基于树莓派的安卓视频监控（v1.1）

顺应潮流，我最近也在玩 Android 编程。买了本《疯狂 Android 讲义》，看了几天，觉得应该弄个小项目练练手，就想能不能在安卓上实现视频监控呢？

之前已经实现了 mjpg-streamer 的网络视频监控。网络上也有做 WiFi 小车的论坛，他们已经实现了安卓端的程序，里面就包含远程视频显示。下载他们的早期版本开始研究（最新版不开源），发现和 mipg-streamer 网络视频监控原理基本一样。

理解了他们的程序，就开始动手自己实现一个吧！

因为我是初学者，对安卓编程还没啥概念，因此我定下了如下目标：

1）功能尽量简单，仅仅实现一个监控功能；

2）尽量做稳定

分为三部分：mjpg-streamer 原理、jpg 编码格式、Android 程序说明。

一、mjpg-streamer 原理

我们既然用了 mjpg-streamer，应该了解一下他的实现。理解了 mjpg-streamer，后面 Android 部分的 jpg 数据帧处理就容易了。因为没有啥文档，直接看他的源代码吧。

先来看看我们打开 mjgp-streamer 的命令

./mjpg_streamer -i "./input_uvc.so -d /dev/video0 -r 640*480 -f 5" -o "./output_http.so -p 9090 -w ./www"

这条命令使用了./input_uvc.so 和./output_http.so 两个动态程序，和 windows 下的动态库是一个道理。mjpg_streamer 是主程序，负责初始化输入（从摄像头读取 jpg 数据帧）和初始化输出（http 服务器端）。

input_uvc.so 负责采集摄像头数据，input_uvc 中的 input_init 由主程序 mjpg-streamer 调用初始化，然后 mjpg-streamer 继续调用 input_run 启动 cam_thread 线程。

```
00297: /*****************************************************************************
00298: Description.: spins of a worker thread
00299: Input Value.: -
00300: Return Value: always 0
00301: *****************************************************************************/
00302: int input_run(int id)
00303: {
00304:     cams[id].pglobal->in[id].buf = malloc(cams[id].videoIn->framesizeIn);
00305:     if(cams[id].pglobal->in[id].buf == NULL) {
00306:         fprintf(stderr, "could not allocate memory\n");
00307:         exit(EXIT_FAILURE);
00308:     }
00309:
00310:     DBG("launching camera thread #%02d\n", id);
00311:     /* create thread and pass context to thread function */
00312:     pthread_create(&(cams[id].threadID), NULL, cam_thread, &(cams[id]));
00313:     pthread_detach(cams[id].threadID);
00314:     return 0;
00315: }
```

这个线程把 jpg 数据帧从 tmpbuffer 拷贝到 pglobal 中供其他线程读取。

```
00396:         /* copy JPG picture to global buffer */
00397:         pthread_mutex_lock(&pglobal->in[pcontext->id].db);
00398:
00399:         /*
00400:          * If capturing in YUV mode convert to JPEG now.
00401:          * This compression requires many CPU cycles, so try to avoid YUV format.
00402:          * Getting JPEGs straight from the webcam, is one of the major advantages of
00403:          * Linux-UVC compatible devices.
00404:          */
00405:         if(pcontext->videoIn->formatIn == V4L2_PIX_FMT_YUYV) {
00406:             DBG("compressing frame from input: %d\n", (int)pcontext->id);
00407:             pglobal->in[pcontext->id].size = compress_yuyv_to_jpeg(pcontext->videoIn, pglobal->in[pcontext->id].buf, pcontext->videoIn->framesizeIn, gquality);
00408:         } else {
00409:             DBG("copying frame from input: %d\n", (int)pcontext->id);
00410:             pglobal->in[pcontext->id].size = memcpy_picture(pglobal->in[pcontext->id].buf, pcontext->videoIn->tmpbuffer, pcontext->videoIn->buf.bytesused);
00411:         }
00412:
```

