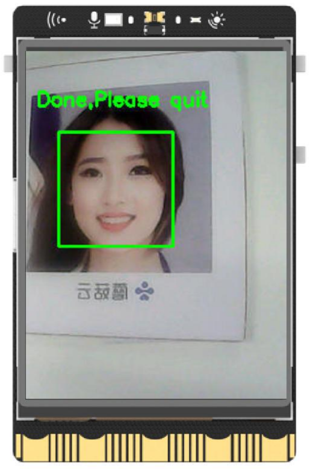


```
'''
Run the program, enter the ID in the terminal and press Enter. The screen will display the camera image. Adjust
The captured images will be saved in the "/root/image/project14/new" folder. This code captures 50 images, but y
'''



import cv2  # Import the OpenCV library
import os  # Import the os library

img_src = '/root/image/project14'  # Define the image path (Specify to a fixed location in Mind+)
os.system('mkdir -p ' + img_src + '/new/')  # Create a folder named "new" in the specified path (The folder needs
# img_src = os.getcwd()  # Specify the current working directory if running directly


cap = cv2.VideoCapture(0)  # Open and initialize the camera
cap.set(cv2.CAP
cv2.namedWin                                                    property of b
cv2.setWindo                                                    ndo      fulls


font = cv2.F
detector = c                                                    oad the face
sampleNum =
ID = input('
```

Please enter the corresponding ID number for the face to be captured, and press Enter to confirm.

```
enter your id: 11_
```

Subsequently, the camera automatically starts, and the real-time captured video stream is displayed in full screen on the board. Next, align the camera with a face, and you will notice that the face is highlighted with a green bounding box. Above the bounding box, the text "shooting" is displayed, indicating that we are capturing images of that face. When 50 images are successfully captured, the text "Done, Please quit" is displayed above the face, indicating completion. At this point, you can press the onboard button "b" to exit the program.

| | |
|---|---|
|  Image capture in progress... |  Image capture completed |

**STEP2:** Multiple Captures

Following the same method as described above, run the program two more times to capture two additional faces. Name the IDs of these faces as "12" and "13" respectively.

| | | |
|---|---|---|
| ID: 11 | ID: 12 | ID: 13 |

**Tips:** Here, we are capturing face images from a card. Similarly, we can also align the camera towards ourselves and capture real faces.

## Task Description 2: Training the Face Model

In the previous task, we captured three different face images. Now, we need to process these images and train them into a face model.

## 1. program coding

**STEP 1:** Creating and Saving Project Files

Create a new Python program file named "main2.py" and double-click to open it.

**STEP 2:** Programming

(1) Importing the numpy and PIL libraries, as well as the Image module

In this task, we will be using the numpy library, the PIL library's Image module, and the os library for image processing. We will also be using the face detection classifier and recognizer from the OpenCV library to create the model file. Therefore, we need to import them.

```
import cv2  # Import the OpenCV library

import os  # Import the os library

import numpy as np  # Import the numpy library

from PIL import Image  # Import the Image module from the PIL library
```

(1) Defining the Path to Save the Model File

In this task, we will be training the captured face images into a face model. Here, we define the location where the model file will be saved in advance.

```
img_src = '/root/image/project14'  # Define the save path (Specify to a
 fixed location in Mind+)
```

(2) Loading the Face Detection Classifier and LBPH Recognizer

In order to detect faces in the saved face images, we still need to use a classifier. Additionally, to train the images into a model, we need to create an empty model instance in advance. Therefore, here, we pre-load the face detection classifier and LBPH recognizer.

```
# Initialize the face detector and recognizer

detector = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_f
rontalface_default.xml')  # Load the face detection classifier

recognizer = cv2.face.LBPHFaceRecognizer_create()  # Create an instance
 of the LBPH recognizer model (empty)
```

(3) Defining the Function - Getting Face Sample List and ID List

When training the face images into a model, we need to provide the face image samples and their corresponding ID numbers to the empty model file. Therefore, we need to process the captured images to extract the necessary information. Since this process is complex, we will write it as a function.

