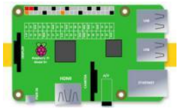
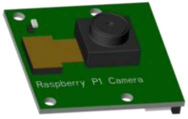
Lesson 10 Transmit Image in Real Time

This tutorial introduces how to transmit images in real time via the Raspberry Pi.

10.1 Components & Parts

Components	Quantity	Picture
Raspberry Pi	1	
Robot HAT	1	
Camera	1	
Camera Flex Cable (black)	1	

10.2 Transmitting via Flask-Video-Streaming

The Raspberry Pi robot features the real-time video and OpenCV functions. There are many methods of transmitting videos captured by the Raspberry Pi camera via network in a real time manner, and this tutorial introduces an open source project following the MIT License on Github:

https://github.com/miguelgrinberg/flask-video-streaming

The project uses Flask and related dependencies which have been included in the installation scripts for the Adeept robot. You may need to install them if your Raspberry Pi has run the script before.

```
sudo pip3 install flask
```

```
sudo pip3 install flask_cors
```

The OpenCV part will not be involved here; the tutorial only introduces how to view the image of the Raspberry Pi camera on other devices in real time. First, download the flask-video-streaming project. You can clone on Github or download on your computer and transfer to the Raspberry Pi, using the command on Raspberry Pi Command Line:

```
sudo git clone https://github.com/miguelgrinberg/flask-video-streaming.git
```

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
```

After flask-video-streaming is downloaded on the Raspberry Pi or transferred, run the file app.py in the project:

```
cd flask-video-streaming
```

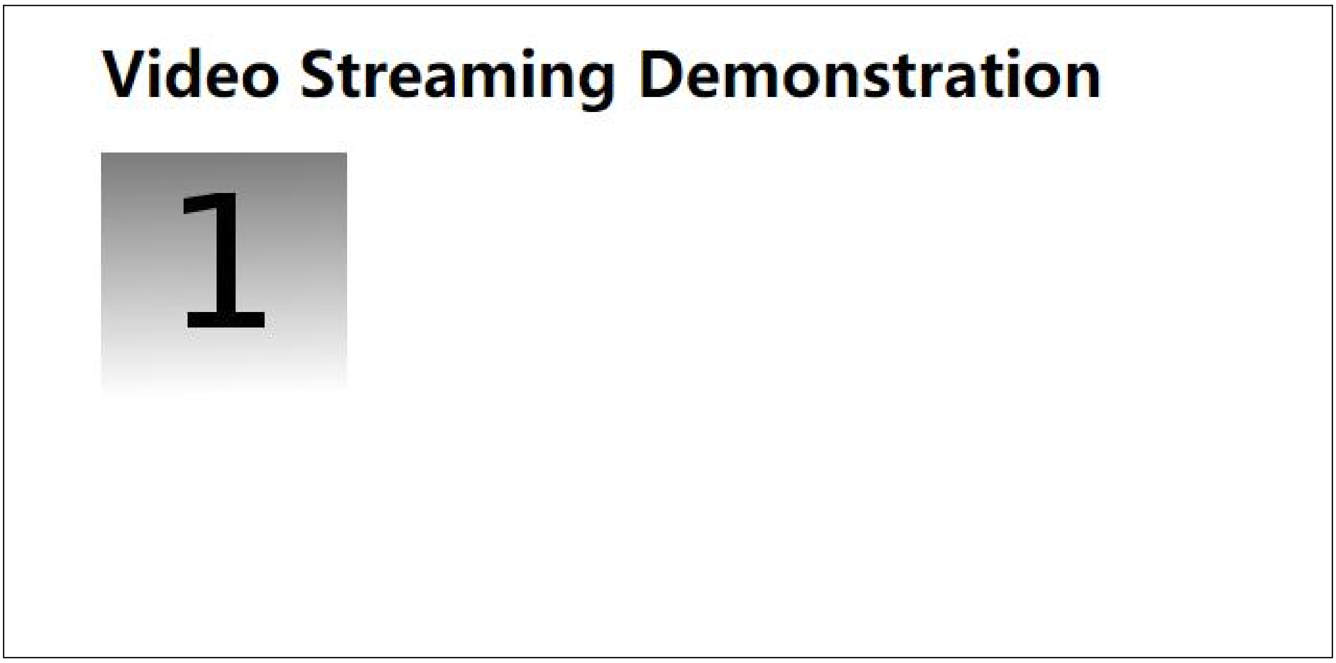
```
sudo python3 app.py
```

Pay attention not to run by the command "sudo python3 flask-video-streaming/app.py", or there will be an error of unfound *.jpeg file.

Open a web browser (Chrome for 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:

