

Lesson 15 Implement Object, Color & Gesture Recognition with OpenCV

15.1 Overview

In this lesson, we will learn how to use Raspberry Pi, Adeept Robot HAT V3.2, and USB camera module, combined with OpenCV library, to achieve motion capture, color recognition, and gesture recognition functions. Through practical operation, master the relevant technical principles and be able to apply these technologies in practical projects, such as implementing visual based interactive control in scenarios such as smart cars.

15.2 Required Components

Components	Quantity	Picture
Raspberry Pi	1	
Adeept Robot HAT V3.2	1	
Camera Module	1	
Camera Cable	1	

15.3 Principle Introduction

In this project, the implementation of motion capture, color recognition, and gesture recognition functions relies on various technical principles. The following will provide a detailed introduction to these principles.

Motion Capture

The motion capture function utilizes OpenCV's Haar cascade classifier or deep learning models (such as YOLO, SSD, etc.) to achieve object recognition.

The working method of Haar cascade classifier is to first learn from massive sample images, extract object features from them, and store them in the form of Haar features. In the actual recognition stage, it will match the images captured by the camera with the learned Haar features. The specific method is to use sliding windows to traverse the image one by one, analyze each window area, and determine whether there is a target object in it. The advantage of this method lies in its relatively simple calculation, but it has a strong dependence on samples and limited adaptability to complex backgrounds and lighting changes.

Deep learning models are trained on a large amount of annotated data and have the ability to automatically learn complex feature representations of objects. Taking the YOLO (You Only Look Once) model as an example, it transforms object detection tasks into a regression problem that can directly predict the category and position of objects on images. The biggest feature of this model is its fast speed, which can achieve real-time detection while ensuring a certain level of accuracy, making it suitable for scenarios with high real-time requirements.

SSD (Single Shot MultiBox Detector) is also based on deep learning technology, which performs object detection on feature maps of different scales. This multi-scale detection mechanism enables it to effectively detect objects of different sizes, demonstrating good detection performance in complex scenes and meeting diverse object detection needs.

Color Recognition

Video color recognition is a technology for detecting and analyzing specific colors in a video stream, and its principles are as follows:

Color Space Conversion: Video images are mostly stored and displayed in the RGB color space. However, for color recognition, they are often converted into color spaces such as HSV that are more conducive to color processing. For example, the characteristics of hue, saturation, and value

in HSV are more in line with human perception of colors, and it has better robustness to changes in lighting conditions.

Color Feature Extraction: Process the video frames in the selected color space and extract color features. For instance, in the HSV color space, colors are determined by setting specific value ranges, or a color histogram is calculated to reflect the probability distribution of colors.

Threshold Setting and Matching: Set a threshold range according to the target color, and match the extracted color features with it to determine whether it is the target color. Also, the threshold needs to be adjusted and optimized according to the actual situation.

Time Series Analysis: Since a video is composed of consecutive frames, analyze the color changes in adjacent frames to determine dynamic information such as the appearance, disappearance, and movement of colors, which improves the accuracy and reliability of recognition.

Assistance of Machine Learning and Deep Learning: Use algorithms such as Support Vector Machines (SVM) and Convolutional Neural Networks (CNN). Train the model with a large amount of annotated image data to learn the pattern of color features and handle complex color recognition tasks.

Gesture Recognition

Gesture recognition begins with a skin color detection algorithm to extract the hand area from video frames. Commonly used skin color detection methods are based on color models, leveraging the distribution characteristics of skin color in specific color spaces (such as the YCbCr color space or the HSV color space). By setting thresholds, the skin color area is filtered out.

Then, a contour detection algorithm is used to obtain the hand contour. In OpenCV, the `findContours` function can be used to find contours in the image. The convex hull of the contour is calculated. The convex hull is the smallest convex polygon that encloses the contour, which can be achieved using the `convexHull` function.

Finally, different gestures are recognized by detecting the number and position of convex defects in the convex hull. For example, when making a fist, the number of convex defects in the hand contour is relatively small, while when opening the palm, the number is larger. Based on these characteristic differences, different gesture actions can be determined.

15.4 Demonstration

1. **Remotely log:** Remotely log in to the Raspberry Pi terminal.

