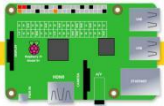
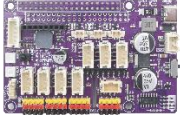




Lesson 12 Implement Object, Color & Gesture Recognition with OpenCV

12.1 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.

12.2 Required Components

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

12.3 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.

12.4 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_DarkPaw-V3/Examples/06_OpenCV/
```

```
pi@raspberrypi:~ $ cd Adeept_DarkPaw-V3/Examples/06_OpenCV/  
pi@raspberrypi:~/Adeept_DarkPaw-V3/Examples/06_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_DarkPaw-V3/Examples/06_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
```

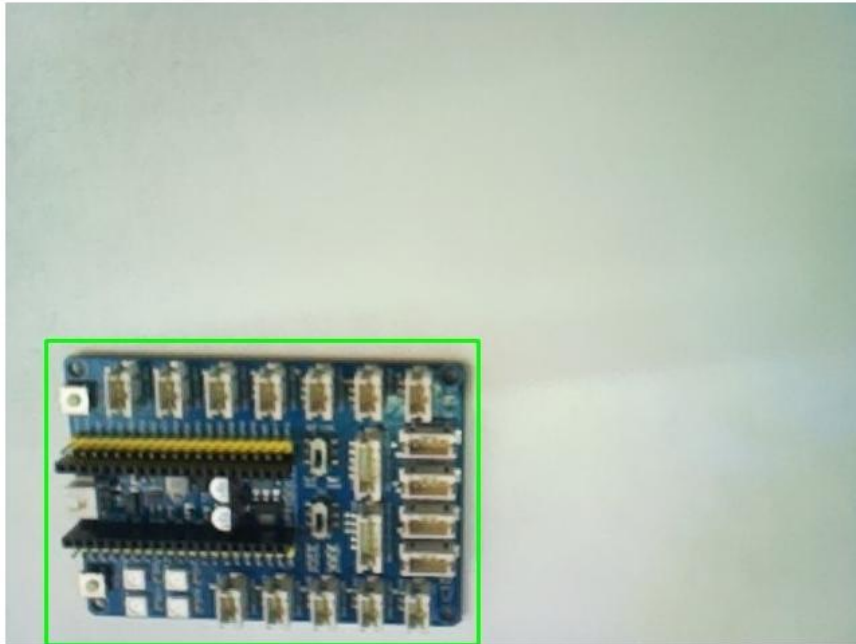
```
pi@raspberrypi:~/Adeept_DarkPaw-V3/Examples/06_OpenCV $ sudo python3 Camera_WatchDog.py  
* Serving Flask app 'Camera_WatchDog'  
* Debug mode: off  
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
* Running on all addresses (0.0.0.0)  
* Running on http://127.0.0.1:5000  
* Running on http://192.168.3.31:5000  
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://192.168.3.31:5000>

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_DarkPaw-V3/Examples/06_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://192.168.3.31:5000
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://192.168.3.31:5000>

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:~/Adeept_DarkPaw-V3/Examples/06_OpenCV $ sudo python3 Camera_FindColor.py
* Serving Flask app 'Camera_FindColor'
* 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://192.168.3.31:5000
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://192.168.3.31:5000>

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

```
colorLower = np.array([24, 100, 100])
```

Define the upper limit of yellow in the HSV color space

```
colorUpper = np.array([44, 255, 255])
```

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

```
colorLower = np.array([155, 105, 105])
```

Define the upper limit of red in the HSV color space

```
colorUpper = np.array([180, 255, 255])
```

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(imgInput, "Yellow Object", (int(box_x - radius), int(box_y - radius - 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(imgInput, "Red Object", (int(box_x - radius), int(box_y - radius - 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.