```
192.168.3.157:5000
```

Then you can view the webpage created by the Raspberry Pi on your mobile or computer. Note that by default, images of 3 numbers 1, 2, and 3 will loop instead of anything from the Raspberry Pi.



Video Streaming Demonstration

If you can log into the page and 1, 2, and 3 images loop display, it indicates the flask program runs well. Then you can change the file *app.py* to display videos collected by the Raspberry Pi's camera.

Here we use the *nano* built in *Raspbian* to open and edit the *app.py*. There's no need to edit in other IDEs as only commenting or uncommenting involved.

```
sudo nano app.py
```

1. Uncomment the code after opening app.py:

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

2. Add "#" at the beginning of the lines, or insert "''" at the beginning and end of the paragraph to comment. The code is changed as follows:

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

Or

```
'''
f os.environ.get('CAMERA'):
    Camera = import_module('camera_' + os.environ['CAMERA']).Camera
lse:
    from camera import Camera
'''
```

3. At last, uncomment the code *Camera* imported from *camera_pi* by deleting "#" – pay attention to delete the space after "#".

Code before change:

```
# from camera_pi import Camera
```

Code changed:

```
from camera_pi import Camera
```

4. Complete code of *app.py* changed as follows:

```
#!/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_pi import Camera

app = Flask(__name__)


@app.route('/')
```

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


def gen(camera):
    """Video streaming generator function."""
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')


@app.route('/video_feed')
def video_feed():
    """Video streaming route. Put this in the src attribute of an img tag."""
    return Response(gen(Camera()),
                    mimetype='multipart/x-mixed-replace; boundary=frame')


if __name__ == '__main__':
    app.run(host='0.0.0.0', threaded=True)
```

5. Press CTRL+X to exit after editing. A prompt will be shown asking you whether to save to not. Type in Y and press Enter to save.