The entire process can be divided into four steps. First, we concatenate the path and specific name of the image to represent each face image with a complete path. Second, we create two empty lists to store the face image samples and ID samples, respectively. Next, we iterate through each image, draw a bounding box around the face region, and store it in the face image sample list. We also extract the ID number from the image name and store it in the ID list. Finally, we return the two lists containing the face regions and ID numbers, respectively.

```
def get_images_and_labels(path):

    # Concatenate the image path to the specific image name, such as '/
root/image/project14/new/50.1.jpg', and store it in the 'image_paths' l
ist

    image_paths = [os.path.join(path, f) for f in os.listdir(path)]

    # Create an empty list for face samples

    face_samples = []

    # Create an empty list for IDs

    ids = []

    for image_path in image_paths:  # Traverse the image path

        # Print the name of each image (image name with path)

        print(image_path)

        # Convert the color image to grayscale

        image = Image.open(image_path).convert('L')

        # Convert the grayscale image format to a Numpy array

        image_np = np.array(image, 'uint8')

        '''To get the ID, we split the image path and retrieve the rele
vant information'''

        # Split by ".", if the last group is "jpg", then execute the fo
llowing steps

        if os.path.split(image_path)[-1].split(".")[-1] != 'jpg':

            continue

        # Extract the ID number from the complete path name of the imag
e, which is the ID number we set when capturing the image

        image_id = int(os.path.split(image_path)[-1].split(".")[1])

        # Detect the face in the array format and store the results in
'faces'
```

```
        faces = detector.detectMultiScale(image_np)  # Detect faces

        # Crop out the face portion from the array format face image and store them in the face samples list, and store the ID number of the image in the ID samples list

        for (x, y, w, h) in faces:

            face_samples.append(image_np[y:y + h, x:x + w])

            ids.append(image_id)

    return face_samples, ids  # Return the face samples list and ID samples list
```

(4) Getting the Face Sample List and ID List

Here, we call the previously defined function and pass in the image path and name to get the face sample list and ID list.

```
faces, Ids = get_images_and_labels(img_src+'/new/')  # Pass in the path and the folder name of the image to get the face samples list and ID samples list
```

(5) Training the Model

After obtaining the face sample and ID sample lists, we proceed to train the model using them.

```
recognizer.train(faces, np.array(Ids))  # Pass the face samples list and ID samples list to the empty LBPH recognizer model to get a complete face model
```

(6) Saving the Model

Finally, after training the facial model with the collected images, we save it for future use.

```
recognizer.save(img_src+'/model.yml')  # Save the generated model

print("generate model done")  # Print "Model has been generated"
```

