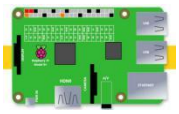
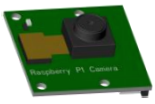
Lesson 17 Demonstration of the Video and Photographing

17.1 Overview

This lesson focuses on demonstrating how to utilize the video and photographing capabilities of a Raspberry Pi integrated with an Adeept Robot HAT V3.2 and a USB Camera Module. We will cover the principles behind video frame processing, both in single - threaded and multi - threaded scenarios, and provide a step - by - step guide on setting up and running the relevant programs. Additionally, the code implementation and its explanations will be presented to help you understand how the system works.

17.2 Required Components

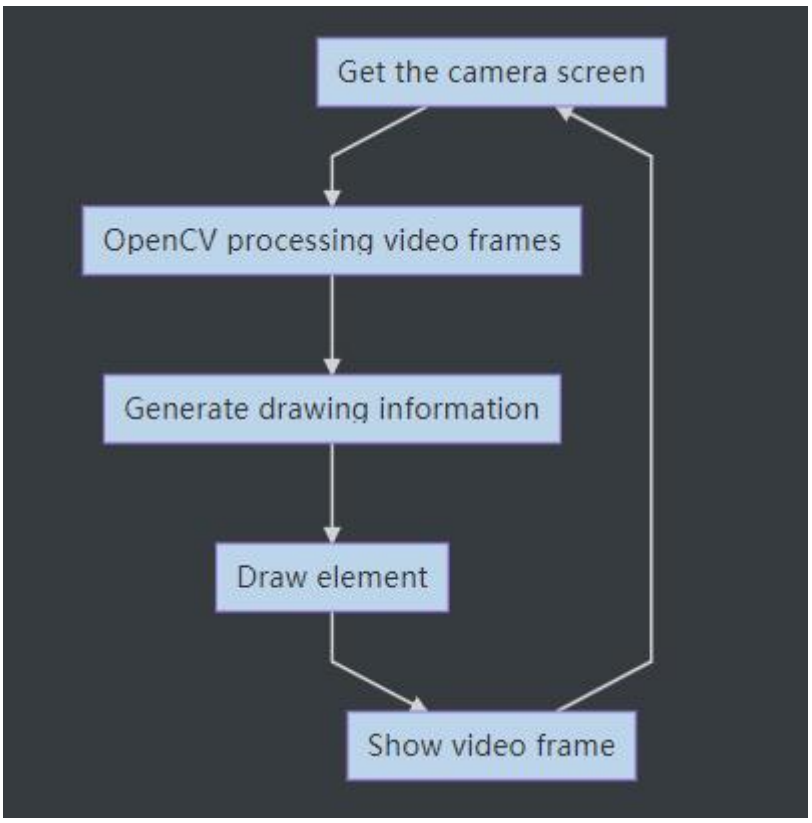
Components	Quantity	Picture
Raspberry Pi	1	
Adeept Robot HAT V3.2	1	
Camera Module	1	
Camera Cable	1	

17.3 Principle of Multithreaded Video Frames Processing

The OpenCV function is based on the flask-video-streaming project on GitHub,here we just changed the *camera_opencv.py* file for operations with OpenCV.

Single threaded video frames processing

The process for single threading is as follows:



1. **Capture Frame**: First, capture an image frame from the camera.

2. **OpenCV Analysis**: Analyze the frame with OpenCV. This could involve tasks like object detection, image filtering, or calculating the position of a target within the frame.

3. **Generate Drawing Information**: Based on the OpenCV analysis, generate the information to be drawn, such as the central position of the target or the text to be displayed on the screen.

4. **Draw Elements**: Draw the elements (like rectangles around detected objects or text) on the frame according to the generated information.

