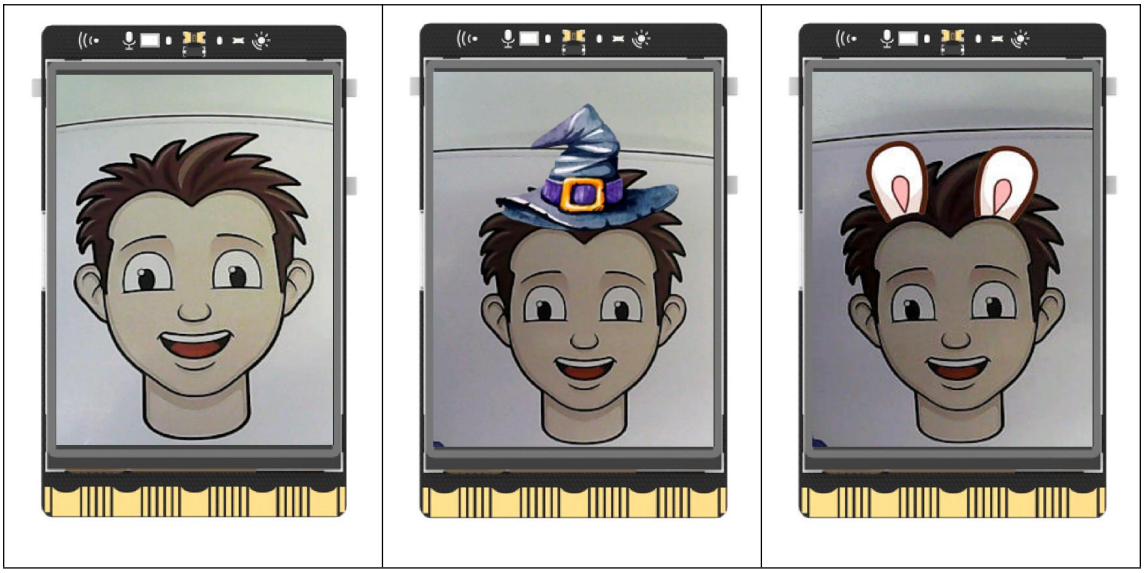
Lesson 12 Magic Face Mask for Face Detection

Selfie is a hobby for many people, and to make the photos more lively and fun, we often add filter masks to our selfies.

Do you know how this is achieved? In this lesson, let's combine the concept of "UNIHIKER" and a camera to simulate the experience of this feature!

Task Objectives

By using a USB camera, we can display the live feed in real-time. When a face is detected in the frame, we can add special effects images to it.



Knowledge points

1. Understand face detection.

2. Learn how to use the opencv library to access the camera and display real-time video streams.

3. Learn how to use the opencv library for face detection.

4. Learn how to use the Image module in the PIL library to convert image formats.

5. Learn how to use the Image module in the PIL library to paste images onto an image.

Material List

Hardware List:

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

Software Preparation: Mind+ Programming Softwarex1

Knowledge background

1. What is face detection?

Face detection refers to the process of searching for and determining the presence of faces in a given image using specific strategies. It involves identifying the location, size, and orientation of the detected faces. In simple terms, face detection is the process of finding regions in an image that contain faces, answering the question of "whether or not there is a face" in the image.

2. What is a cascade classifier?

In face detection, classifiers are often used to determine whether an object belongs to a certain category. A classifier is a device or tool that discriminates whether something belongs to a specific class or category. A cascade classifier can be understood as a series of N individual classifiers that are connected in a cascade. If an object satisfies all the classifiers in the cascade, then the final result is considered positive. For example, in the case of face detection, various facial attributes can be modeled as individual classifiers. If a model satisfies all the defined facial attributes (such as two eyebrows, two eyes, a nose, a mouth, a roughly U-shaped chin, or facial contours), it is considered a face.

In the opencv library, there are pre-trained cascade classifiers available for detecting faces, facial features (such as eyes and nose), humans, and other objects.