**Tips:** The complete example program is as follows:

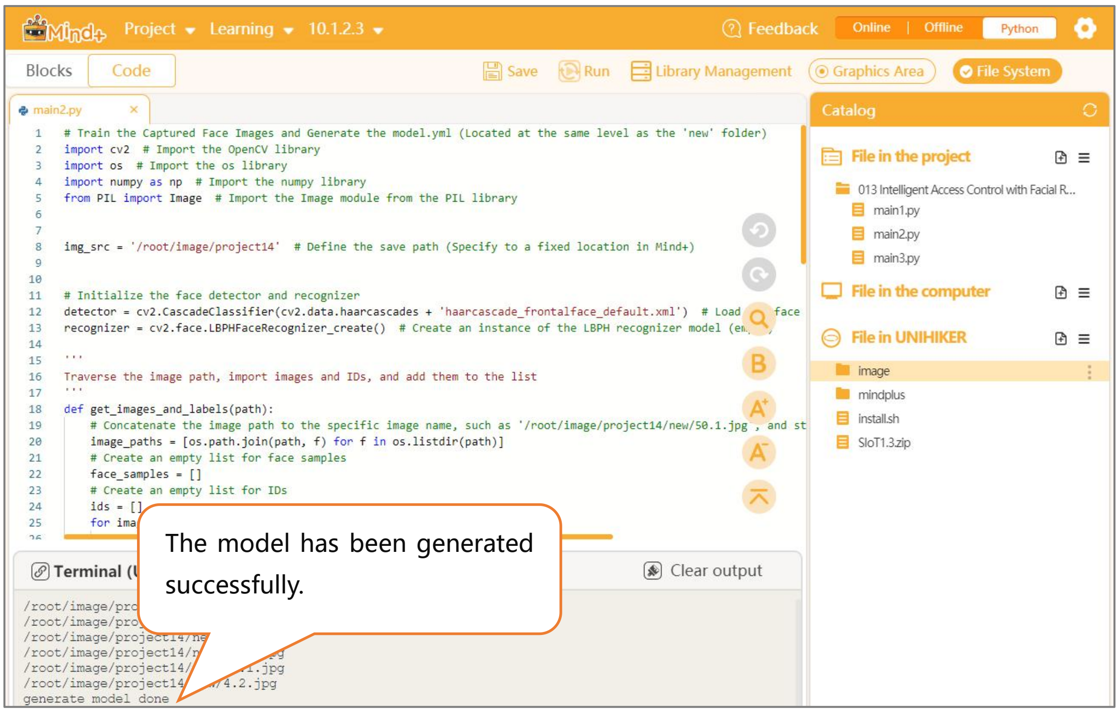
```
# Train the Captured Face Images and Generate the model.yml (Located at the same level as the 'new' folder)

import cv2  # Import the OpenCV library

import os  # Import the os library

import numpy as np  # Import the numpy library

from PIL import Image  # Import the Image module from the PIL library
```

```python
img_src = '/root/image/project14'  # Define the save path (Specify to a fixed location in Mind+)

# Initialize the face detector and recognizer
detector = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')  # Load the face detection classifier
recognizer = cv2.face.LBPHFaceRecognizer_create()  # Create an instance of the LBPH recognizer model (empty)

'''
Traverse the image path, import images and IDs, and add them to the list
'''
def get_images_and_labels(path):
    # Concatenate the image path to the specific image name, such as '/root/image/project14/new/50.1.jpg', and store it in the 'image_paths' list
    image_paths = [os.path.join(path, f) for f in os.listdir(path)]
    # Create an empty list for face samples
    face_samples = []
    # Create an empty list for IDs
    ids = []
    for image_path in image_paths:  # Traverse the image path
        # Print the name of each image (image name with path)
        print(image_path)
        # Convert the color image to grayscale
        image = Image.open(image_path).convert('L')
        # Convert the grayscale image format to a Numpy array
        image_np = np.array(image, 'uint8')
        '''To get the ID, we split the image path and retrieve the relevant information'''
        # Split by ".", if the last group is "jpg", then execute the f
ollowing steps

        if os.path.split(image_path)[-1].split(".")[-1] != 'jpg':

            continue

        # Extract the ID number from the complete path name of the ima
ge, which is the ID number we set when capturing the image

        image_id = int(os.path.split(image_path)[-1].split(".")[1])

        # Detect the face in the array format and store the results in
 'faces'

        faces = detector.detectMultiScale(image_np)  # Detect faces

        # Crop out the face portion from the array format face image a
nd store them in the face samples list, and store the ID number of the
 image in the ID samples list

        for (x, y, w, h) in faces:

            face_samples.append(image_np[y:y + h, x:x + w])

            ids.append(image_id)

    return face_samples, ids  # Return the face samples list and ID sa
mples list


# Train the LBPH recognizer model with the face samples and ID samples
, and output the corresponding face model (.yml format file)

faces, Ids = get_images_and_labels(img_src+'/new/')  # Pass in the pat
h and the folder name of the image to get the face samples list and ID
 samples list

recognizer.train(faces, np.array(Ids))  # Pass the face samples list a
nd ID samples list to the empty LBPH recognizer model to get a complet
e face model

recognizer.save(img_src+'/model.yml')  # Save the generated model

print("generate model done")  # Print "Model has been generated"
```