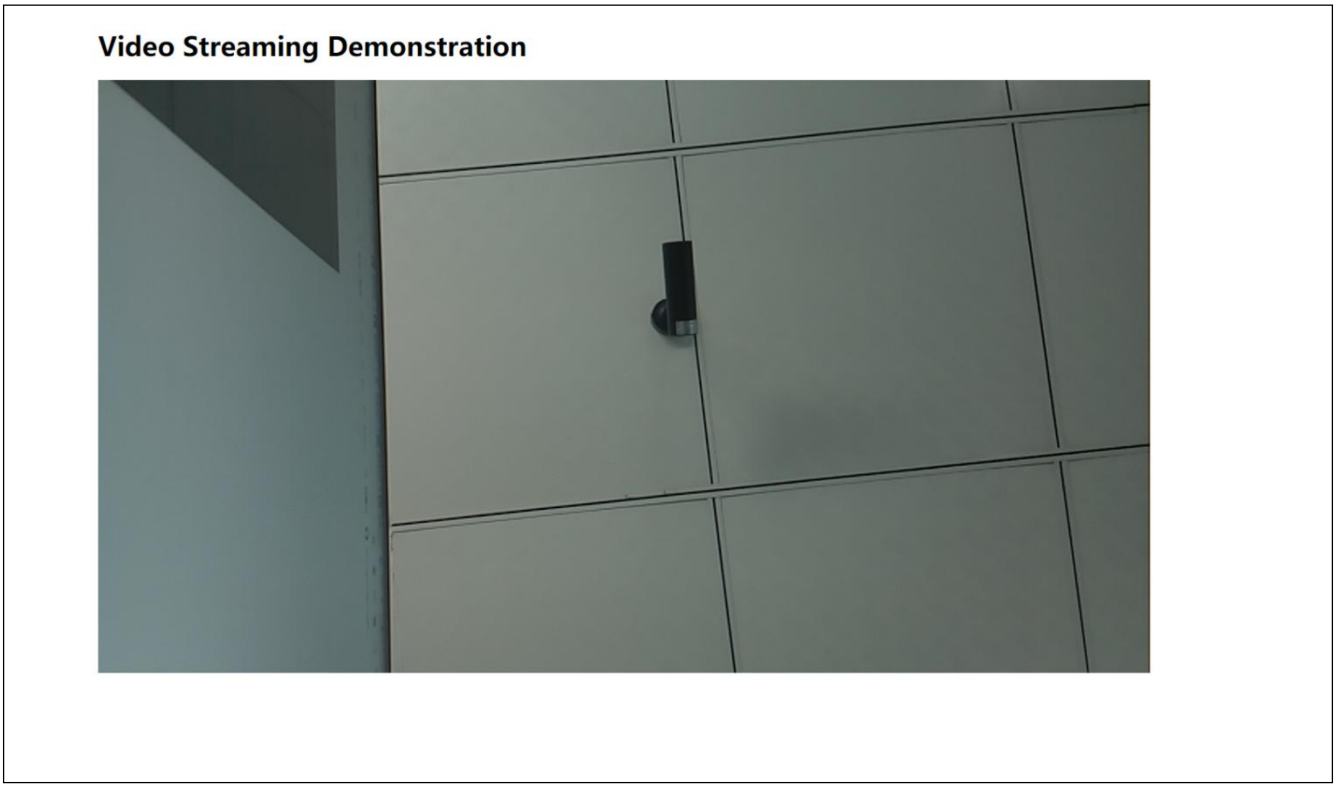
6. Next, run *app.py*.

```
sudo python3 app.py
```

7. 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:

```
192.168.3.157:5000
```