3. Opening the camera with the "VideoCapture()" function in the opencv library

The "VideoCapture()" function in the opencv library is used to open the camera and initialize it for displaying real-time video streams. Before using this function, it is necessary to import the opencv library. Additionally, specific parameter identifiers are required to indicate which camera to open. Using the value "-1" will randomly select a camera if multiple cameras are available. If there are multiple cameras, using the numbers "0" and "1" will represent the first and second cameras, respectively. This pattern continues for additional cameras. If only one camera is available, both "0" and "-1" can be used as the camera ID.

```
import cv2 # Import the opencv library

cap = cv2.VideoCapture(0) # Open usb camera 0.
```

In this case, "cap" is an object created for the camera.

4. Setting camera properties with the "set()" function in the opencv library

After opening the camera, we can use the "set()" method in opencv to configure various camera properties. For example, we can set the buffer time, which can be understood as the screen delay when displaying the video stream. A higher value for the buffer time will result in a longer delay in displaying the video on the screen.

```
cap.set(cv2.CAP_PROP_BUFFERSIZE, 1) # Set the camera buffer to 1, to decrease th

e latency.
```

In this case, "cv2.CAP_PROP_BUFFERSIZE" refers to the buffer size property, and the value "1" represents 1 frame.

5. Checking if the camera is successfully opened with the "isOpened()" function in the opencv library

After opening the camera, we can use the "isOpened()" method to check if the camera is successfully opened. If the camera is opened successfully, it will return True, otherwise it will return False.

```
while(cap.isOpened()):  # When the camera is open
```

6. Reading images with the "read()" function in the opencv library

After successfully opening the camera, we can use the "read()" method to read images frame by frame. By continuously and quickly reading images, we can achieve a real-time video stream effect.

```
while(cap.isOpened()):  # When the camera is open

    ret, img = cap.read() # Read one frame from the USB camera.
```

In this case, "ret" and "img" are two variables. The former is used to store whether an image is successfully read, returning True if an image is read and False otherwise. The latter is used to store the actual image that is read.

7. Resizing images with the "resize()" function in the opencv library

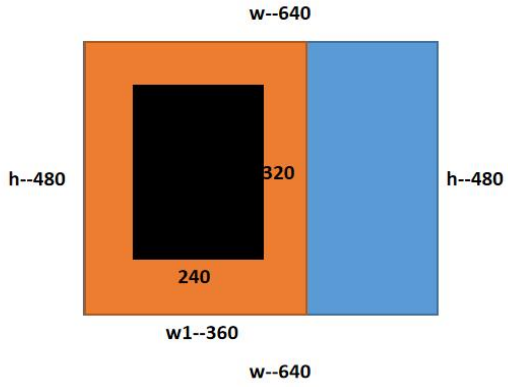
When an image has an inappropriate size, we can use the "resize()" function in the opencv library to adjust its size by scaling it.

```
# Read one frame from the USB camera.

    ret, img = cap.read()

    # If frame available.

    if ret:

        '''Crop the center of the frame and resize to (240, 320) while maintaining the i
mage ratio.'''

        '''# Crop the center of the frame and resize it to the same ratio as "UNIHIKER"
 screen. # Display the image captured by the camera on the screen in the same rat
io as "UNIHIKER" screen.'''

        h, w, c = img.shape  # Record the shape of the image, which includes 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))  # Adjust the image size.
```

In this case, the "img" in the first line refers to the original image read from the camera, and "img2" in the last line refers to the resized image. "(240, 320)" refers to the pixel values for width and height. Here, we are adjusting the size of the image to match the "行空板" so that it can be displayed on the screen.

Additionally, since the resolution of the image captured by the camera is 640x480, and resizing can only be done proportionally, we need to crop the original image first. As shown in the diagram, the original image is 640x480, and after cropping, we obtain an orange rectangle with a size of 360x480, which is proportional to 240x320. During cropping, the default base point is the top-left corner, while during scaling, the default base point is the center of the rectangle.