## 2. Running the Program

**STEP1:** Remote Connect to the UNIHIKER, Run the Program, and Observe the Results

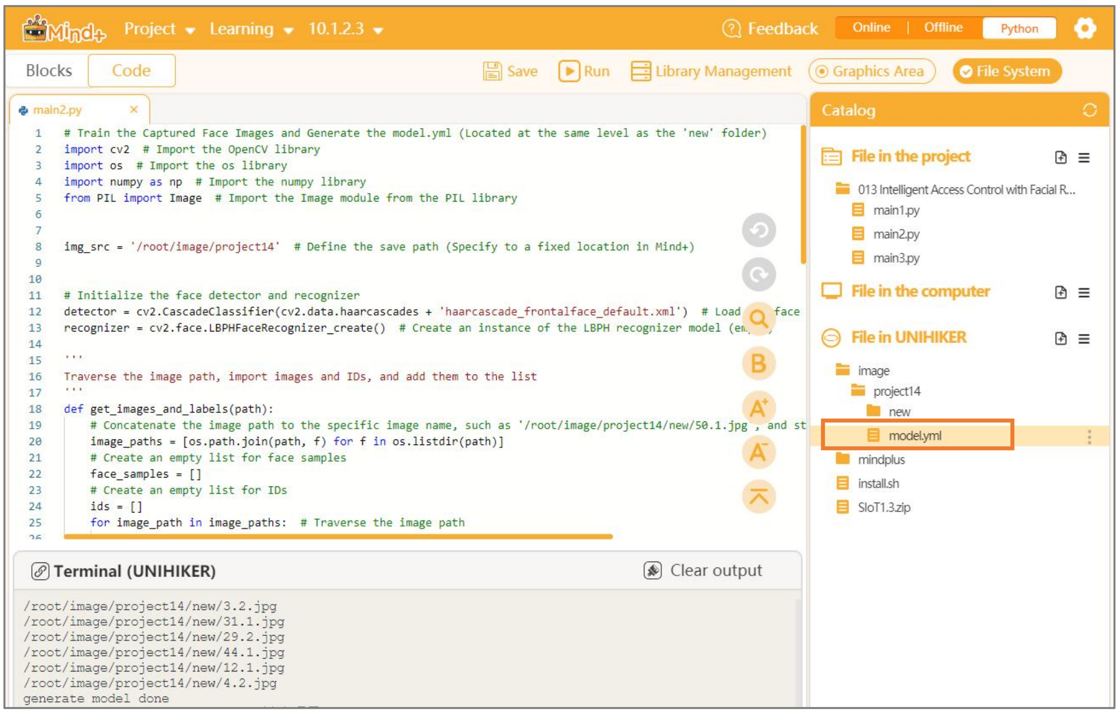
When running Mind+, you can see a series of image file names displayed in the terminal. At the end, it prints "generate model done," indicating that the model files for the three facial images mentioned above have been generated.



```python
# Train the Captured Face Images and Generate the model.yml (Located at the same level as the 'new' folder)
import cv2  # Import the OpenCV library
import os  # Import the os library
import numpy as np  # Import the numpy library
from PIL import Image  # Import the Image module from the PIL library


img_src = '/root/image/project14'  # Define the save path (Specify to a fixed location in Mind+)


# Initialize the face detector and recognizer
detector = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')  # Load face
recognizer = cv2.face.LBPHFaceRecognizer_create()  # Create an instance of the LBPH recognizer model (em

'''
Traverse the image path, import images and IDs, and add them to the list
'''
def get_images_and_labels(path):
    # Concatenate the image path to the specific image name, such as '/root/image/project14/new/50.1.jpg', and st
    image_paths = [os.path.join(path, f) for f in os.listdir(path)]
    # Create an empty list for face samples
    face_samples = []
    # Create an empty list for IDs
    ids = []
    for ima
```

```
/root/image/pro
/root/image/pro
/root/image/project14/ne
/root/image/project14/n                 pg
/root/image/project14/          1.jpg
/root/image/project14         /4.2.jpg
generate model done
```

Additionally, after reconnecting the UNIHIKER board, you can also find the model file in the "Image" folder.



```python
# Train the Captured Face Images and Generate the model.yml (Located at the same level as the 'new' folder)
import cv2  # Import the OpenCV library
import os  # Import the os library
import numpy as np  # Import the numpy library
from PIL import Image  # Import the Image module from the PIL library


img_src = '/root/image/project14'  # Define the save path (Specify to a fixed location in Mind+)


# Initialize the face detector and recognizer
detector = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')  # Load face
recognizer = cv2.face.LBPHFaceRecognizer_create()  # Create an instance of the LBPH recognizer model (en

'''
Traverse the image path, import images and IDs, and add them to the list
'''
def get_images_and_labels(path):
    # Concatenate the image path to the specific image name, such as '/root/image/project14/new/50.1.jpg', and st
    image_paths = [os.path.join(path, f) for f in os.listdir(path)]
    # Create an empty list for face samples
    face_samples = []
    # Create an empty list for IDs
    ids = []
    for image_path in image_paths:  # Traverse the image path
```

```
/root/image/project14/new/3.2.jpg
/root/image/project14/new/31.1.jpg
/root/image/project14/new/29.2.jpg
/root/image/project14/new/44.1.jpg
/root/image/project14/new/12.1.jpg
/root/image/project14/new/4.2.jpg
generate model done
```

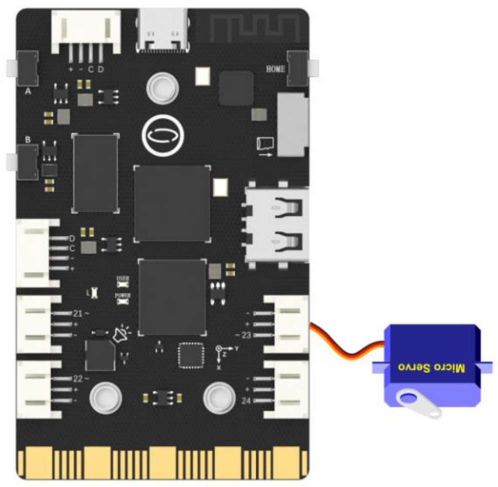
Tips: If you encounter a missing "cv2.face.LBPHFaceRecognizer_create" library file error during runtime, you can refer to Appendix 2 for offline installation of the missing library.