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

9. This function is based on the flask-video-streaming project from GitHub:

https://github.com/miguelgrinberg/flask-video-streaming

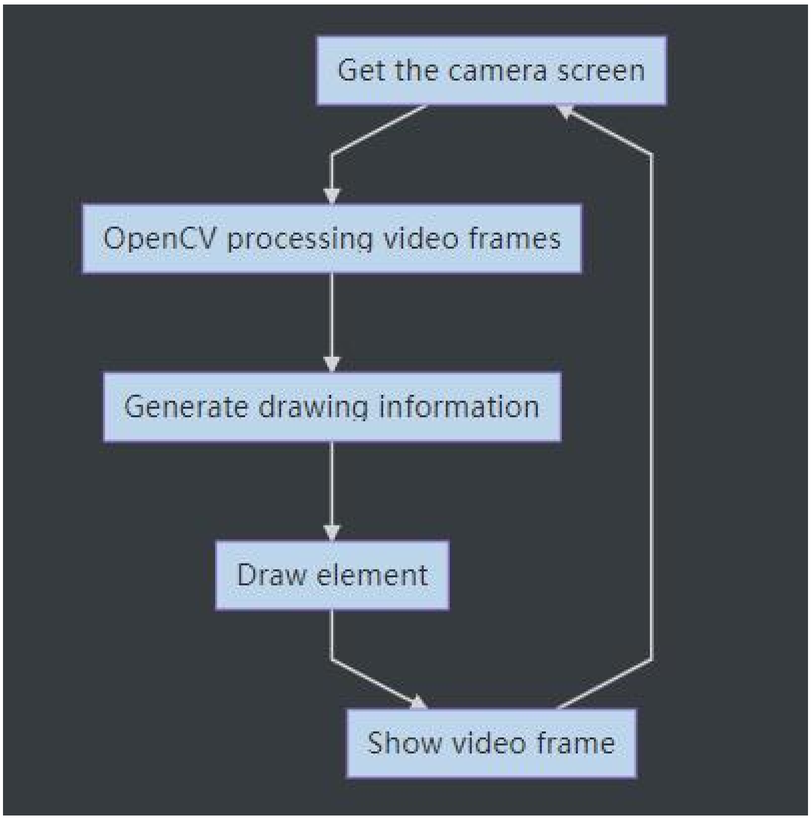
10.3 Processing Video Frames

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

Here we start with single threading for you to better understand why multithreading is needed for processing OpenCV video frames. The process for single threading is as follows:

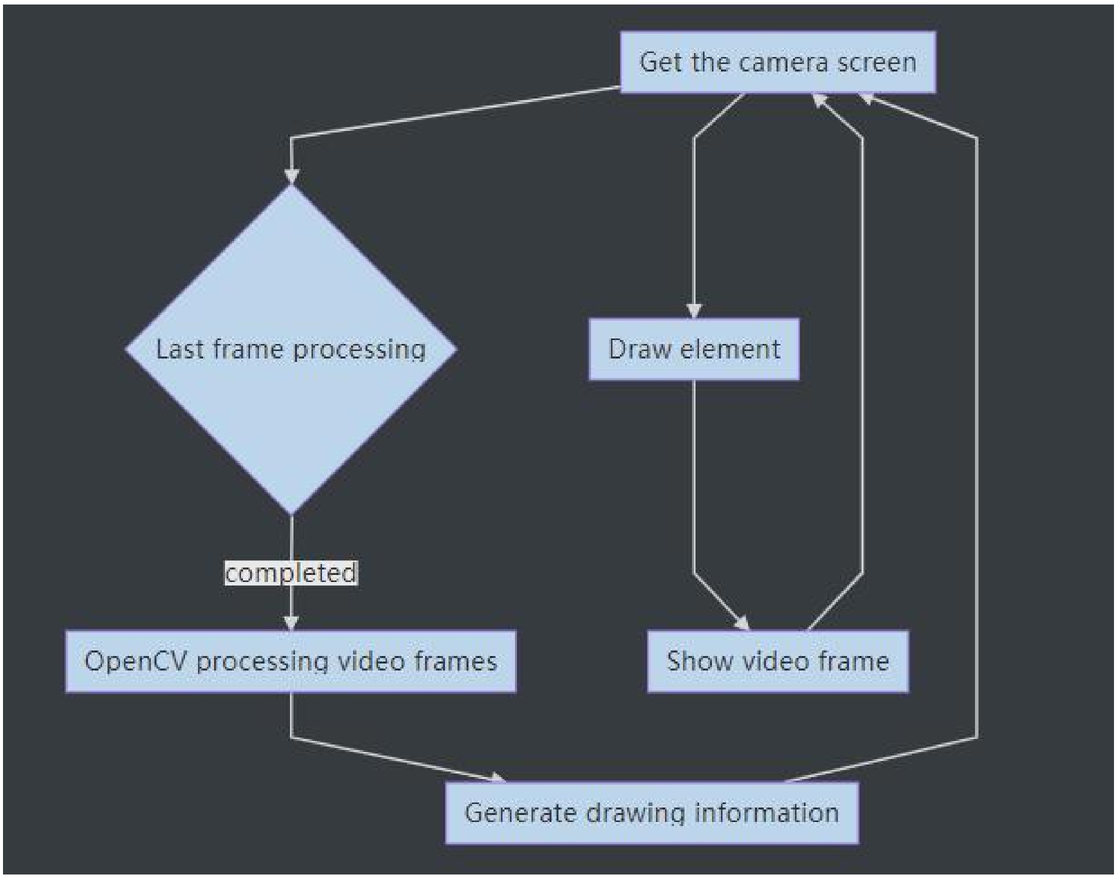


Process explanation: First, capture an image frame from the camera, analyze the frame with OpenCV, generate the information to be drawn, like the central position of the target or the text to be displayed on the screen, draw accordingly, and then display the image which has been processed and drawn on the webpage.