2. **Navigate to the Program Folder:** Enter the following command in the terminal and press **Enter** to access the folder where the program is located:

```
cd Adeept_AWR-V3/Examples/09_OpenCV/
```

```
pi@raspberrypi:~ $ cd Adeept_AWR-V3/Examples/09_OpenCV/  
pi@raspberrypi:~/Adeept_AWR-V3/Examples/09_OpenCV $
```

3. **View Directory Contents:** Type "ls" in the terminal and press **Enter**. This will display all the files in the current directory, ensuring that the "**Camera_FindColor.py**", "**Camera_Gesture.py**" and "**Camera_WatchDog.py**" file is present:

```
ls
```

```
pi@raspberrypi:~/Adeept_AWR-V3/Examples/09_OpenCV $ ls  
base_camera.py Camera_FindColor.py Camera_Gesture.py Camera_WatchDog.py templates
```

4. **Run the Program:** Enter the command below and press Enter to start the **Camera_WatchDog.py** program:

```
sudo python3 Camera_WatchDog.py
```

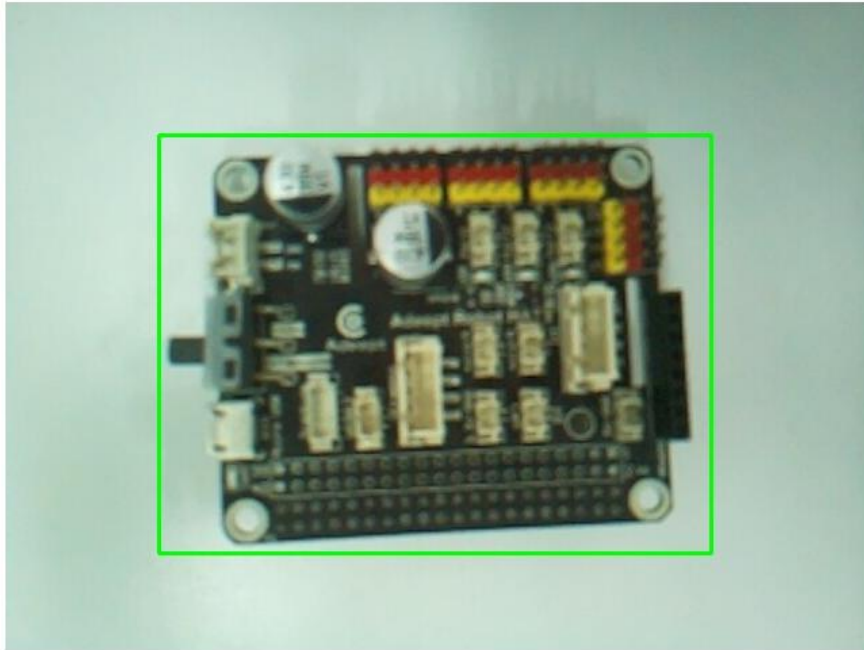
```
pi@raspberrypi:~/Adeept_AWR-V3/Examples/09_OpenCV $ sudo python3 Camera_WatchDog.py  
* Serving Flask app 'Camera_WatchDog'  
* Debug mode: off  
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
* Running on all addresses (0.0.0.0)  
* Running on http://127.0.0.1:5000  
* Running on http://192.168.3.130:5000  
Press CTRL+C to quit
```

5. Open a web browser (here we use Chrome as an example) on a device on the same LAN of the Raspberry Pi, enter in the address bar the Raspberry Pi's IP address and the video stream port number ":5000", as shown below:

Example: <http://192.168.3.130:5000>

6. Now, you can view web pages created by Raspberry Pi on your phone or computer. After successfully loading the data, make an action in front of the camera and you should be able to see the detected action area in the video stream marked by a green rectangle. and when you want to terminate the running program, you can press the "**Ctrl+C**" shortcut key on the keyboard.

Video Streaming Demonstration



7. **Run the Program:** Enter the command below and press **Enter** to start the **Camera_Gesture.py** program:

```
sudo python3 Camera_Gesture.py
```

```
pi@raspberrypi:~/Adeept_AWR-V3/Examples/09_OpenCV $ sudo python3 Camera_Gesture.py
* Serving Flask app 'Camera_Gesture'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.3.130:5000
Press CTRL+C to quit
```

8. Open a web browser (here we use Chrome as an example) on a device on the same LAN of the Raspberry Pi, enter in the address bar the Raspberry Pi's IP address and the video stream port number ":5000", as shown below:

Example: <http://192.168.3.130:5000>

9. Use a Raspberry Pi camera to capture video streams. In each frame, detect the hand skin area, recognize gestures (fist - clenching or palm - opening) by the number of convex defects in the hand, and display the results in real - time on the frames. Then, stream the processed frames to a webpage for viewing. and when you want to terminate the running program, you can press the "**Ctrl+C**" shortcut key on the keyboard.