12.5 Code

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

```
001  #!/usr/bin/env/python3
002  # File name   : Camera_WatchDog.py
003  # Website    : www.Aadept.com
004  # Author     : Aadept
005  # Date      : 2025/04/24
006  import time
007  import cv2
008  import imutils
009  import numpy as np
010  from picamera2 import Picamera2
011  import libcamera
012  from base_camera import BaseCamera
013  import datetime
014  from flask import Flask, render_template, Response
015
016  hflip = 0
017  vflip = 0
018  ImgIsNone = 0
019  app = Flask(__name__)
020
021
022  class Camera(BaseCamera):
023      def __init__(self):
024          super().__init__()
025          self.avg = None
026          self.drawing = 0
027          self.motionCounter = 0
028          self.lastMovtionCaptured = datetime.datetime.now()
029
030      def watchDog(self, frame):
031          gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
032          gray = cv2.GaussianBlur(gray, (21, 21), 0)
033
034          if self.avg is None:
035              print("[INFO] starting background model...")
036              self.avg = gray.copy().astype("float")
037              return frame
038
039          cv2.accumulateWeighted(gray, self.avg, 0.5)
040          frameDelta = cv2.absdiff(gray, cv2.convertScaleAbs(self.avg))
041
042          thresh = cv2.threshold(frameDelta, 5, 255, cv2.THRESH_BINARY)[1]
043          thresh = cv2.dilate(thresh, None, iterations=2)
044          cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
045          cnts = imutils.grab_contours(cnts)
046
047          for c in cnts:
```

```

048         if cv2.contourArea(c) < 5000:
049             continue
050         (x, y, w, h) = cv2.boundingRect(c)
051         cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
052         self.drawing = 1
053         self.motionCounter += 1
054         self.lastMovtionCaptured = datetime.datetime.now()
055
056     if (datetime.datetime.now() - self.lastMovtionCaptured).seconds >= 0.5:
057         self.drawing = 0
058
059     return frame
060
061 @staticmethod
062 def frames():
063     picam2 = Picamera2()
064     preview_config = picam2.preview_configuration
065     preview_config.size = (640, 480)
066     preview_config.format = 'RGB888'
067     preview_config.transform = libcamera.Transform(hflip=hflip, vflip=vflip)
068     preview_config.colour_space = libcamera.ColorSpace.Sycc()
069     preview_config.buffer_count = 4
070     preview_config.queue = True
071
072     if not picam2.is_open:
073         raise RuntimeError('Could not start camera.')
074
075     try:
076         picam2.start()
077         time.sleep(2)
078         camera_ins = Camera()
079
080         while True:
081             img = picam2.capture_array()
082
083             if img is None:
084                 if ImgIsNone == 0:
085                     print("-----")
086                     print("\033[31merror: Unable to read camera data.\033[0m")
087                     print("Press the keyboard keys \033[34m'Ctrl + C'\033[0m multiple times to exit
088 the current program.")
089                     print("-----Ctrl+C quit-----")
090                     ImgIsNone = 1
091                     continue
092
093             processed_frame = camera_ins.watchDog(img)
094             _, encoded_frame = cv2.imencode('.jpg', processed_frame)
095             yield encoded_frame.tobytes()
096
097     except Exception as e:
098         print(f"\033[38;5;1mError:\033[0m\n{e}")
099         print("\nPlease check whether the camera is connected well, "
100               "and disable the \"legacy camera driver\" on raspi-config")
101     finally:
102         picam2.stop()
103

```

```

104
105 @app.route('/')
106 def index():
107     """Video streaming home page."""
108     return render_template('index.html')
109
110
111 def gen(camera):
112     """Video streaming generator function."""
113     yield b'--frame\r\n'
114     while True:
115         frame = camera.get_frame()
116         yield b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n--frame\r\n'
117
118
119 @app.route('/video_feed')
120 def video_feed():
121     """Video streaming route. Put this in the src attribute of an img tag."""
122     return Response(gen(Camera()),
123                     mimetype='multipart/x-mixed-replace; boundary=frame')
124
125
126 if __name__ == '__main__':
127     app.run(host='0.0.0.0', threaded=True)

```

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

```

001 #!/usr/bin/env/python3
002 # File name : Camera_Gesture.py
003 # Website : www.Adeept.com
004 # Author : Adeept
005 # Date : 2025/04/24
006 from flask import Flask, render_template, Response
007 import time
008 import cv2
009 import numpy as np
010 import picamera2
011 import libcamera
012 from base_camera import BaseCamera
013 from picamera2 import Picamera2
014 hflip = 0
015 vflip = 0
016 ImgIsNone = 0
017 app = Flask(__name__)
018
019 class Camera(BaseCamera):
020     @staticmethod
021     def frames():
022         picam2 = Picamera2()
023         preview_config = picam2.preview_configuration
024         preview_config.size = (640, 480)
025         preview_config.format = 'RGB888'
026         preview_config.transform = libcamera.Transform(hflip=hflip, vflip=vflip)
027         preview_config.colour_space = libcamera.ColorSpace.Sycc()
028         preview_config.buffer_count = 4
029         preview_config.queue = True
030