This whole process is inefficient as it needs to wait the OpenCV to implement the processing and display on the screen for each frame before starting the next frame processing. It may cause a stuck video transmission.

Multi-threaded video frames processing

The process is as shown below:



Process explanation: To increase frame rate, here we separate the analyzing of video frames from the collection-display process and run on background thread to generate image drawing information.

The code *camera_opencv.py* is changed as below: (the OpenCV function is not included here to explain the multi-threaded processing principle; refer to the *camera_opencv.py* file in the zip file downloaded)

```
import os
import cv2
from base_camera import BaseCamera
import numpy as np
import datetime
import time
import threading
import imutils

class CVThread(threading.Thread):
    """
    This class is used to process OpenCV's task of analyzing video frames in the background
```

```
'''
def __init__(self, *args, **kwargs):
    self.CVThreading = 0

    super(CVThread, self).__init__(*args, **kwargs)
    self.__flag = threading.Event()
    self.__flag.clear()


def mode(self, imgInput):
    "'''
    This method is used to pass in video frames that need to be processed
    '''
    self.imgCV = imgInput
    self.resume()


def elementDraw(self,imgInput):
    "'''
    Draw elements on the screen
    '''
    return imgInput

def doOpenCV(self, frame_image):
    "'''
    Add content to be processed by OpenCV here
    '''
    self.pause()


def pause(self):
    "'''
    Block the thread and wait for the next frame to be processed
    '''
    self.__flag.clear()
    self.CVThreading = 0

def resume(self):
    "'''
    Resuming the thread
    '''
    self.__flag.set()

    def run(self):
        """
        Processing video frames in a background thread
        '''
        while 1:
            self.__flag.wait()
            self.CVThreading = 1
            self.doOpenCV(self.imgCV)


class Camera(BaseCamera):
    video_source = 0

    def __init__(self):
        if os.environ.get('OPENCV_CAMERA_SOURCE'):
            Camera.set_video_source(int(os.environ['OPENCV_CAMERA_SOURCE']))
        super(Camera, self).__init__()

    @staticmethod
    def set_video_source(source):
        Camera.video_source = source

    @staticmethod
    def frames():
        camera = cv2.VideoCapture(Camera.video_source)
        if not camera.isOpened():
            raise RuntimeError('Could not start camera.')
        """
        Instantiate CVThread()
        '''
        cvt = CVThread()
        cvt.start()

        while True:
            # read current frame
            _, img = camera.read()

            if cvt.CVThreading:
                """
                If OpenCV is processing video frames, skip
                '''
            pass
        else:
            """
            If OpenCV is not processing video frames, give the thread that processes the video frame a new video frame and resume the processing thread
            '''
            cvt.mode(img)
            cvt.resume()
        """
        Draw elements on the screen
        '''
        img = cvt.elementDraw(img)

        # encode as a jpeg image and return it
        yield cv2.imencode('.jpg', img)[1].tobytes()
```

That's the code for multi-threaded OpenCV processing. In the subsequent part of introducing details of the OpenCV function, we will only explain the method of video frame processing with OpenCV and skip this part.