## Task Description 3: Real-time Face Recognition

In the previous task, we trained a model using the collected facial photos. Now, we will utilize this model to perform real-time face detection and recognition. When a recognized face matches one of the collected faces, we will control the servo motor to rotate to the 170° position to open the door. After 5 seconds, the servo motor will automatically return to the 10° position to close the door.

### 1. Hardware setup

**STEP1:** Please connect the servo motor to the P23 pin.



**STEP2：** Connect the USB camera to the UNIHIKER.

## 2. program coding

**STEP 1：** Creating and Saving Python Files

Create a Python program file named "main3.py" and double-click to open it.

**STEP 2：** Programming

(1) Importing Required Libraries

In this task, we will need to import the OpenCV library to access the camera and display the video stream, the PinPong library to control the servo motor movement, and the time library to set delays. Let's import them first.

```
import cv2  # Import the OpenCV library

from pinpong.board import Board, Pin, Servo  # Import the Pinpong libra
ry

import time  # Import the time library
```

(2) Initializing the UNIHIKER and controlling the servo motor to the initial position of 10°.

```
Board("UNIHIKER").begin()  # Initialize and select the board type, if n
ot specified, it will be automatically recognized



s1 = Servo(Pin(Pin.P23))  # Initialize the servo pin by passing it to t
he Servo class

s1.angle(10)  # Control the servo to rotate to the initial position of
```

```
10 degrees (closed door)

time.sleep(1)
```

(3) Defining the image path.

```
img_src = '/root/image/project14'  # Define the save path (Specify to a
 fixed location in Mind+)
```

(4) Setting up the camera and font type.

```
cap = cv2.VideoCapture(0)  # Open the camera with index 0 and initializ
e

cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)  # Set the buffer to 1 frame to red
uce latency

cv2.namedWindow('frame', cv2.WND_PROP_FULLSCREEN)  # Create a window wi
th the name 'frame', and set its property to fullscreen

cv2.setWindowProperty('frame', cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULL
SCREEN)  # Set the 'frame' window to fullscreen

font = cv2.FONT_HERSHEY_SIMPLEX  # Set the font type to normal-sized sa
ns-serif
```

(5) Loading the face detection classifier and LBPH recognizer.

```
detector = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_f
rontalface_default.xml')  # Load the face detection classifier

recognizer = cv2.face.LBPHFaceRecognizer_create()  # Create an instance
 of the LBPH recognizer model
```

(6) Reading the face model file.

In this task, we will utilize the trained face model mentioned above to perform real-time face detection and recognition. Therefore, we need to read the model file first.

```
recognizer.read(img_src + '/model.yml')  # Read the trained face model
from the specified path
```

(7) Real-time face recognition.

During the face detection and recognition process, we also need to adjust the camera's display so that it can be properly shown on the screen of the UniHiker. After that, we will add a functionality that, when a recognized face is detected, it will draw a rectangle around the face, display the text "Welcome Home" along with the ID number of the face, and control the servo motor to rotate to 170 degrees to simulate the opening of the door. After 5 seconds, the servo motor will return to its initial position at 10 degrees. On the other hand, when an unrecognized face is detected, it will display "unknown" and keep the servo motor at the 10-degree position to simulate the closing of the door.