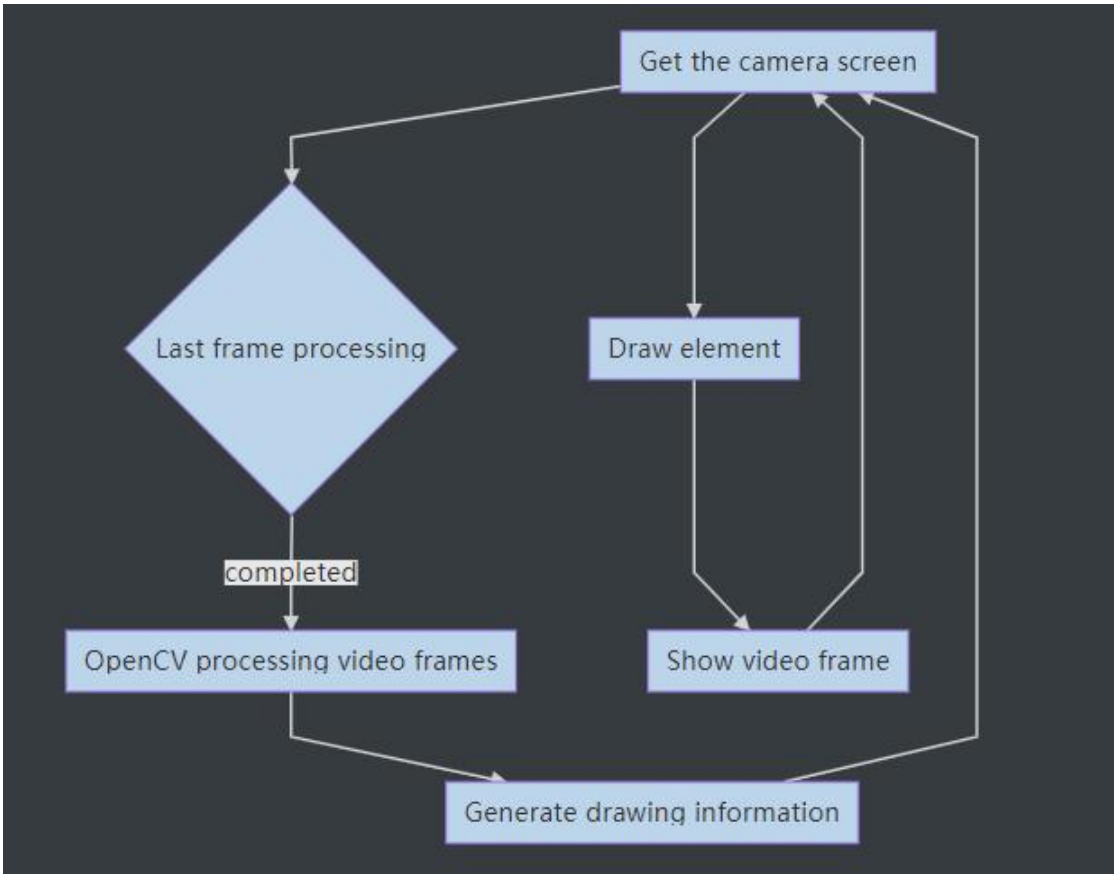
5. **Display Frame**: Display the image which has been processed and drawn on the webpage.

This single - threaded process is inefficient because it needs to wait for the OpenCV processing and screen display of each frame to complete before starting the next frame processing. As a

result, it may cause the video transmission to be stuck or have a low frame rate, especially when the OpenCV processing is computationally intensive.

Multi-threaded video frames processing

The process for multi - threaded video frames processing is as shown below:



1. **Get the camera screen**: This is the starting point of the process, where video frames are captured from the camera.

2. **Draw element**: After obtaining the frames, elements are drawn on the video frames in this step, such as the detected hand gesture contours.

3. **Show video frame**: Display the video frames with the drawn elements. Meanwhile, the displayed video frames are used to generate drawing information, which may include details such as the position and shape of the drawings.

4. **Generate drawing information**: The generated drawing information is fed back to the "Draw element" step on one hand to assist subsequent drawing. On the other hand, the video frames enter the "Last frame processing" judgment stage.

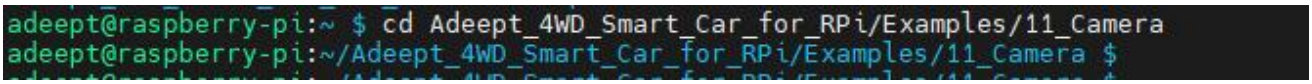
5. **Last frame processing judgment**: Determine whether the processing of the last frame is completed. If not, return for further processing. If completed, proceed to the "OpenCV processing video frames" step, where the OpenCV library is used to process the video frames, such as denoising, binarization, feature extraction, etc. The processed video frames then return to the "Get the camera screen" step, forming a loop to enable continuous processing and display of the camera video.