```

```

031     if not picam2.is_open:
032         raise RuntimeError('Could not start camera.')
033
034     try:
035         picam2.start()
036     except Exception as e:
037         print(f"\033[38;5;1mError:\033[0m\n{e}")
038         print("\nPlease check whether the camera is connected well, "
039               "and disable the \"legacy camera driver\" on raspi-config")
040
041     kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
042
043     while True:
044         img = picam2.capture_array()
045
046         if img is None:
047             if ImgIsNone == 0:
048                 print("-----")
049                 print("\033[31merror: Unable to read camera data.\033[0m")
050                 print("Press the keyboard keys \033[34m'Ctrl + C'\033[0m multiple times to exit the
051 current program.")
052                 print("-----Ctrl+C quit-----")
053                 ImgIsNone = 1
054                 continue
055
056         hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
057         lower_skin = np.array([0, 20, 70], dtype=np.uint8)
058         upper_skin = np.array([20, 255, 255], dtype=np.uint8)
059         mask = cv2.inRange(hsv, lower_skin, upper_skin)
060         mask = cv2.erode(mask, kernel, iterations=2)
061         mask = cv2.dilate(mask, kernel, iterations=2)
062         contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
063         if contours:
064             max_contour = max(contours, key=cv2.contourArea)
065             if cv2.contourArea(max_contour) > 500:
066                 hull = cv2.convexHull(max_contour, returnPoints=False)
067                 defects = cv2.convexityDefects(max_contour, hull)
068                 if defects is not None:
069                     num_defects = 0
070                     for i in range(defects.shape[0]):
071                         s, e, f, d = defects[i, 0]
072                         start = tuple(max_contour[s][0])
073                         end = tuple(max_contour[e][0])
074                         far = tuple(max_contour[f][0])
075                         a = np.sqrt((end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2)
076                         b = np.sqrt((far[0] - start[0]) ** 2 + (far[1] - start[1]) ** 2)
077                         c = np.sqrt((end[0] - far[0]) ** 2 + (end[1] - far[1]) ** 2)
078                         angle = np.arccos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c)) * 57.2958
079                         if angle <= 90:
080                             num_defects += 1
081                             cv2.circle(img, far, 5, [0, 0, 255], -1)
082                 if num_defects < 3:
083                     cv2.putText(img, "Fist", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,
084 0), 2)
085                 elif num_defects >= 3:
086                     cv2.putText(img, "Open Hand", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,

```

```

087     255, 0), 2)
088         _, encoded_frame = cv2.imencode('.jpg', img)
089         yield encoded_frame.tobytes()
090
091 @app.route('/')
092 def index():
093     return render_template('index.html')
094
095 def gen(camera):
096     yield b'--frame\r\n'
097     while True:
098         frame = camera.get_frame()
099         yield b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n--frame\r\n'
100
101 @app.route('/video_feed')
102 def video_feed():
103     return Response(gen(Camera()),
104                     mimetype='multipart/x-mixed-replace; boundary=frame')
105
106 if __name__ == '__main__':
107     app.run(host='0.0.0.0', threaded=True)
108
109
110
111

```

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

```

001  #!/usr/bin/env/python3
002  # File name   : Camera_FindColor.py
003  # Website    : www.Adeept.com
004  # Author     : Adeept
005  # Date      : 2025/04/24
006  from importlib import import_module
007  import os
008  from flask import Flask, render_template, Response
009  import io
010  import time
011  import cv2
012  import numpy as np
013  import threading
014  import picamera2
015  import libcamera
016  from picamera2 import Picamera2
017  from base_camera import BaseCamera
018
019  hflip = 0
020  vflip = 0
021  ImgIsNone = 0
022  app = Flask(__name__)
023
024  colorUpper = np.array([44, 255, 255])
025  colorLower = np.array([24, 100, 100])
026
027