8. Closing the camera with the "release()" function in the opencv library

After opening the camera, it is necessary to release it when finished. This can be achieved using the "release()" method in the opencv library.

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

9. Loading the face detection classifier with the "CascadeClassifier()" function in the opencv library

The opencv library provides pre-trained face detection classifiers, which are stored as XML files in the "data" folder located in the root directory of opencv. This folder contains three subfolders: "haarcascades", "hogcascades", and "lbpcascades", which store the Harr, HOG, and LBP cascade classifiers, respectively. By loading the XML files of different cascade classifiers, we can detect different objects.

For example, in the Harr cascade classifier, there is a specific XML file named "haarcascade_frontalface_default.xml" used for frontal face detection. To use it, we can utilize the "CascadeClassifier()" function and provide the path and the specific classifier name as parameters.

```
# Load face recognition model.

faceCascade=cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalfac

e_default.xml')
```

In this case, "haarcascade_frontalface_default.xml" refers to the specific cascade classifier used for detecting frontal faces. The preceding "cv2.data.haarcascades" is the path where it is located, corresponding to the "haarcascades" subfolder in the "data" folder of the opencv library.

10. Detecting Faces with the "detectMultiScale()" Function in the opencv Library

After loading the face detection classifier, the "detectMultiScale()" method can be used to detect all faces in an image. It returns the coordinates and size (using the parameters x, y, w, and h) of each detected face.

```
faces = faceCascade.detectMultiScale(

    gray,  # The image to be detected

    scaleFactor=1.1, # The scale factor by which the image size is reduced each tim

e, default is 1.1

    minNeighbors=5,  # Represents the minimum number of times a target must b

e detected to be considered a valid target (since different window sizes and surro

unding pixels can detect faces)

    minSize=(30, 30) # The minimum size of the target

)
```

In this case, "gray" refers to the grayscale image to be detected, "scaleFactor" is the scaling factor, "minNeighbors" represents the minimum number of times a target must be detected, usually chosen as 3-5, and "minSize" represents the minimum size of the face targets in the image. Here, we set the minimum width and height both to 30 pixels.

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

The Image module in the PIL library is a Python module that allows for image processing. It offers a wide range of functions, but here we will provide a brief introduction. Before using it, the module needs to be imported.

(1) Converting an Array Format Image to an Image Format Image with the "fromarray()" Function

Since opencv processes images in array format, while the special effect image for pasting needs to be in Image format, we need to use the "fromarray()" function to convert the image from array format to Image format before applying any effects.

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

ret, img = cap.read()

img = Image.fromarray(img) # Convert from opencv Mat to Pillow Image.
```

(2) Pasting Images with the "paste()" Function

The "paste()" function in the Image module of the PIL library allows you to overlay and paste one image onto another.

img.paste(mark, (x, y-h1), mark) # Paste the special effect image onto the camera image

Here, "img" refers to the original image onto which we want to paste, the first "mark" refers to the special effect image used for pasting, "(x, y-h1)" indicates the position on the image where the top-left corner of the pasted image should be, and the third "mark" refers to the image used for masking during the pasting process (in this case, it is the same image). By using this method, we can paste an image onto the original image with transparency. Alternatively, we can write it as "img.paste(mark, (x, y-h1), mask=mark)".

Hands-on practice

Task Description 1: Display Real-Time Video Stream

To display the live video stream captured by the camera on the screen.

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 "012 Magic Face Mask for Face Detection".

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 need to use the opencv library to access the camera and display the video stream. Therefore, we need to import it first.

```
import cv2  # Import the opencv library
```

(2) Configuring the Camera

To display the real-time video feed from the camera 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 prevent any stuttering or high latency. Next, we create a window and, finally, set it to full screen mode to display the video feed on the window.