Video Streaming Demonstration



Video Streaming Demonstration



10. **Run the Program:** Enter the command below and press **Enter** to start the **Camera_FindColor.py** program:

```
sudo python3 Camera_FindColor.py
```

```
pi@raspberrypi:~/Adept_AWR-V3/Examples/09_OpenCV $ sudo python3 Camera_Gesture.py
* Serving Flask app 'Camera_Gesture'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.3.130:5000
Press CTRL+C to quit
```

11. Open a web browser (here we use Chrome as an example) on a device on the same LAN of the Raspberry Pi, enter in the address bar the Raspberry Pi's IP address and the video stream port number ":5000", as shown below:

Example: <http://192.168.3.130:5000>

12. Now capture video streams in real-time using the Raspberry Pi camera (Picamera2) and detect yellow objects in video frames. After detecting a yellow object, a green rectangular box will be drawn around it and a text label of "**Yellow Object**" will be added. Finally, the processed

video stream is displayed in real-time through a web page.. and when you want to terminate the running program, you can press the "**Ctrl+C**" shortcut key on the keyboard.

Video Streaming Demonstration



13. In **Camera_FindColor.py**, the default color recognition is to search for yellow objects, which is mainly achieved by defining the color range in the HSV (Hue, Saturation, Value) color space. If you want to find other colors, you need to modify the definition of the color range and the corresponding text labels. The specific steps are as follows:

Determine the HSV range of the target color: Different colors have different value ranges in the HSV color space. The approximate HSV range of the target color can be determined through tools such as online HSV color selectors or experiments.

Note that in practical applications, adjustments may need to be made based on specific lighting conditions and color depth.

Modify the color range in the code: In the **Camera_FindColor.py** file, find the code section that defines the yellow color range:

Define the lower limit of yellow in the HSV color space

```
lower = np.array([20, 100, 100], dtype=np.uint8)
```

Define the upper limit of yellow in the HSV color space

```
upper = np.array([30, 255, 255], dtype=np.uint8)
```

Modify it to the HSV range of the target color. For example, when searching for red:

Define the lower limit of red in the HSV color space

```
lower = np.array([0, 100, 100], dtype=np.uint8)
```

Define the upper limit of red in the HSV color space

```
upper = np.array([10, 255, 255], dtype=np.uint8)
```

Modify Text Label: When a target color object is detected, the code will draw a text label on the image. By default, when searching for yellow objects, the label is "**Yellow Object**". In order to display the detected colors more intuitively, it is necessary to modify this text label. Find the code section for drawing text labels:

```
cv2.putText(frame, "Yellow Object", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
```

Modify it to the text corresponding to the target color, such as when searching for red objects:

```
cv2.putText(frame, "Red Object", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
```

By following the above steps, you can modify the **Camera_FindColor.py** code to enable it to search for objects of colors other than yellow. During the modification process, attention should be paid to the accuracy of the HSV color range, as well as the clarity and accuracy of the text labels, to ensure that the program can correctly recognize and display the target color object.