17.4 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_4WD_Smart_Car_for_RPi/Examples/11_Camera
```



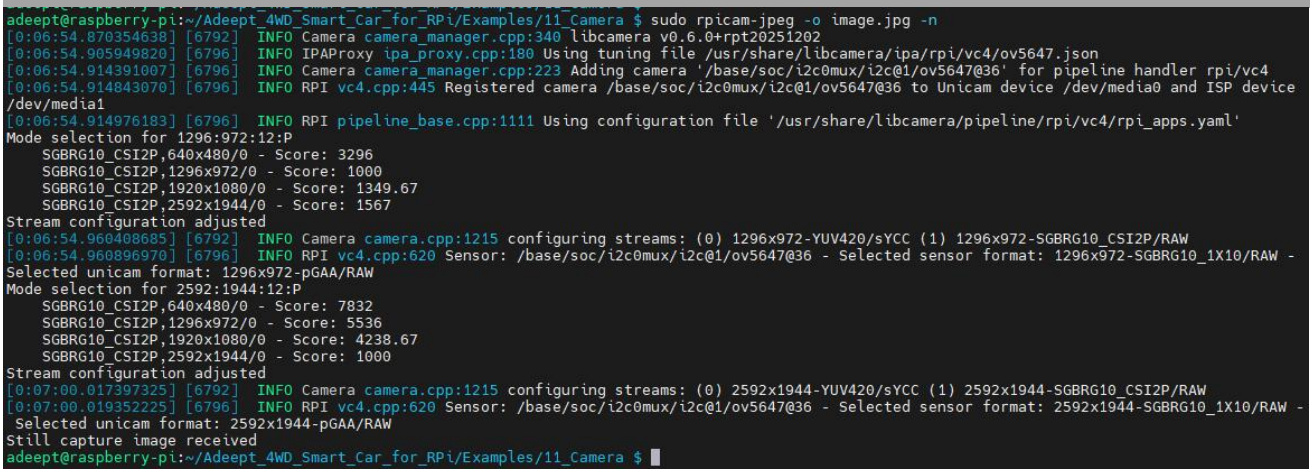
3. **View Directory Contents**: Type "**ls**" in the terminal and press Enter. This will display all the files in the current directory, ensuring that the "**app.py**" ,"**camera_pi2.py**"and "**base_camera.py**" file is present:

```
ls
```



4. **Run the Program**: After successful operation, the camera will take a photo **image.jpg**:(Note: If you are using the Bookworm system, please use the command: sudo libcamera-jpeg -o image.jpg -n)

```
sudo rpicam-jpeg -o image.jpg -n
```



Some warning messages may appear, please ignore it. If other messages appear, please check whether the camera is connected correctly.

Note: You need to disconnect the Raspberry Pi power supply before plugging or unplugging the camera cable.

5. Type in "**ls**" to view the file.

```
ls
```

```
adeept@raspberry-pi:~/Adeept_4WD_Smart_Car_for_RPi/Examples/11_Camera $ ls
app.py  base_camera.py  camera_pi2.py  image.jpg  __pycache__  templates
```

6. **Install Dependencies and Manage Programs**:The Raspberry Pi robot integrates real - time video and OpenCV functions, and there are multiple ways to achieve real - time transmission of videos captured by Raspberry Pi cameras over the network. This article will introduce a specific method for achieving real - time video transmission.

The project uses Flask and related dependencies which have been included in the installation scripts for the Adeept robot. However, if your Raspberry Pi has not run the script before or if there are issues, you may need to install them.