```
# Open USB camera 0.

cap = cv2.VideoCapture(0)

# Set the camera buffer size to 1 to decrease latency.

cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)

# Create a window named 'winname' with default properties that allow fullscreen.

cv2.namedWindow('winname',cv2.WND_PROP_FULLSCREEN)

# Set the 'winname' window to fullscreen.

cv2.setWindowProperty('winname', cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_F
```

```
ULLSCREEN)
```

(1) Displaying the Video Stream

After setting up the window, we can proceed to program the display of the video stream. Here, once the camera is opened, we start by reading the frames of the video. If a frame is successfully read, we first record the dimensions of the obtained image. Then, we proceed to crop the image and resize it to match the screen size of the UNIHIKER display. Finally, we set up the display of the processed image on the window and refresh it every 1ms to create a video stream. Additionally, we also set up the program to exit when the 'A' key is pressed.

```
while (cap.isOpened()):  # When the camera is open.
    # Read one frame from the USB camera. ret returns True or False, indicating wh
ether an image is read, img is the frame read.
    ret, img = cap.read()
    # If frame available.
    if ret:
        '''Crop the center of the frame and resize to (240, 320) while maintaining ima
ge ratio.'''
        h, w, c = img.shape  # Record the shape of the image, with height, width, and
 channels.
        w1 = h*240//320  # Change the height to fit the render image.
        x1 = (w-w1)//2  # Midpoint of width without resizing.
        img = img[:, x1:x1+w1]  # Crop the image to the center.
        img = cv2.resize(img, (240, 320))  # Resize the image according to the screen,
```

```
keeping the aspect ratio.



    '''Display the real-time video stream.'''

    cv2.imshow('winname', img)  # Display the image on the 'winname' window.

    key = cv2.waitKey(1)  # Refresh the image every 1ms. Delay cannot be 0, othe
rwise the result will be a static frame.

    '''Press 'a' key to exit the program.'''

    if key & 0xFF == ord('a'):

        break
```

(1) Closing the Camera and Windows

After pressing the 'A' key to exit, we release the camera and close all image windows.

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

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

Tips: The complete example program is as follows:

```
import cv2  # Import the opencv library



# Open USB camera 0.

cap = cv2.VideoCapture(0)
```

```
# Set the camera buffer size to 1 to decrease latency.

cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)

# Create a window named 'winname' with default properties that allow fullscreen

.

cv2.namedWindow('winname',cv2.WND_PROP_FULLSCREEN)

# Set the 'winname' window to fullscreen.

cv2.setWindowProperty('winname', cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_

FULLSCREEN)


while (cap.isOpened()):  # When the camera is open.

    # Read one frame from the USB camera. ret returns True or False, indicating w

hether an image is read, img is the frame read.

    ret, img = cap.read()

    # If frame available.

    if ret:

        '''Crop the center of the frame and resize to (240, 320) while maintaining im

age ratio.'''

        h, w, c = img.shape  # Record the shape of the image, with height, width, an

d channels.
```

```
        w1 = h*240//320  # Change the height to fit the render image.
        x1 = (w-w1)//2  # Midpoint of width without resizing.
        img = img[:, x1:x1+w1]  # Crop the image to the center.
        img = cv2.resize(img, (240, 320))  # Resize the image according to the scree
n, keeping the aspect ratio.

        '''Display the real-time video stream.'''
        cv2.imshow('winname', img)  # Display the image on the 'winname' window.
        key = cv2.waitKey(1)  # Refresh the image every 1ms. Delay cannot be 0, oth
erwise the result will be a static frame.
        '''Press 'a' key to exit the program.'''
        if key & 0xFF == ord('a'):
            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

By observing the UniHiker board, it can be noticed that the real-time captured images from the camera are displayed in full screen on the board. As shown in the image below, when the camera is pointed towards a face, the corresponding image can be seen on the screen.

Task Description 2: Face Detection and Adding Special Effects Images

In the previous task, we successfully displayed the live video stream captured by the camera on the UNIHIKER. Now, we will enhance the functionality by implementing face detection and adding filter overlays (special effects images) on top of detected faces.

