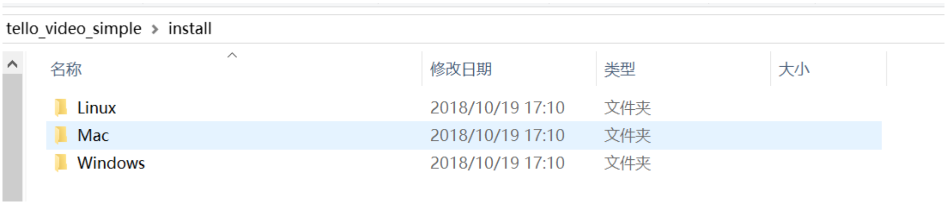
1、 Install several third-party libraries：this demo code is based on python2.To run this demo,you should install a series of dependency,including boost-python，ffmpeg,numpy,PIL,opencv and other third-party libraries.For ease of installation,this demo provide scripts that can help you install python2,pip and all the dependency needed.There are four installation scripts,which support different os,including windows32,windows64,Linux and macos.They are under the path shown below:

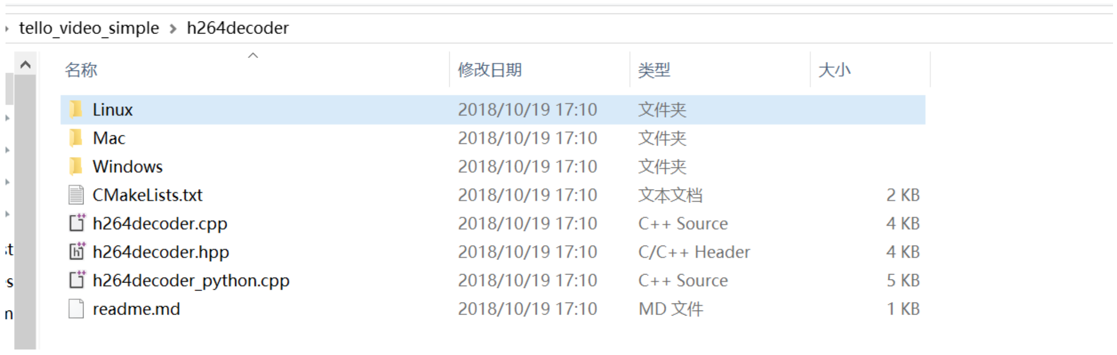

tello_video_simple > install

名称	修改日期	类型	大小
Linux	2018/10/19 17:10	文件夹	
Mac	2018/10/19 17:10	文件夹	
Windows	2018/10/19 17:10	文件夹	

So you can select one of the scripts according to your computer os.And you should copy the script to the path where main.py is located.Then run the script.
Explanation of the use of the dependencies:
pip：Assist you install other dependency.Homebrew is installed in addition if macos.
boost:Use the boost-python,in order to enable the interaction between libh264decoder based on c++ and the main program based on python.
ffmpeg:Call by the libh264decoder,to help decode the h264decoder.
numpy,:For some calculation of the Two-dimensional image matrix data.
opencv：For image processing
PIL,Tkinter：Enable the GUI and its control panel.
cmake：To compile the source code of the h264decoder,and release the dynamic link library.

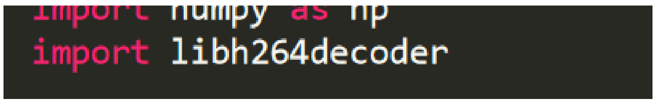
2、 Run the demo:Connect to tello's wifi,and run the main.py with python2.