```
sudo pip3 install flask

sudo pip3 install flask_cors
```

When the Raspberry Pi is configured with the robot software, the Raspberry Pi will automatically run the **WebServer.py** program. If you need to use the camera in other programs, you need to terminate this program. Termination command:

```
sudo killall python3
```

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

```
sudo python3 app.py
```

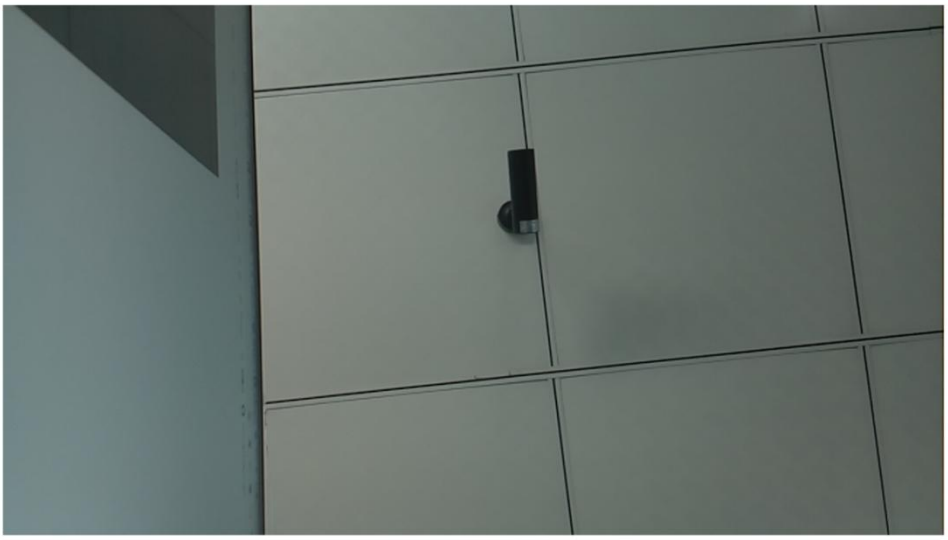
```
pi@raspberrypi:~/Adeept_4WD_Smart_Car_for_RPi/Examples/11_Camera $ sudo python3 app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server
 instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.3.130:5000
Press CTRL+C to quit
```

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**

8. Now you can view the webpage created by the Raspberry Pi on your mobile or computer. After data is loaded successfully, it'll display the videos captured by the Raspberry Pi in real time.

Video Streaming Demonstration



17.5 Code

Complete code refer to app.py

```
#!/usr/bin/env/python
# File name   : app.py
# Website     : www.Adeept.com
# Author      : Adeept
# Date        : 2025/03/10
#!/usr/bin/env python
from importlib import import_module
import os
from flask import Flask, render_template, Response

# import camera driver
#if os.environ.get('CAMERA'):
#    Camera = import_module('camera_' + os.environ['CAMERA']).Camera
#else:
#    from camera import Camera

# Raspberry Pi camera module (requires picamera package)
from camera_pi2 import Camera

app = Flask(__name__)


@app.route('/')
def index():
    """Video streaming home page."""
    return render_template('index.html')
```

```
27
28
29  def gen(camera):
30      """Video streaming generator function."""
31      yield b'--frame\r\n'
32      while True:
33          frame = camera.get_frame()
34          yield b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n--frame\r\n'
35
36
37  @app.route('/video_feed')
38  def video_feed():
39      """Video streaming route. Put this in the src attribute of an img tag."""
40      return Response(gen(Camera()),
41                      mimetype='multipart/x-mixed-replace; boundary=frame')
42
43
44  if __name__ == '__main__':
45      app.run(host='0.0.0.0', threaded=True)
```

Complete code refer to base_camera.py

```
001 #!/usr/bin/env/python
002 # File name   : base_camera.py
003 # Website     : www.Adeept.com
004 # Author      : Adeept
005 # Date        : 2025/03/10
006 import time
007 import threading
008 try:
009     from greenlet import getcurrent as get_ident
010 except ImportError:
011     try:
012         from thread import get_ident
013     except ImportError:
014         from _thread import get_ident
015
016
017 class CameraEvent(object):
018     """An Event-like class that signals all active clients when a new frame is
019     available.
020     """
021     def __init__(self):
022         self.events = {}
023
024     def wait(self):
025         """Invoked from each client's thread to wait for the next frame."""
026         ident = get_ident()
027         if ident not in self.events:
028             # this is a new client
029             # add an entry for it in the self.events dict
030             # each entry has two elements, a threading.Event() and a timestamp
031             self.events[ident] = [threading.Event(), time.time()]
032         return self.events[ident][0].wait()
033
```