1. program coding

STEP 1: Creating and Saving Python Files

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

STEP 2: Import image folder

Import the sticker images 1.png and 2.png from the "img" folder into the project folder.

STEP 3: Programming

(1) Importing the numpy Library and the Image Module from the PIL Library
In this task, we will be using the numpy library and the Image module from the PIL library to process images. Therefore, we need to add the following imports to include them:

```
import numpy as np # Import the numpy library

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

(2) Loading the Face Detection Classifier

Since we will be adding special effects to the faces in the camera feed, we need to perform face

detection first. Fortunately, the opencv library already provides pre-trained face detection classifiers, so we just need to load them to use.

```
# Load face recognition model.

faceCascade=cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalfac

e_default.xml')
```

(3) Reading the Special Effects Image

To add special effects after detecting a face, we need to first read the provided special effects image.

```
ImageID = 1

# Read the mark image file from disk.

markOrigin = Image.fromarray(cv2.imread(str(ImageID) + ".png" , cv2.IMREAD_UN

CHANGED))
```

(4) Define Function - Get Face and Add Special Effects Image

Here, due to the complexity of detecting faces and adding special effects in a video stream, we will encapsulate this process into a function.

Firstly, since the image captured by the camera is in three channels, we convert it to grayscale (one channel) to simplify the image processing. Then, we perform face detection on the grayscale image to obtain face recognition data. Finally, when a face is detected, we add the special effects image. In the face detection process, we set four parameters: the grayscale image to be detected, the default scale reduction ratio for the image size, the minimum number of detections for each target, and the minimum size of the target.

```
# get and render face with mark.

def getAndRenderFace(img):

    # Face recognition data.

    # Convert the image img from BGR color space to HSV color space and name th
```

```
e new image as gray (convert to grayscale to reduce dimensionality and complexit
y of the image).
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Detect and obtain face recognition data.
    faces = faceCascade.detectMultiScale(
        gray,  # Image to be detected
        scaleFactor=1.1,  # Scale factor by which the image size is reduced each time,
 default is 1.1
        minNeighbors=5,  # Minimum number of times a target must be detected to
consider it a true target (since faces can be detected with different window sizes a
nd surrounding pixels)
        minSize=(30, 30)  # Minimum size of the target
    )
    for (x, y, w, h) in faces:
        # img = cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)  # Draw face recognitio
n data.
        img = drawMark(img, x, y, w, h)  # Add special effects to the image based on
face recognition data.
    return img
```

(5) Define Function - Add Special Effects Image to Video Stream

Due to the complexity of adding special effects images, we will also encapsulate this process into a function.

Here, we need to process the special effects image to be displayed, which can be done in three steps. First, we obtain the dimensions of the special effects image. Then, we proportionally adjust the size of the image to fit the screen. Finally, we convert the image from array format to Image format.

After processing the special effects image, we paste it onto the image captured by the camera. We then convert the final complete image into an array format to display it as a video stream in the window of the board.

```
# Define a function - Add avatar functionality

def drawMark(img, x, y, w, h):

    # Get the size of the mark image.

    wm, hm = markOrigin.size

    # Keep the ratio and resize the image to fit the width of the face.

    h1 = w * hm // wm

    # Resize.

    mark = markOrigin.resize((w, h1))

    # Convert from opencv Mat to Pillow Image.

    img = Image.fromarray(img)

    # Paste the special effects image onto the image from the camera.

    img.paste(mark, (x, y - h1), mark)

    # Convert from Pillow Image to opencv Mat.
```

```
    img = np.array(img)

    return img
```

(6) Call the Function to Achieve the Effect

Finally, we will add the function call to detect faces and add special effects images within a loop. Additionally, we will add the functionality to switch between different special effects images by pressing the "B" key.

```
while(cap.isOpened()):  # When the camera is open

    # Read one frame from the USB camera.

    ret, img = cap.read()

    # If frame available.

    if ret:

        '''Crop the center of the frame and resize to (240, 320) while maintaining the i
mage ratio.'''