3、 Documents related to the h264decoder：decoding of the h264 video data received is enable by call the dynamic link library named libh264decoder .Their path are shown below,and the source code is under this path:


tello_video_simple > h264decoder

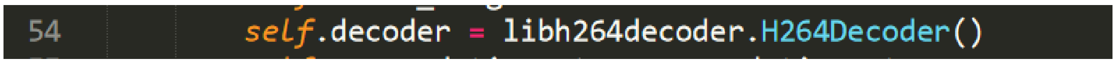
名称	修改日期	类型	大小
Linux	2018/10/19 17:10	文件夹	
Mac	2018/10/19 17:10	文件夹	
Windows	2018/10/19 17:10	文件夹	
CMakeLists.txt	2018/10/19 17:10	文本文档	2 KB
h264decoder.cpp	2018/10/19 17:10	C++ Source	4 KB
h264decoder.hpp	2018/10/19 17:10	C/C++ Header	4 KB
h264decoder_python.cpp	2018/10/19 17:10	C++ Source	5 KB
readme.md	2018/10/19 17:10	MD 文件	1 KB

4、 Call the libh264decoder in python main program：In the tello.py file, libh264decoder is imported：

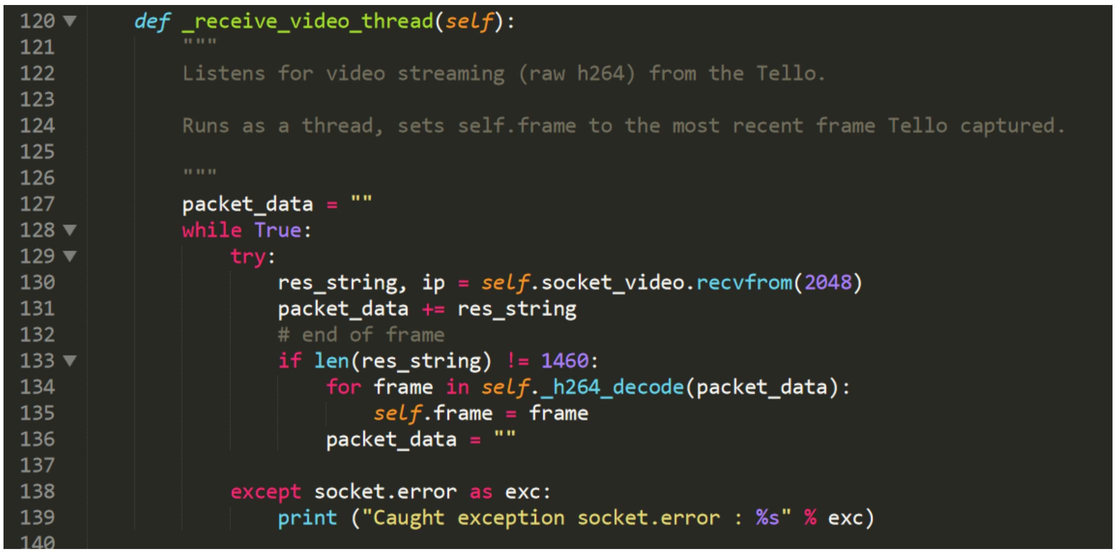

```
import numpy as np
import libh264decoder
```

Listen to the data stream with the port number 11111 through UDP.The data stream is encoded with the format of h264. Because size of encoded data of one frame image in this stream is larger than the maximum load for a single udp transmission,every segment of one frame encoded picture'data are divided into different blocks,with the unit size of 1460 bytes.And the last block of a image data is smaller than 1460 bytes.