15.5 Code

Complete code refer to [Camera_WatchDog.py](#)

```
001 #!/usr/bin/env/python
002 # File name   : Camera_WatchDog.py
003 # Website    : www.Aadept.com
004 # Author     : Aadept
005 # Date      : 2025/03/10
006 import io
007 import time
008 import cv2
009 import imutils
010 import numpy as np
```

```

011 from picamera2 import Picamera2, Preview
012 from base_camera import BaseCamera
013 import datetime
014 from importlib import import_module
015 import os
016 from flask import Flask, render_template, Response
017
018 app = Flask(__name__)
019 class Camera(BaseCamera):
020     def __init__(self):
021         super().__init__()
022         self.avg = None
023         self.drawing = 0
024         self.motionCounter = 0
025         self.lastMovtionCaptured = datetime.datetime.now()
026
027     def watchDog(self, frame):
028         gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
029         gray = cv2.GaussianBlur(gray, (21, 21), 0)
030
031         if self.avg is None:
032             print("[INFO] starting background model...")
033             self.avg = gray.copy().astype("float")
034             return frame
035
036         cv2.accumulateWeighted(gray, self.avg, 0.5)
037         frameDelta = cv2.absdiff(gray, cv2.convertScaleAbs(self.avg))
038
039         thresh = cv2.threshold(frameDelta, 5, 255, cv2.THRESH_BINARY)[1]
040         thresh = cv2.dilate(thresh, None, iterations=2)
041         cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
042         cnts = imutils.grab_contours(cnts)
043
044         for c in cnts:
045             if cv2.contourArea(c) < 5000:
046                 continue
047             (x, y, w, h) = cv2.boundingRect(c)
048             cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
049             self.drawing = 1
050             self.motionCounter += 1
051             self.lastMovtionCaptured = datetime.datetime.now()
052
053         if (datetime.datetime.now() - self.lastMovtionCaptured).seconds >= 0.5:
054             self.drawing = 0
055
056         return frame
057
058     @staticmethod
059     def frames():
060         with Picamera2() as camera:
061             camera.start()
062             time.sleep(2)
063             stream = io.BytesIO()
064             camera_ins = Camera()
065             try:
066                 while True:

```

```

067         camera.capture_file(stream, format='jpeg')
068         stream.seek(0)
069         frame = cv2.imdecode(np.frombuffer(stream.read(), dtype=np.uint8),
070 cv2.IMREAD_COLOR)
071         processed_frame = camera_ins.watchDog(frame)
072         _, encoded_frame = cv2.imencode('.jpg', processed_frame)
073         yield encoded_frame.tobytes()
074         stream.seek(0)
075         stream.truncate()
076     finally:
077         camera.stop()
078 @app.route('/')
079 def index():
080     """Video streaming home page."""
081     return render_template('index.html')
082
083
084 def gen(camera):
085     """Video streaming generator function."""
086     yield b'--frame\r\n'
087     while True:
088         frame = camera.get_frame()
089         yield b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n--frame\r\n'
090
091
092 @app.route('/video_feed')
093 def video_feed():
094     """Video streaming route. Put this in the src attribute of an img tag."""
095     return Response(gen(Camera()),
096                     mimetype='multipart/x-mixed-replace; boundary=frame')
097
098
099 if __name__ == '__main__':
100     app.run(host='0.0.0.0', threaded=True)
101
102
103
104
105
106
107

```

Complete code refer to [Camera_Gesture.py](#)

```

001 #!/usr/bin/env/python
002 # File name   : Camera_Gesture.py
003 # Website    : www.Adeept.com
004 # Author     : Adeept
005 # Date      : 2025/03/10
006 import io
007 import time
008 import cv2
009 import numpy as np
010 from picamera2 import Picamera2, Preview
011 from base_camera import BaseCamera
012 import os
013 from flask import Flask, render_template, Response

```

```

014 from importlib import import_module
015
016 app = Flask(__name__)
017 class Camera(BaseCamera):
018     @staticmethod
019     def frames():
020         with Picamera2() as camera:
021             camera.start()
022             time.sleep(2)
023             stream = io.BytesIO()
024             kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
025             try:
026                 while True:
027                     camera.capture_file(stream, format='jpeg')
028                     stream.seek(0)
029                     frame = cv2.imdecode(np.frombuffer(stream.read(), dtype=np.uint8),
030 cv2.IMREAD_COLOR)
031                     hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
032                     # Define the range of skin color in the HSV color space
033                     lower_skin = np.array([0, 20, 70], dtype=np.uint8)
034                     upper_skin = np.array([20, 255, 255], dtype=np.uint8)
035                     mask = cv2.inRange(hsv, lower_skin, upper_skin)
036                     # Morphological operations: erosion and dilation
037                     mask = cv2.erode(mask, kernel, iterations=2)
038                     mask = cv2.dilate(mask, kernel, iterations=2)
039                     contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
040                     if contours:
041                         max_contour = max(contours, key=cv2.contourArea)
042                         if cv2.contourArea(max_contour) > 500:
043                             hull = cv2.convexHull(max_contour, returnPoints=False)
044                             defects = cv2.convexityDefects(max_contour, hull)
045                             if defects is not None:
046                                 num_defects = 0
047                                 for i in range(defects.shape[0]):
048                                     s, e, f, d = defects[i, 0]
049                                     start = tuple(max_contour[s][0])
050                                     end = tuple(max_contour[e][0])
051                                     far = tuple(max_contour[f][0])
052                                     # Calculate the angle to determine if it is a convexity defect
053                                     a = np.sqrt((end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2)
054                                     b = np.sqrt((far[0] - start[0]) ** 2 + (far[1] - start[1]) ** 2)
055                                     c = np.sqrt((end[0] - far[0]) ** 2 + (end[1] - far[1]) ** 2)
056                                     angle = np.arccos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c)) * 57.2958
057                                     if angle <= 90:
058                                         num_defects += 1
059                                         cv2.circle(frame, far, 5, [0, 0, 255], -1)
060                                 if num_defects == 0:
061                                     cv2.putText(frame, "Fist", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1,
062 (0, 255, 0), 2)
063                                 elif num_defects >= 3:
064                                     cv2.putText(frame, "Open Hand", (50, 50), cv2.FONT_HERSHEY_SIMPLEX,
065 1, (0, 255, 0), 2)
066                                 # Encode the processed frame into JPEG format
067                                 _, encoded_frame = cv2.imencode('.jpg', frame)
068                                 yield encoded_frame.tobytes()
069                                 stream.seek(0)