        '''# Crop the center of the frame and resize it to the same ratio as "UNIHIKER"
 screen. # Display the image captured by the camera on the screen in the same rat
io as "UNIHIKER" screen.'''

        h, w, c = img.shape  # Record the shape of the image, which includes 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))  # Adjust the image size.



        '''Real-time recognition and adding avatar.'''

        img = getAndRenderFace(img)



        '''Display real-time video stream.'''

        cv2.imshow('winname', img)  # Display the image on the "winname" window.

        key = cv2.waitKey(1)  # Refresh the image every 1ms. Delay cannot be 0, othe
rwise the result will be a static frame.



        '''Press the "a" key to exit the program, press the "b" key to switch the mark.''
'


        if key & 0xFF == ord('a'):  # Press the "a" key to exit the video.

            print("Exiting video")

            break

        elif key & 0xFF == ord('b'):  # Press the "b" key to change the mark.

            ImageID += 1  # Change the ImageID on every "a" click.

            if ImageID > 2:  # Two marks are available in this program.
```

```
            ImageID = 1

        # Reload the mark from disk.

        markOrigin = Image.fromarray(cv2.imread(str(ImageID) + ".png", cv2.IMRE
AD_UNCHANGED))

    else:

        break
```

Tips: The complete example program is as follows:

```
import numpy as np # Import the numpy library

import cv2 # Import the opencv library

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



# Open usb camera 0.

cap = cv2.VideoCapture(0)

# Set the camera buffer to 1, to decrease the latency.

cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)

# Set the windows to be full screen.

cv2.namedWindow('winname',cv2.WND_PROP_FULLSCREEN)
```

```
# Set the windows to be full screen.

cv2.setWindowProperty('winname', cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_
FULLSCREEN)



# Load face recognition model.

faceCascade=cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalfa
ce_default.xml')

# Name of the png file like 1.png 2.png ...

ImageID = 1

# Read the mark image file from disk.

markOrigin = Image.fromarray(cv2.imread(str(ImageID) + ".png" , cv2.IMREAD_U
NCHANGED))



# get and render face with mark.

def getAndRenderFace(img):

    # Face recognition data.

    # Convert the image img from BGR color space to HSV color space and name t
he new image as gray (convert to grayscale to reduce dimensionality and comple
```

```
xity of the image).
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Detect and obtain face recognition data.
    faces = faceCascade.detectMultiScale(
        gray,  # Image to be detected
        scaleFactor=1.1,  # Scale factor by which the image size is reduced each tim
e, default is 1.1
        minNeighbors=5,  # Minimum number of times a target must be detected t
o consider it a true target (since faces can be detected with different window size
s and surrounding pixels)
        minSize=(30, 30)  # Minimum size of the target
    )
    for (x, y, w, h) in faces:
        # img = cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)  # Draw face recogniti
on data.
        img = drawMark(img, x, y, w, h)  # Add special effects to the image based o
n face recognition data.
    return img
```

```
# Define a function - Add avatar functionality

def drawMark(img, x, y, w, h):

    # Get the size of the mark image.

    wm, hm = markOrigin.size

    # Keep the ratio and resize the image to fit the width of the face.

    h1 = w * hm // wm

    # Resize.

    mark = markOrigin.resize((w, h1))

    # Convert from opencv Mat to Pillow Image.

    img = Image.fromarray(img)

    # Paste the special effects image onto the image from the camera.

    img.paste(mark, (x, y - h1), mark)

    # Convert from Pillow Image to opencv Mat.

    img = np.array(img)

    return img


while(cap.isOpened()):  # When the camera is open

    # Read one frame from the USB camera.
```

```
    ret, img = cap.read()

    # If frame available.

    if ret:

        '''Crop the center of the frame and resize to (240, 320) while maintaining the
 image ratio.'''

