**Lumix LX 100 wifi teather project with arduino control of the orientation of the camera, together with movement detection or .object identification**

Before telling you how to operate this software, let me tell you where it is coming from…

Several years ago, I acquired a Lumix LX100, an excellent compact APN from Panasonic. Panasonic developed an Android (and Iphone) app to remotely control it for taking pictures, through its wifi capabilities.

I do not have a tablet, and the screen of a smartphone is small. Tethering with the wifi of my PC (under Windows 10) would be much more convenient. Although Panasonic developed a tethering application for PCs, the firm only released it till now for USB connection, not Wifi.

I also planned to control remotely the orientation of the camera on its photo stand through servos driven by a low cost Arduino type micro-controller. Trying to include that also within a single PC program was the objective (although, for the servo part, it was much simpler to develop an android app, eg with MIT App Inventor…). Finally, I also wanted to integrate the video stream from the LX-100 in the main panel.

Almost two years after, it works pretty nicely. I used a lot of previously developed softwares, cited below, and nevertheless faced a lot of troubles (some due to the use of Windows, some due to the camera, others due to the microcontroller I used).

Hopefully, finding solutions is possible nowadays googling internet, but this also sends you on time consuming bad leads as well.

**LX100 wifi tethering**

Pioneering and inspiring work done in <https://github.com/peci1/lumix-link-desktop/> following the work of <http://www.personal-view.com/talks/discussion/6703/control-your-gh3-from-a-web-browser-now-with-video-/p1> .

(There is also <https://github.com/Rambalac/GMaster/releases/latest> , written with visual studio, but I preferred a potentially cross-platform environment).

After many efforts, I could connect to lumix-link desktop, but only using the following trick:

1. Download ImageApp on your smartphone
2. Start the wifi on LX100 -> new connection -> remote operation
3. Connect to LX-100 with ImageApp (through QR code or entering the password)
4. Once connected, start remote operation on ImageApp
5. Then and only then, connect your PC wifi on Lumix LX-100: it becomes possible because communication with the LX-100 was already initiated by InageApp and the PC can connect to it as well
6. Start lumix-link with IP address 192.168.54.1
7. This should work. **It furthermore registers LX-100 as a wifi possible connection for Windows 10.**

**You will probably have to do that at least once to have the LX-100 wifi network registered in windows, this being eventually required if you want to connect to it.**

The reason for this complicated procedure is due to the handshake needed to connect to LX-100 server: it seems to accept initial connections on a particular port (60606 in my case), with a precise string in the GET request, as well as with a user agent being Panasonic Android.

**“GET /00AEFA2E107E/Server0/ddd HTTP/1.1\r\nHost: 192.168.54.1:60606 \r\nUser-Agent: Panasonic Android/1 DM-CP\r\n\r\n"**

Then a second initialization/handshake command is send on port 80:

“GET /cam.cgi?mode=acctrl&type=req\_acc&value=4D454930-0100-1000-8001-024D00026C8C&value2=SM-930F HTTP/1.1\r\nHost: 192.168.54.1\r\nConnection: Keep-Alive\r\nUser-Agent: Apache-HttpClient\r\n\r\n”

Then all GET commands will be accepted whatever the useragent on port 80, as for example:

GET /cam.cgi?mode=camcmd&value=recmode

The critical step is the one in bold above. You could do it using the right tool to change the useragent of your favorite web browser prior to launching lumix-tether, but it is not so practical. I also suppose it should be possible to integrate the video stream in the displayed web page of lumix-tether, but I finally decided to re-write the whole thing (which again was only made possible thanks to the previous work in particular for identifying the useful commands to send to the camera).

**Remark on the video flux**: the images in the video flux are preceded by a descriptor (?) which makes them unrecognized by VLC or other video softwares… (or I did not succeed in doing so): some pre-processing to discard these headers/descriptor is required before sending the successive images to a viewer…

Having tested the system to capture some birds bathing in the morning, I became fed up of staying in front of my PC to check their arrival. As we are preprocessing the video flux anyway, why not add a simple movement detection in the incoming flux to beep when a movement is detected?

openCV <https://opencv.org/> (and other tools) do that very well and fastly enough to keep realtime.

Finally, motion detection is nice, but you can have many “false” detections due to changes in the background: object identification can be also an option. I added this feature using the fantastic You Only Look Once (YOLO) V3 neural network to classify what you get in the video flux.

<https://pjreddie.com/darknet/yolo/>

**Java development**

Its interoperability and robustness, as well as the excellent development tools available dictated the choice of Java: I personally used Eclipse <https://www.eclipse.org/downloads/packages/release/mars/r/eclipse-ide-java-developers> .

Though I made a lot of programming in various languages (from assembler to C, Python…), I never used Java before. I asked the professionals to be indulgent on the structure (or more probably the lack of structure) of my project, as well as the poorly written classes it contains.

Integration of simple movement detection with openCV was painful because of the memory heap problem linked to the poor garbage collection by Java collector of matrices (created with Mallos in C) which are not used anymore.

When you have a loop (as required to successively treat the images of a flux ) with explicit or implicit creation of new matrices, they are not correctly freed when no more used, even through .release() function calls. The heap grows, with an eventual out of memory error after a while. Never use functions as clone(): use CopyTo() instead, initialize all the matrices outside the loop, ….). I also have these treatment in a separate (but single) thread.

The program needs anyway RAM memory (java is called with 512 Mo initial memory, and a max of 1Go).

Fortunately, core Java libraries are extremely well documented.

Bluetooth operation (through bluecove <http://www.bluecove.org/> ) is much more difficult to implement and furthermore the X64 Jar adapted for windows 64 bits was really difficult to find on the web (why it disappeared from bluecove repositories is a mystery… ).

**Arduino part:**

Having servos operated from an Arduino is in principle very simple and works flawlessly when you send the commands through the serial line, when connected to your PC.

I used a simple cheap Arduino-like circuit: <https://learn.adafruit.com/adafruit-feather-huzzah-esp8266/overview> , and a cheap HC05 circuit for Bluetooth communication with the PC.

At first, I wrote a complicated program, on the Arduino side, with software timers and interrupts, eg to send the servos positions every two second or detect the state of the connection.

**Servo problems**

It worked well, except that the servos did wiggle or made strange moves before reaching the required position or cross-talked. Long searches on Internet. Most recommendations are on the powering of the servos , some on avoiding interferences. I tried and nothing worked.

Finally, doing the right search on “Bluetooth servo problems Arduino” led me on the right track.

Bluetooth communication is made at a given baud rate: this requires the use of an internal timer and interruptions. On their side, the servos require pwm signals which are basically square “1” signals of a given duration (between typically 1 and 2 ms coding the desired position) sent every 20 ms or so. This requires also the use of a timer. Servo library can make use of a several software timers to operate several servos without problems.

**Using both Bluetooth with SoftwareSerial and servos with the standard libraries does not work.** This can have several causes, the most probable being due to their use of interruptions. Anyhow, it does not work. I have used initially SoftwareSerial because I connected HC05 to normal IO pins. My problem was then solved by using the hardware RX-TX pins of the feather circuit, sending and receiving through normal Serial library (which does not use those timers and only very short interruptions). I got rid this way of software serial and just added a switch to disconnect RX (feather) to TX (HC05) pins during program upload. Finally, for now, I also suppressed all interruption procedures from the program which became very basic.