```
034     def set(self):
035         """Invoked by the camera thread when a new frame is available."""
036         now = time.time()
037         remove = None
038         for ident, event in self.events.items():
039             if not event[0].isSet():
040                 # if this client's event is not set, then set it
041                 # also update the last set timestamp to now
042                 event[0].set()
043                 event[1] = now
044             else:
045                 # if the client's event is already set, it means the client
046                 # did not process a previous frame
047                 # if the event stays set for more than 5 seconds, then assume
048                 # the client is gone and remove it
049                 if now - event[1] > 5:
050                     remove = ident
051         if remove:
052             del self.events[remove]
053 
054     def clear(self):
055         """Invoked from each client's thread after a frame was processed."""
056         self.events[get_ident()][0].clear()
057 
058 
059 class BaseCamera(object):
060     thread = None  # background thread that reads frames from camera
061     frame = None  # current frame is stored here by background thread
062     last_access = 0  # time of last client access to the camera
063     event = CameraEvent()
064 
065     def __init__(self):
066         """Start the background camera thread if it isn't running yet."""
067         if BaseCamera.thread is None:
068             BaseCamera.last_access = time.time()
069 
070             # start background frame thread
071             BaseCamera.thread = threading.Thread(target=self._thread)
072             BaseCamera.thread.start()
073 
074             # wait until first frame is available
075             BaseCamera.event.wait()
076 
077     def get_frame(self):
078         """Return the current camera frame."""
079         BaseCamera.last_access = time.time()
080 
081         # wait for a signal from the camera thread
082         BaseCamera.event.wait()
083         BaseCamera.event.clear()
084 
085         return BaseCamera.frame
086 
087     @staticmethod
088     def frames():
089         """Generator that returns frames from the camera."""
```

```
090         raise RuntimeError('Must be implemented by subclasses.')
091
092     @classmethod
093     def _thread(cls):
094         """Camera background thread."""
095         print('Starting camera thread.')
096         frames_iterator = cls.frames()
097         for frame in frames_iterator:
098             BaseCamera.frame = frame
099             BaseCamera.event.set()  # send signal to clients
100             time.sleep(0)
101
102             # if there hasn't been any clients asking for frames in
103             # the last 10 seconds then stop the thread
104             if time.time() - BaseCamera.last_access > 10:
105                 frames_iterator.close()
106                 print('Stopping camera thread due to inactivity.')
107                 break
108         BaseCamera.thread = None
```

Complete code refer to camera_pi2.py

```
01  #!/usr/bin/env/python
02  # File name   : camera_pi2.py
03  # Website     : www.Adeept.com
04  # Author      : Adeept
05  # Date        : 2025/03/10
06  import io
07  import time
08  from picamera2 import Picamera2, Preview
09  from base_camera import BaseCamera
10
11  class Camera(BaseCamera):
12      @staticmethod
13      def frames():
14          with Picamera2() as camera:
15              camera.start()
16
17              # let camera warm up
18              time.sleep(2)
19
20              stream = io.BytesIO()
21              try:
22                  while True:
23                      camera.capture_file(stream, format='jpeg')
24                      stream.seek(0)
25                      yield stream.read()
26
27                      # reset stream for next frame
28                      stream.seek(0)
29                      stream.truncate()
30              finally:
31                  camera.stop()
```

Code explanations

app.py

Loop Control Process

Initiate the Loop: Enter an infinite loop to continuously perform the following operations related to video streaming.

Video Frame Generation and Streaming:

Step 1: Call the functions from base_camera.py and camera_pi2.py to capture a video frame from the camera.

Step 2: Process the captured frame using OpenCV (as per the functionality implemented in those files).

Step 3: Generate the drawing information based on the processed frame.

Step 4: Draw the relevant elements on the frame according to the drawing information.

Step 5: Send the processed and drawn frame to the client's web browser via the /video_feed route, which is set up to stream the video data.

Step 6: Wait for a short period (determined by the desired frame rate) before repeating the process for the next frame. This creates a continuous video - streaming effect.