        '''# Crop the center of the frame and resize it to the same ratio as "UNIHIKE
R" screen. # Display the image captured by the camera on the screen in the same
 ratio as "UNIHIKER" screen.'''

        h, w, c = img.shape  # Record the shape of the image, which includes 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))  # Adjust the image size.


        '''Real-time recognition and adding avatar.'''

        img = getAndRenderFace(img)


        '''Display real-time video stream.'''
```

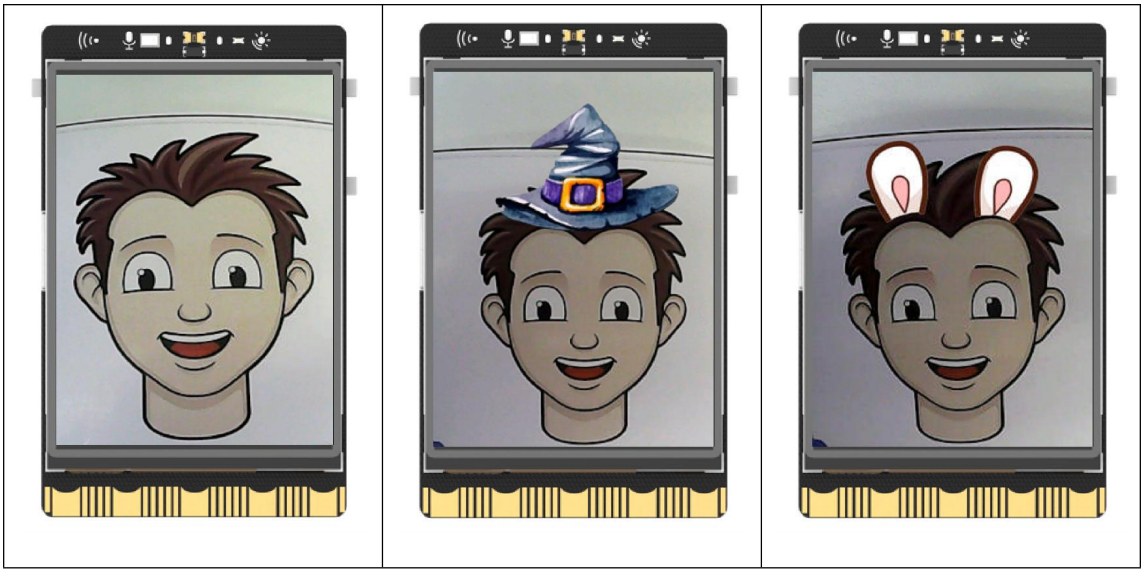
```
        cv2.imshow('winname', img)  # Display the image on the "winname" window
.

        key = cv2.waitKey(1)  # Refresh the image every 1ms. Delay cannot be 0, oth
erwise the result will be a static frame.


        '''Press the "a" key to exit the program, press the "b" key to switch the mark.
'''

        if key & 0xFF == ord('a'):  # Press the "a" key to exit the video.

            print("Exiting video")

            break

        elif key & 0xFF == ord('b'):  # Press the "b" key to change the mark.

            ImageID += 1  # Change the ImageID on every "a" click.

            if ImageID > 2:  # Two marks are available in this program.

                ImageID = 1

            # Reload the mark from disk.

            markOrigin = Image.fromarray(cv2.imread(str(ImageID) + ".png", cv2.IMR
EAD_UNCHANGED))

    else:

        break
```

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

cv2.destroyAllWindows()  # Close all windows created by opencv.
```

2. Running the Program

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

Upon observing the UNIHIKER, it can be observed that when the camera detects a face, a magical hat promptly materializes atop the person's head. Upon pressing the onboard button "b" to toggle, upon subsequent face detection, the magical hat transforms into adorable bunny ears.



Challenge Yourself

Feel free to give it a try yourself and explore a plethora of amusing and captivating magical image transformations! Let your imagination run wild and indulge in a delightful array of enchanting visual experiences!