再来看 output_http，同样由 mjpg-streamer 调用初始化接口 output_init 和 output_run 启动 server_thread 线程。

```
00205: int output_run(int id)
00206: {
00207:     DBG("launching server thread #%02d\n", id);
00208:
00209:     /* create thread and pass context to thread function */
00210:     pthread_create(&(servers[id].threadID), NULL, server_thread, &(servers[id]));
00211:     pthread_detach(servers[id].threadID);
00212:
00213:     return 0;
00214: }
```

server_thread 线程负责监听 TCP 端口，等待客户端连接。

```
01106:
01107:         for(i = 0; i < max_fds + 1; i++) {
01108:             if(pcontext->sd[i] != -1 && FD_ISSET(pcontext->sd[i], &selectfds)) {
01109:                 pcfd->fd = accept(pcontext->sd[i], (struct sockaddr *)&client_addr, &addr_len);
01110:                 pcfd->pc = pcontext;
01111:
01112:                 /* start new thread that will handle this TCP connected client */
01113:                 DBG("create thread to handle client that just established a connection\n");
01114:
01115: #if 0
01116: /* commented out as it fills up syslog with many redundant entries */
01117:
01118:                 if(getnameinfo((struct sockaddr *)&client_addr, addr_len, name, sizeof(name), NULL, 0, NI_NUM
01119:                     syslog(LOG_INFO, "serving client: %s\n", name);
01120:                 }
01121: #endif
01122:                 if(pthread_create(&client, NULL, &client_thread, pcfd) != 0) {
01123:                     DBG("could not launch another client thread\n");
01124:                     close(pcfd->fd);
01125:                     free(pcfd);
01126:                     continue;
01127:                 }
01128:                 pthread_detach(client);
01129:             } ? end if pcontext->sd[i]!=-1&&... ?
01130:         } ? end for i=0;i<max_fds+1;i++ ?
```

一旦有客户端连接上来，就创建一个 client_thread 线程专门来处理和客户端的交互。client_thread 分析客户端的请求，在后面会看到在 Android 上输入的地址为：

http://192.168.1.209:9090/?action=stream

后面的 action/stream 对应下面的”GET /?action=stream”

```
00777:     /* determine what to deliver */
00778:     if(strstr(buffer, "GET /?action=snapshot") != NULL) {
00779:         req.type = A_SNAPSHOT;
00780: #ifdef WXP_COMPAT
00781:     } else if((strstr(buffer, "GET /cam") != NULL) && (strstr(buffer, ".jpg") != NULL)) {
00782:         req.type = A_SNAPSHOT;
00783: #endif
00784:         input_suffixed = 255;
00785:     } else if(strstr(buffer, "GET /?action=stream") != NULL) {
00786:         input_suffixed = 255;
00787:         req.type = A_STREAM;
00788: #ifdef WXP_COMPAT
00789:     } else if((strstr(buffer, "GET /cam") != NULL) && (strstr(buffer, ".mjpg") != NULL)) {
00790:         req.type = A_STREAM;
00791: #endif
00792:         input_suffixed = 255;
00793:     } else if((strstr(buffer, "GET /input") != NULL) && (strstr(buffer, ".json") != NULL)) {
00794:         req.type = A_INPUT_JSON;
00795:         input_suffixed = 255;
00796:     } else if((strstr(buffer, "GET /output") != NULL) && (strstr(buffer, ".json") != NULL)) {
00797:         req.type = A_OUTPUT_JSON;
00798:         input_suffixed = 255;
00799:     } else if(strstr(buffer, "GET /program.json") != NULL) {
00800:         req.type = A_PROGRAM_JSON;
00801:         input_suffixed = 255;
00802:     } else if(strstr(buffer, "GET /?action=command") != NULL) {
00803:         int len;
00804:         req.type = A_COMMAND;
00805:
00806:         /* advance by the length of known string */
00807:         if((pb = strstr(buffer, "GET /?action=command")) == NULL) {
00808:             DBG("HTTP request seems to be malformed\n");
00809:             send_error(lcfd.fd, 400, "Malformed HTTP request");
00810:             close(lcfd.fd);
00811:             return NULL;
00812:         }
```