10.4 OpenCV Function

– First, create two *.py* files in a same folder of the Raspberry Pi (they are already included in the product download package for the Adeept Robot; refer to app.py and base_camera.py), with code as shown below:

App.py

```
#!/usr/bin/env python3

from importlib import import_module
import os
from flask import Flask, render_template, Response

from camera_opencv import Camera

app = Flask(__name__)

def gen(camera):
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

@app.route('/')
def video_feed():
    return Response(gen(Camera()),
                    mimetype='multipart/x-mixed-replace; boundary=frame')


if __name__ == '__main__':
    app.run(host='0.0.0.0', threaded=True)
```

base_camera.py

```
import time
import threading
try:
    from greenlet import getcurrent as get_ident
except ImportError:
    try:
        from thread import get_ident
    except ImportError:
        from _thread import get_ident


class CameraEvent(object):
    """An Event-like class that signals all active clients when a new frame is
    available.
    """
    def __init__(self):
        self.events = {}

    def wait(self):
        """Invoked from each client's thread to wait for the next frame."""
        ident = get_ident()
        if ident not in self.events:
            # this is a new client
            # add an entry for it in the self.events dict
            # each entry has two elements, a threading.Event() and a timestamp
            self.events[ident] = [threading.Event(), time.time()]
        return self.events[ident][0].wait()

    def set(self):
        """Invoked by the camera thread when a new frame is available."""
        now = time.time()
        remove = None
        for ident, event in self.events.items():
            if not event[0].isSet():
                # if this client's event is not set, then set it
                # also update the last set timestamp to now
                event[0].set()
                event[1] = now
            else:
                # if the client's event is already set, it means the client
                # did not process a previous frame
                # if the event stays set for more than 5 seconds, then assume
                # the client is gone and remove it
                if now - event[1] > 5:
                    remove = ident
        if remove:
            del self.events[remove]

    def clear(self):
        """Invoked from each client's thread after a frame was processed."""
        self.events[get_ident()][0].clear()


class BaseCamera(object):
    thread = None  # background thread that reads frames from camera
    frame = None  # current frame is stored here by background thread
    last_access = 0  # time of last client access to the camera
    event = CameraEvent()

    def __init__(self):
        """Start the background camera thread if it isn't running yet."""
        if BaseCamera.thread is None:
            BaseCamera.last_access = time.time()

            # start background frame thread
            BaseCamera.thread = threading.Thread(target=self._thread)
            BaseCamera.thread.start()

            # wait until frames are available
            while self.get_frame() is None:
                time.sleep(0)

    def get_frame(self):
        """Return the current camera frame."""
        BaseCamera.last_access = time.time()

        # wait for a signal from the camera thread
        BaseCamera.event.wait()
        BaseCamera.event.clear()

        return BaseCamera.frame

    @staticmethod
    def frames():
        """""Generator that returns frames from the camera."""
        raise RuntimeError('Must be implemented by subclasses.')

    @classmethod
    def _thread(cls):
        """Camera background thread."""
        print('Starting camera thread.')
        frames_iterator = cls.frames()
        for frame in frames_iterator:
            BaseCamera.frame = frame
            BaseCamera.event.set()  # send signal to clients
            time.sleep(0)

            # if there haven't been any clients asking for frames in
            # the last 10 seconds then stop the thread
              if time.time() - BaseCamera.last_access > 10:
                  frames_iterator.close()
                  print('Stopping camera thread due to inactivity.')
                  break
          BaseCamera.thread = None
```

When developing any function related with OpenCV in the following tutorial, you only need to include the respective *camera_opencv.py* file in the same folder with *app.py* and *base_camera.py* and run *app.py* in the Raspberry Pi command line.

Open a web browser on the device under the same LAN with the Raspberry Pi, enter the Raspberry Pi's IP address with the port :5000, as shown below:

```
192.168.3.157:5000
```

10.5 Real-time video display on the web page

The video display in real time by web controller is implemented based on the OpenCV function as mentioned above. A web controller is a web interface to control the robot product to perform various actions and it can be applied on any device that is able to run a browser, including PC, mobile phones, tablets, etc.

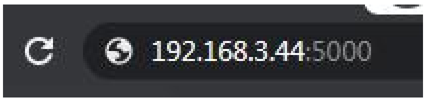
If you've completed all installations based on the instructional document, it will be quite easy to open a web controller.