Tips: Due to the processing time required for the UniHiker's CPU to display the green rectangle and text on the screen, it may be slower than the camera's frame rate. Therefore, not every frame after detecting a face will show the green rectangle and ID text. To address this, we can define a "count" variable to keep track of each frame from the camera, and only display the green rectangle and text on the third frame where a face is detected.

```python
count = 0  # Define a count flag


while True:
    ret, img = cap.read()  # Read the image frame by frame
    # img = cv2.flip(img, 1)  # Mirror the image (horizontally flip the img image)
    if ret:  # If an image is read successfully
        h, w, c = img.shape  # Record the shape of the image, including height, width, and channels
        w1 = h * 240 // 320
        x1 = (w - w1) // 2
        img = img[:, x1:x1 + w1]  # Crop the image
        img = cv2.resize(img, (240, 320))  # Resize the image to match the UNIHIKER display
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convert the image to grayscale
        faces = detector.detectMultiScale(gray, 1.2, 5)  # Detect and obtain face recognition data
        for (x, y, w, h) in faces:
            cv2.rectangle(img, (x - 50, y - 50), (x + w + 50, y + h + 50), (0, 255, 0), 2)  # Draw a rectangular box around the face
            img_id, confidence = recognizer.predict(gray[y:y + h, x:x + w])  # Predict on the grayscale image within the specified rectangular region (used for face presentation)
            if confidence < 80:  # If confidence < 80, it means a trained face is detected
                cv2.putText(img, "Welcome Home", (x, y), font, 0.6, (0, 255, 0), 2)  # Add the text "Welcome Home" to the specified position on the image
                cv2.putText(img, str(img_id), (x, y + h), font, 0.6, (0
, 255, 0), 2)  # Add the corresponding face ID label to the specified p
osition on the image

                if count == 2:  # On the third frame

                    s1.angle(170)  # Control the servo to rotate to the
 position of 170 degrees (open the door)

                    print("The door is open")

                    time.sleep(5)  # Wait for five seconds

                    s1.angle(10)  # Control the servo to rotate to the
position of 10 degrees (close the door)

                    print("The door is closed")

                    count = 0  # Reset the count

                else:

                    s1.angle(10)

                    count = count + 1  # Increment the frame count

            else:  # If an unknown face is detected

                img_id = "unknown"

                cv2.putText(img, str(img_id), (x, y + h), font, 0.6, (0
, 255, 0), 2)  # Add the text "unknown" to the specified position on th
e image

        cv2.imshow('frame', img)  # Display the image

        key = cv2.waitKey(1)  # Delay each frame by 1ms, delay cannot b
e 0, otherwise the result will be a static frame

        if key & 0xFF == ord('b'):  # Press 'b' to exit

            break
```

(8) Closing the camera and window.

```
cap.release()  # Release the camera

cv2.destroyAllWindows()  # Close all windows
```

**Tips:** The complete example program is as follows:

```
'''

Run this program to predict faces using the trained face model. When a
```

```
 trained face is detected, display the ID number and rotate the servo to 170° to open the door. When an unfamiliar face is detected, display "unknown". The more images collected, the higher the recognition accuracy.
'''
import cv2  # Import the OpenCV library
from pinpong.board import Board, Pin, Servo  # Import the Pinpong library
import time  # Import the time library

Board("UNIHIKER").begin()  # Initialize and select the board type, if not specified, it will be automatically recognized

s1 = Servo(Pin(Pin.P23))  # Initialize the servo pin by passing it to the Servo class
s1.angle(10)  # Control the servo to rotate to the initial position of 10 degrees (closed door)
time.sleep(1)

img_src = '/root/image/project14'  # Define the save path (Specify to a fixed location in Mind+)

cap = cv2.VideoCapture(0)  # Open the camera with index 0 and initialize
cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)  # Set the buffer to 1 frame to reduce latency
cv2.namedWindow('frame', cv2.WND_PROP_FULLSCREEN)  # Create a window with the name 'frame', and set its property to fullscreen
cv2.setWindowProperty('frame', cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)  # Set the 'frame' window to fullscreen
font = cv2.FONT_HERSHEY_SIMPLEX  # Set the font type to normal-sized sans-serif

'''Initialize the face detector and recognizer, and use the previously
```