```

```

028 def map(input, in_min, in_max, out_min, out_max):
029     return (input - in_min) / (in_max - in_min) * (out_max - out_min) + out_min
030
031
032 class Camera(BaseCamera):
033     modeSelect = 'findColor'
034
035     @staticmethod
036     def frames():
037         picam2 = Picamera2()
038
039         preview_config = picam2.preview_configuration
040         preview_config.size = (640, 480)
041         preview_config.format = 'RGB888'
042         preview_config.transform = libcamera.Transform(hflip=hflip, vflip=vflip)
043         preview_config.colour_space = libcamera.ColorSpace.Sycc()
044         preview_config.buffer_count = 4
045         preview_config.queue = True
046
047         if not picam2.is_open:
048             raise RuntimeError('Could not start camera.')
049
050         try:
051             picam2.start()
052         except Exception as e:
053             print(f"\033[38;5;1mError:\033[0m\n{e}")
054             print("\nPlease check whether the camera is connected well, "
055                   "and disable the \"legacy camera driver\" on raspi-config")
056
057
058         CVThreading = 0
059         CVMode = 'none'
060         imgCV = None
061         findColorDetection = 0
062         radius = 0
063         box_x = None
064         box_y = None
065         drawing = 0
066         __flag = threading.Event()
067         __flag.clear()
068
069         def mode(invar, imgInput):
070             nonlocal CVMode, imgCV
071             CVMode = invar
072             imgCV = imgInput
073             __flag.set()
074
075         def elementDraw(imgInput):
076             nonlocal findColorDetection, drawing, radius, box_x, box_y
077             if CVMode == 'findColor':
078                 if findColorDetection:
079                     drawing = 1
080                 else:
081                     drawing = 0
082
083             if radius > 10 and drawing:
084                 cv2.rectangle(imgInput, (int(box_x - radius), int(box_y + radius)),

```

```

084         (int(box_x + radius), int(box_y - radius)), (0, 255, 0), 2)
085         cv2.putText(imgInput, "Yellow Object", (int(box_x - radius), int(box_y - radius -
086 10)), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
087
088     return imgInput
089
090     def findColor(frame_image):
091         nonlocal findColorDetection, radius, box_x, box_y
092         if frame_image is None or frame_image.size == 0:
093             print("Error: Input image is empty in findColor function")
094             return
095         hsv = cv2.cvtColor(frame_image, cv2.COLOR_BGR2HSV)
096         mask = cv2.inRange(hsv, colorLower, colorUpper)
097         mask = cv2.erode(mask, None, iterations=2)
098         mask = cv2.dilate(mask, None, iterations=2)
099         cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
100                                cv2.CHAIN_APPROX_SIMPLE)[-2]
101         center = None
102         if len(cnts) > 0:
103             findColorDetection = 1
104             c = max(cnts, key=cv2.contourArea)
105             ((box_x, box_y), radius) = cv2.minEnclosingCircle(c)
106             M = cv2.moments(c)
107             center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
108         else:
109             findColorDetection = 0
110         __flag.clear()
111
112     def run():
113         nonlocal CVThreading, CVMode, imgCV
114         while 1:
115             __flag.wait()
116             if CVMode == 'findColor':
117                 if imgCV is not None:
118                     CVThreading = 1
119                     findColor(imgCV)
120                     CVThreading = 0
121                 else:
122                     pass
123
124         thread = threading.Thread(target=run)
125         thread.start()
126
127         mode(Camera.modeSelect, None)
128
129         while True:
130             img = picam2.capture_array()
131
132             if img is None:
133                 if ImgIsNone == 0:
134                     print("-----")
135                     print("\033[31merror: Unable to read camera data.\033[0m")
136                     print("Press the keyboard keys \033[34m'Ctrl + C'\033[0m multiple times to exit the
137 current program.")
138                     print("-----Ctrl+C quit-----")
139                     ImgIsNone = 1

```



```

140         continue
141
142         if Camera.modeSelect == 'none':
143             __flag.clear()
144         elif Camera.modeSelect == 'findColor':
145             if not CVThreading:
146                 mode(Camera.modeSelect, img)
147             try:
148                 img = elementDraw(img)
149             except:
150                 pass
151
152         if cv2.imencode('.jpg', img)[0]:
153             yield cv2.imencode('.jpg', img)[1].tobytes()
154
155
156 @app.route('/')
157 def index():
158     """Video streaming home page."""
159     return render_template('index.html')
160
161
162 def gen(camera):
163     """Video streaming generator function."""
164     yield b'--frame\r\n'
165     while True:
166         frame = camera.get_frame()
167         yield b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n--frame\r\n'
168
169
170 @app.route('/video_feed')
171 def video_feed():
172     """Video streaming route. Put this in the src attribute of an img tag."""
173     return Response(gen(Camera()),
174                     mimetype='multipart/x-mixed-replace; boundary=frame')
175
176
177 if __name__ == '__main__':
178     app.run(host='0.0.0.0', threaded=True)

```

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: 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.