```

```

070         stream.truncate()
071     finally:
072         camera.stop()
073 @app.route('/')
074 def index():
075     """Video streaming home page."""
076     return render_template('index.html')
077
078
079 def gen(camera):
080     """Video streaming generator function."""
081     yield b'--frame\r\n'
082     while True:
083         frame = camera.get_frame()
084         yield b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n--frame\r\n'
085
086
087 @app.route('/video_feed')
088 def video_feed():
089     """Video streaming route. Put this in the src attribute of an img tag."""
090     return Response(gen(Camera()),
091                     mimetype='multipart/x-mixed-replace; boundary=frame')
092
093
094 if __name__ == '__main__':
095     app.run(host='0.0.0.0', threaded=True)
096
097
098
099
100
101
102
103
104
105

```

Complete code refer to [Camera_FindColor.py](#)

```

001 #!/usr/bin/env/python
002 # File name   : Camera_FindColor.py
003 # Website    : www.Adeept.com
004 # Author     : Adeept
005 # Date      : 2025/03/10
006 import io
007 import time
008 import cv2
009 import numpy as np
010 from picamera2 import Picamera2, Preview
011 from base_camera import BaseCamera
012 import os
013 from flask import Flask, render_template, Response
014 from importlib import import_module
015
016 app = Flask(__name__)

```

```

017 class Camera(BaseCamera):
018     @staticmethod
019     def frames():
020         # Open the camera using Picamera2
021         with Picamera2() as camera:
022             # Start the camera
023             camera.start()
024             # Wait for 2 seconds to let the camera warm up
025             time.sleep(2)
026             # Create a ByteIO object to store image data
027             stream = io.BytesIO()
028             # Create an elliptical structuring element for morphological operations
029             kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
030             try:
031                 while True:
032                     # Capture an image from the camera and save it to the ByteIO stream in JPEG format
033                     camera.capture_file(stream, format='jpeg')
034                     # Move the pointer of the ByteIO stream to the beginning
035                     stream.seek(0)
036                     # Read data from the ByteIO stream and decode it into an image
037                     frame = cv2.imdecode(np.frombuffer(stream.read(), dtype=np.uint8),
038 cv2.IMREAD_COLOR)
039                     # Convert the image from BGR color space to HSV color space
040                     hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
041                     # Define the lower bound of yellow in the HSV color space
042                     lower = np.array([20, 100, 100], dtype=np.uint8)
043                     # Define the upper bound of yellow in the HSV color space
044                     upper = np.array([30, 255, 255], dtype=np.uint8)
045                     # Create a mask based on the color range
046                     mask = cv2.inRange(hsv, lower, upper)
047                     # Perform erosion operation on the mask, with 2 iterations
048                     mask = cv2.erode(mask, kernel, iterations=2)
049                     # Perform dilation operation on the mask, with 2 iterations
050                     mask = cv2.dilate(mask, kernel, iterations=2)
051                     # Find contours in the mask
052                     contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
053                     for contour in contours:
054                         # If the contour area is greater than 500
055                         if cv2.contourArea(contour) > 500:
056                             # Calculate the bounding rectangle of the contour
057                             x, y, w, h = cv2.boundingRect(contour)
058                             # Draw a green rectangle on the original image to mark the object's
059 position
060                             cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
061                             # Add the text label "Yellow Object" above the rectangle
062                             cv2.putText(frame, "Yellow Object", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
063 0.9, (0, 255, 0), 2)
064                             # Encode the processed image into JPEG format
065                             _, encoded_frame = cv2.imencode('.jpg', frame)
066                             # Return the encoded image data through a generator
067                             yield encoded_frame.tobytes()
068                             # Move the pointer of the ByteIO stream to the beginning
069                             stream.seek(0)
070                             # Clear the content of the ByteIO stream
071                             stream.truncate()
072             finally:

```



```

073             # Stop the camera
074             camera.stop()
075 @app.route('/')
076 def index():
077     """Video streaming home page."""
078     return render_template('index.html')
079
080
081 def gen(camera):
082     """Video streaming generator function."""
083     yield b'--frame\r\n'
084     while True:
085         frame = camera.get_frame()
086         yield b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n--frame\r\n'
087
088
089 @app.route('/video_feed')
090 def video_feed():
091     """Video streaming route. Put this in the src attribute of an img tag."""
092     return Response(gen(Camera()),
093                     mimetype='multipart/x-mixed-replace; boundary=frame')
094
095
096 if __name__ == '__main__':
097     app.run(host='0.0.0.0', threaded=True)
098
099
100
101
102

```

Code explanation

Camera_WatchDog.py

Initialization Stage:

During the initialization phase, the code underwent a series of preparatory work, including importing necessary libraries, creating Flask application instances, initializing camera related settings, and defining variables for motion detection..

Loop Control Process:

Continuously capture camera images in a loop, perform motion detection, and stream the processed images to a webpage for display.

Stage 1: Stage 1: Video Frame Capture → Continuously obtain new video frames from the Raspberry Pi camera and save them in JPEG format for subsequent processing and display in real - time.

Stage 2: Image Data Preparation and Decoding → Move the byte - stream pointer to the start, read data, and decode it into a color image format processable by OpenCV.

Stage 3: Motion Detection and Processing → Call the watchDog method to detect and process motion in the current frame, marking the moving areas.

Stage 4: Image Encoding and Streaming Transmission → Re - encode the processed frame into JPEG format and send it frame - by - frame to the web page via a generator for real - time video playback.

Stage 5: Byte Stream Cleanup and Loop Preparation → Move the byte - stream pointer to the beginning and clear its content to prepare for the next image capture.

Stage 6: Camera Resource Management → Close the camera resources in the finally block to ensure proper release when the program ends and avoid resource leaks.

[Camera_FindColor.py](#)

Initialization Stage:

At this stage, the code mainly involves necessary library imports, creation of Flask application instances, initialization of cameras, and creation of structural elements required for morphological operations.

Loop Control Process:

Enter an infinite loop and perform the following steps in sequence to process and transmit video frames.

Stage 1: Image capture and decoding: Use camera.captureFILE to capture images and save them to the stream, then use cv2.imdecode to decode them into an image format that OpenCV can process.

Stage 2: Color space conversion: Convert an image from BGR color space to HSV color space, as color filtering is easier in HSV color space.

Stage 3: Color filtering: Define the upper and lower limits of yellow in the HSV color space, create a mask using cv2.inRange, and filter out the yellow area.

Stage 4: Morphological operation: Corrode and expand the mask to remove noise and fill small holes inside the object.

Stage 5: Image encoding and transmission: Encode the processed image into JPEG format, and return the encoded image data through the generator yield keyword for subsequent streaming on web pages.

Stage 6: Stream cleaning: Move the pointer of the stream to the beginning and clear the content to prepare for the next image capture.

Camera_Gesture.py

Initialization Stage:

Import multiple libraries such as io, time, cv2, etc. for functions such as input/output, time control, image processing, etc; Create a Flask application instance app and initialize the camera in the frames method of the Camera class, including startup, preheating, creating data storage objects, and image processing elements.

Loop Control Process:

Enter an infinite loop and perform the following steps in sequence to process and transmit video frames.

Stage 1: Capture and process frames: Capture images from the camera, convert and decode them, and then transform the color space.

Stage 2: Skin color detection: Define skin color range, create and optimize masks.

Stage 3: Gesture recognition: find the contour, process the maximum contour, judge the gesture and draw a prompt.

Stage 4: Encoding output: Encode images, output frame by frame, reset and clear storage objects.