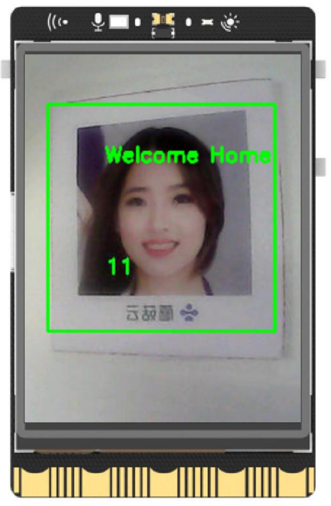
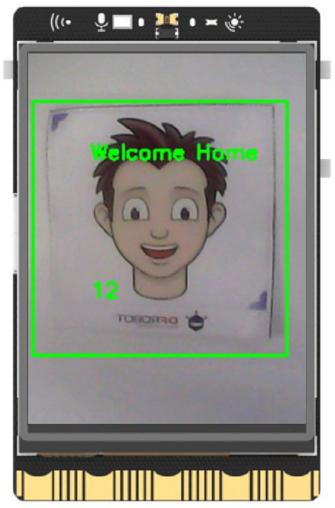
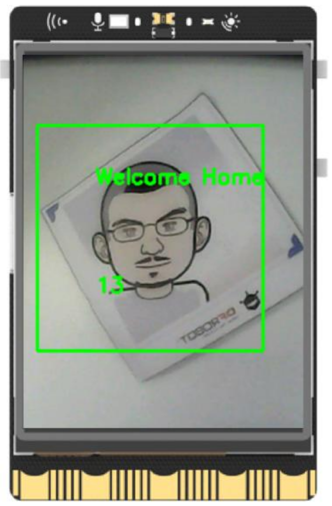
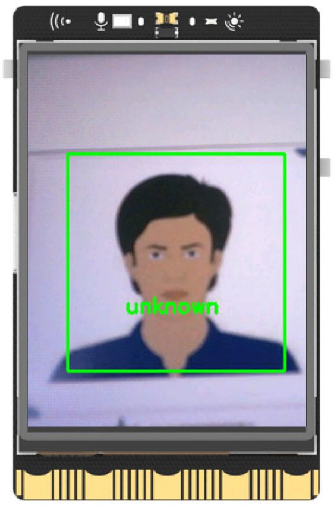
```
 trained .yml file to recognize faces'''
detector = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')  # Load the face detection classifier
recognizer = cv2.face.LBPHFaceRecognizer_create()  # Create an instance of the LBPH recognizer model
recognizer.read(img_src + '/model.yml')  # Read the trained face model from the specified path

count = 0  # Define a count flag

while True:
    ret, img = cap.read()  # Read the image frame by frame
    # img = cv2.flip(img, 1)  # Mirror the image (horizontally flip the img image)
    if ret:  # If an image is read successfully
        h, w, c = img.shape  # Record the shape of the image, including height, width, and channels
        w1 = h * 240 // 320
        x1 = (w - w1) // 2
        img = img[:, x1:x1 + w1]  # Crop the image
        img = cv2.resize(img, (240, 320))  # Resize the image to match the UNIHIKER display
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convert the image to grayscale
        faces = detector.detectMultiScale(gray, 1.2, 5)  # Detect and obtain face recognition data
        for (x, y, w, h) in faces:
            cv2.rectangle(img, (x - 50, y - 50), (x + w + 50, y + h + 50), (0, 255, 0), 2)  # Draw a rectangular box around the face
            img_id, confidence = recognizer.predict(gray[y:y + h, x:x + w])  # Predict on the grayscale image within the specified rectangular region (used for face presentation)
            if confidence < 80:  # If confidence < 80, it means a trai
```

```
ned face is detected

                cv2.putText(img, "Welcome Home", (x, y), font, 0.6, (0
, 255, 0), 2)  # Add the text "Welcome Home" to the specified position
 on the image

                cv2.putText(img, str(img_id), (x, y + h), font, 0.6, (
0, 255, 0), 2)  # Add the corresponding face ID label to the specified
 position on the image

                if count == 2:  # On the third frame

                    s1.angle(170)  # Control the servo to rotate to th
e position of 170 degrees (open the door)

                    print("The door is open")

                    time.sleep(5)  # Wait for five seconds

                    s1.angle(10)  # Control the servo to rotate to the
 position of 10 degrees (close the door)

                    print("The door is closed")

                    count = 0  # Reset the count

                else:

                    s1.angle(10)

                    count = count + 1  # Increment the frame count

            else:  # If an unknown face is detected

                img_id = "unknown"

                cv2.putText(img, str(img_id), (x, y + h), font, 0.6, (
0, 255, 0), 2)  # Add the text "unknown" to the specified position on
the image

        cv2.imshow('frame', img)  # Display the image

        key = cv2.waitKey(1)  # Delay each frame by 1ms, delay cannot
be 0, otherwise the result will be a static frame

        if key & 0xFF == ord('b'):  # Press 'b' to exit

            break


cap.release()  # Release the camera

cv2.destroyAllWindows()  # Close all windows
```

## 3. Running the Program

**STEP1:** Remote Connect to the UNIHIKER, Run the Program, and Observe the Results

Observing the UniHiker, once the camera display is shown, we can align it with the faces on the 3 captured cards. We will notice that each face on the image is surrounded by a rectangle, displaying the corresponding ID number and the text "Welcome Home". At this point, the image remains static, and the servo motor rotates from its original position of 10 degrees to 170 degrees. After 5 seconds, it returns to the initial position of 10 degrees. Afterwards, if we align it with an unrecognized face, the text "unknown" will be displayed on the face, while the servo motor remains at the initial position of 10 degrees.

| | |
|---|---|
|  ID: 11 |  ID: 12 |
|  ID: 13 |  ID: unknown |

**Tips 1:** The more images we capture, the higher the accuracy of the recognition. You can adjust the recognition rate by changing the threshold value "80" in the program.