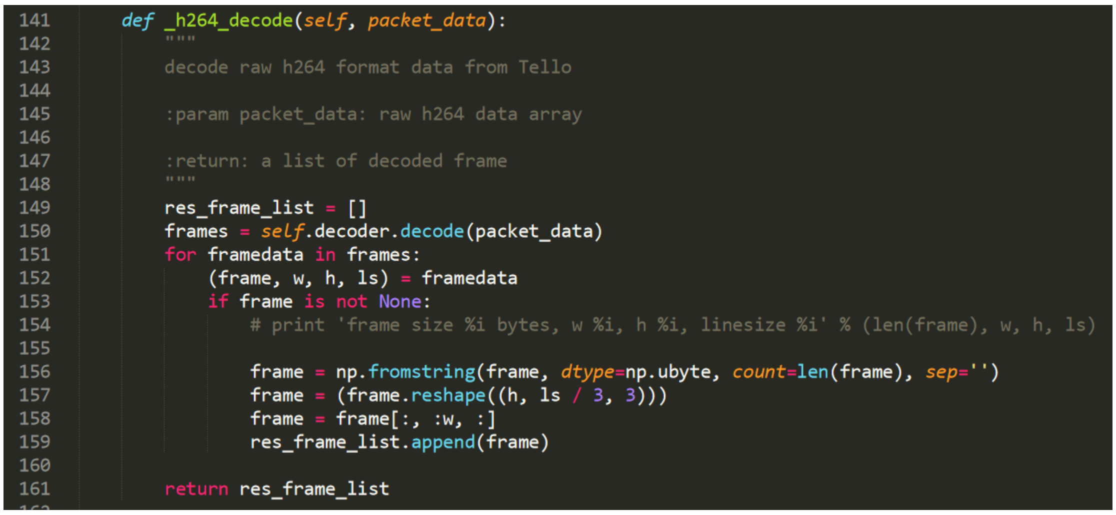
So by detecting the size of the block, it can be determined whether the data block is the end of the data of the current single image. Splicing the last block in sequence with the data blocks previously listened to，you can get the complete encoded data of a single image. Call the libh264decoder. H264Decoder,and transfer the param of a singel image data,with the format of string.The function will will return the decoded image data.
Codes related are shown below:



```
        self.decoder = libh264decoder.H264Decoder()
```

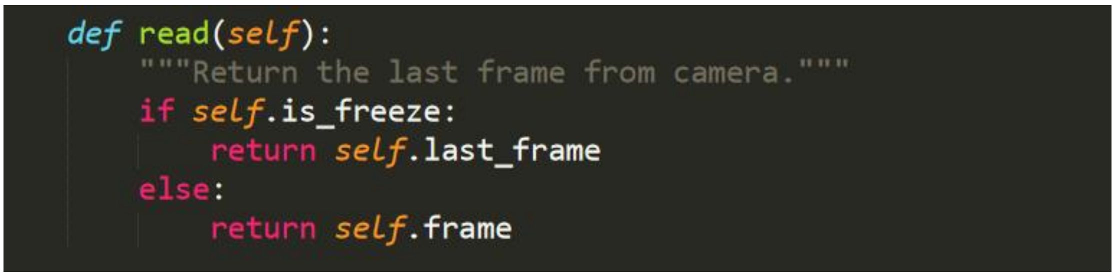


```
    def _receive_video_thread(self):
        """
        Listens for video streaming (raw h264) from the Tello.

        Runs as a thread, sets self.frame to the most recent frame Tello captured.

        """
        packet_data = ""
        while True:
            try:
                res_string, ip = self.socket_video.recvfrom(2048)
                packet_data += res_string
                # end of frame
                if len(res_string) != 1460:
                    for frame in self._h264_decode(packet_data):
                        self.frame = frame
                    packet_data = ""

            except socket.error as exc:
                print ("Caught exception socket.error : %s" % exc)
```



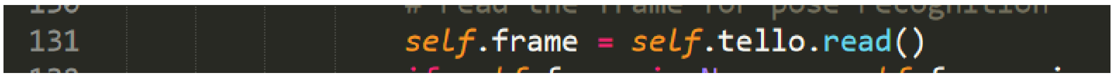
```
    def _h264_decode(self, packet_data):
        """
        decode raw h264 format data from Tello

        :param packet_data: raw h264 data array

        :return: a list of decoded frame
        """
        res_frame_list = []
        frames = self.decoder.decode(packet_data)
        for framedata in frames:
            (frame, w, h, ls) = framedata
            if frame is not None:
                # print 'frame size %i bytes, w %i, h %i, linesize %i' % (len(frame), w, h, ls)

                frame = np.fromstring(frame, dtype=np.ubyte, count=len(frame), sep='')
                frame = (frame.reshape((h, ls / 3, 3)))
                frame = frame[:, :w, :]
                res_frame_list.append(frame)

        return res_frame_list
```

5、 For the method of calling a single frame image after decoding, refer to the function in tello.py:



```
def read(self):
    """Return the last frame from camera."""
    if self.is_freeze:
        return self.last_frame
    else:
        return self.frame
```

Called in tello_control_ui.py:



```
self.frame = self.tello.read()
```