- Check that your device is under the same LAN with the Raspberry Pi.
- Obtain the Raspberry Pi's IP address.
- If you terminate the Raspberry Pi auto-run program, you need to re-run the program, run the command:

```
sudo killall python3
```

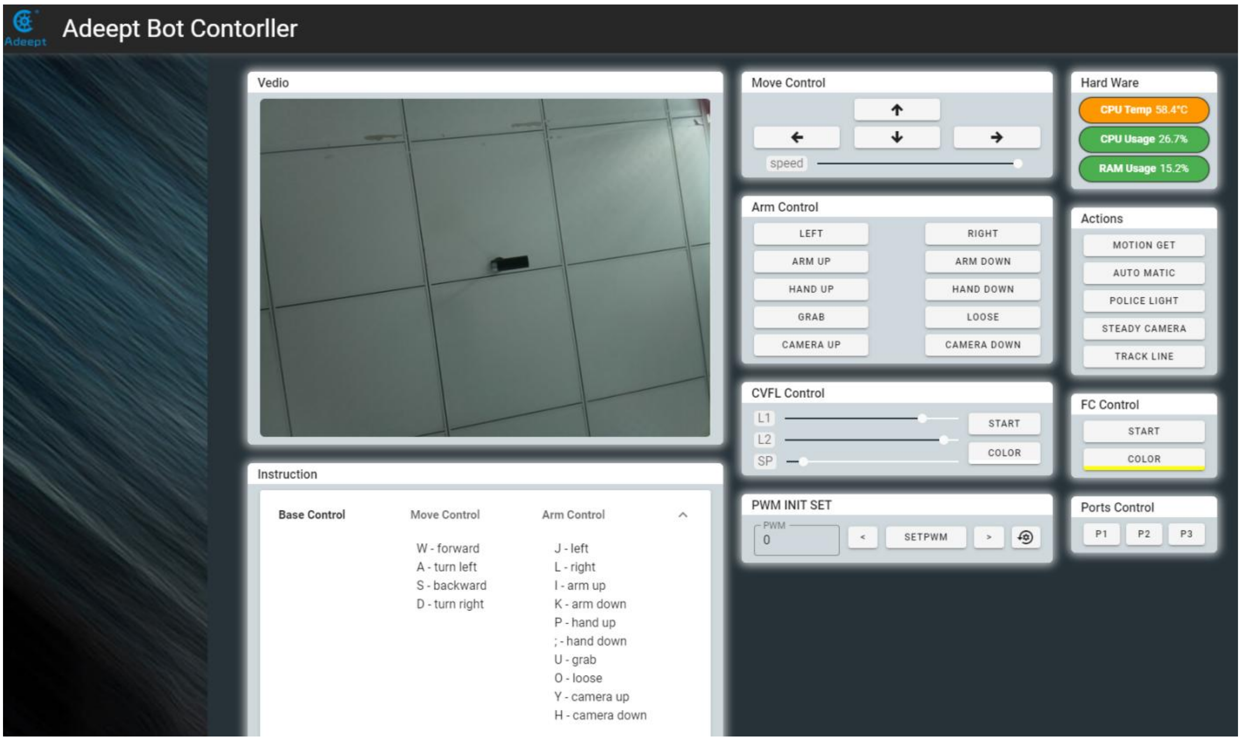
```
sudo python3 adeept_rasptank/server/webServer.py
```

- Open a web browser (recommended to use Chrome in case of any possible incompatibility with other browsers), enter the Raspberry Pi's IP address with the port :5000, for instance: 192.168.3.44:5000



192.168.3.44:5000

Then the web controller will be loaded into the browser.



W - forward
A - turn left
S - backward
D - turn right

J - left
L - right
I - arm up
K - arm down
P - hand up
; - hand down
U - grab
O - loose
Y - camera up
H - camera down

The video window on the top left corner shows the images from the Raspberry Pi camera. Modules on the web controller may vary from products. More details of the modules will be explained subsequently.