**Tips 2:** When detecting familiar faces, it is advisable to maintain a consistent environment with the one during image capture. This can improve the recognition rate.

## Challenge Yourself

Think about it, what are some areas that you think are worth further improving for the current access control system?

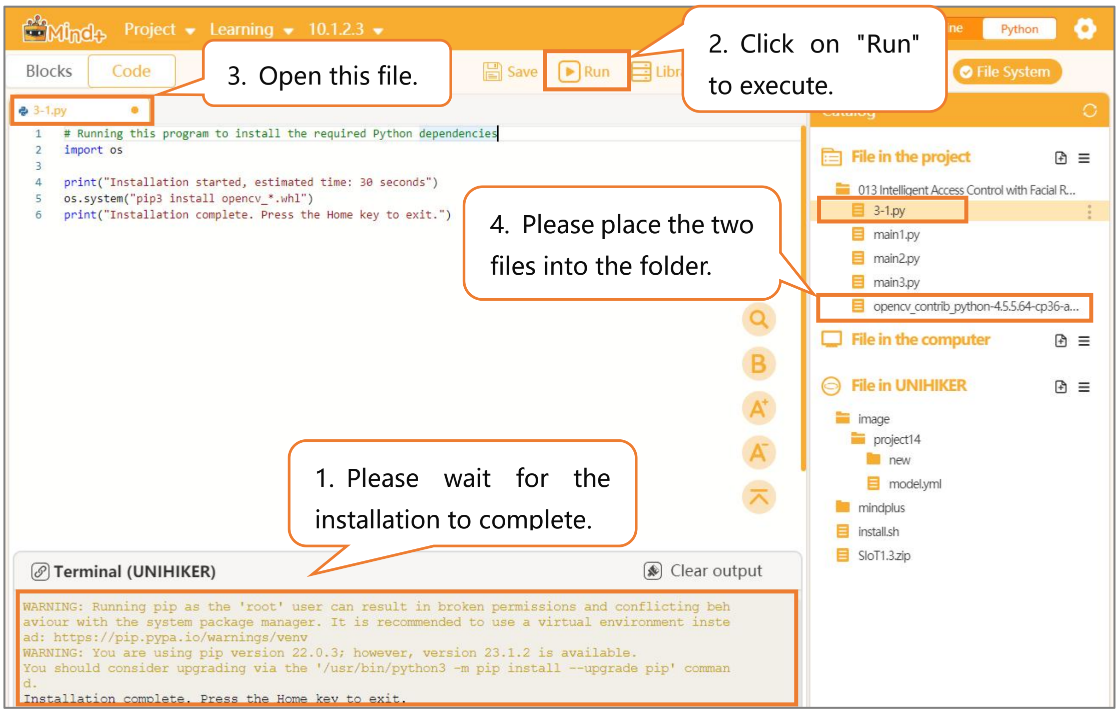
## Appendix

Operation Instructions:

**STEP 1:** Upgrade Mind+ to the latest version (V1.7.2 RC3.0 or above).

**STEP 2:** Download the "3-1.py" file and the "opencv_contrib_python-4.5.5.64-cp36-abi3-manylinux_2_17_aarch64.manylinux2014_aarch64.whl" file from the link provided in Appendix 1.

**STEP 3:** Install the library files.



```python
# Running this program to install the required Python dependencies
import os

print("Installation started, estimated time: 30 seconds")
os.system("pip3 install opencv_*.whl")
print("Installation complete. Press the Home key to exit.")
```

```
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting beh
aviour with the system package manager. It is recommended to use a virtual environment inste
ad: https://pip.pypa.io/warnings/venv
WARNING: You are using pip version 22.0.3; however, version 23.1.2 is available.
You should consider upgrading via the '/usr/bin/python3 -m pip install --upgrade pip' comman
d.
Installation complete. Press the Home key to exit.
```