继续看，send_stream 这个函数才真正给客户端发 jpg 数据帧。

```
00921:
00922:     switch(req.type) {
00923:     case A_SNAPSHOT:
00924:         DBG("Request for snapshot from input: %d\n", input_number);
00925:         send_snapshot(lcfd.fd, input_number);
00926:         break;
00927:     case A_STREAM:
00928:         DBG("Request for stream from input: %d\n", input_number);
00929:         send_stream(lcfd.fd, input_number);
00930:         break;
00931:     case A_COMMAND:
00932:         if(lcfd.pc->conf.nocommands) {
00933:             send_error(lcfd.fd, 501, "this server is configured to not accept commands");
00934:             break;
00935:         }
00936:         command(lcfd.pc->id, lcfd.fd, req.parameter);
00937:         break;
00938:     case A_INPUT_JSON:
00939:         DBG("Request for the Input plugin descriptor JSON file\n");
00940:         send_Input_JSON(lcfd.fd, input_number);
00941:         break;
00942:     case A_OUTPUT_JSON:
00943:         DBG("Request for the Output plugin descriptor JSON file\n");
00944:         send_Output_JSON(lcfd.fd, input_number);
00945:         break;
00946:     case A_PROGRAM_JSON:
00947:         DBG("Request for the program descriptor JSON file\n");
00948:         send_Program_JSON(lcfd.fd);
00949:         break;
00950:     case A_FILE:
00951:         if(lcfd.pc->conf.www_folder == NULL)
00952:             send_error(lcfd.fd, 501, "no www-folder configured");
00953:         else
00954:             send_file(lcfd.pc->id, lcfd.fd, req.parameter);
00955:         break;
00956:     default:
00957:         DBG("unknown request\n");
00958:     } ? end switch req.type ?
```

分为四步

1）获取 mutex，拷贝当前 jpg 数据帧到 frame 缓存中；

2）发送包头，包括 jgp 数据帧的长度（Content-Length）。这个后面 Android 编程会用到；

3）开始发送 jpg 数据帧；

4）发送 BOUNDARY，标示整个数据帧发送完成。

```
00399:
00400:         /* copy v4l2_buffer timeval to user space */
00401:         timestamp = pglobal->in[input_number].timestamp;
00402:
00403:     (1) memcpy(frame, pglobal->in[input_number].buf, frame_size);
00404:         DBG("got frame (size: %d kB)\n", frame_size / 1024);
00405:
00406:         pthread_mutex_unlock(&pglobal->in[input_number].db);
00407:
00408:         /*
00409:          * print the individual mimetype and the length
00410:          * sending the content-length fixes random stream disruption observed
00411:          * with firefox
00412:          */
00413:     (2) sprintf(buffer, "Content-Type: image/jpeg\r\n" \
00414:                 "Content-Length: %d\r\n" \
00415:                 "X-Timestamp: %d.%06d\r\n" \
00416:                 "\r\n", frame_size, (int)timestamp.tv_sec, (int)timestamp.tv_usec);
00417:         DBG("sending intemdiate header\n");
00418:         if(write(fd, buffer, strlen(buffer)) < 0) break;
00419:
00420:         DBG("sending frame\n");
00421:     (3) if(write(fd, frame, frame_size) < 0) break;
00422:
00423:         DBG("sending boundary\n");
00424:         sprintf(buffer, "\r\n--" BOUNDARY "\r\n");
00425:     (4) if(write(fd, buffer, strlen(buffer)) < 0) break;
00426:     } ? end while !pglobal->stop ?
00427:
```

至此，我们了解到每个 jpg 的数据帧是这样构成的：

```
// ------------ JPG ------------
// Content-Type: image/jpeg\r\n
// Content-Length: %d\r\n
// X-Timestamp: %d.%06d\r\n
// ... JPG Frame ...
// \r\n--BOUNDARY\r\n
```

二、jpg 图片编码

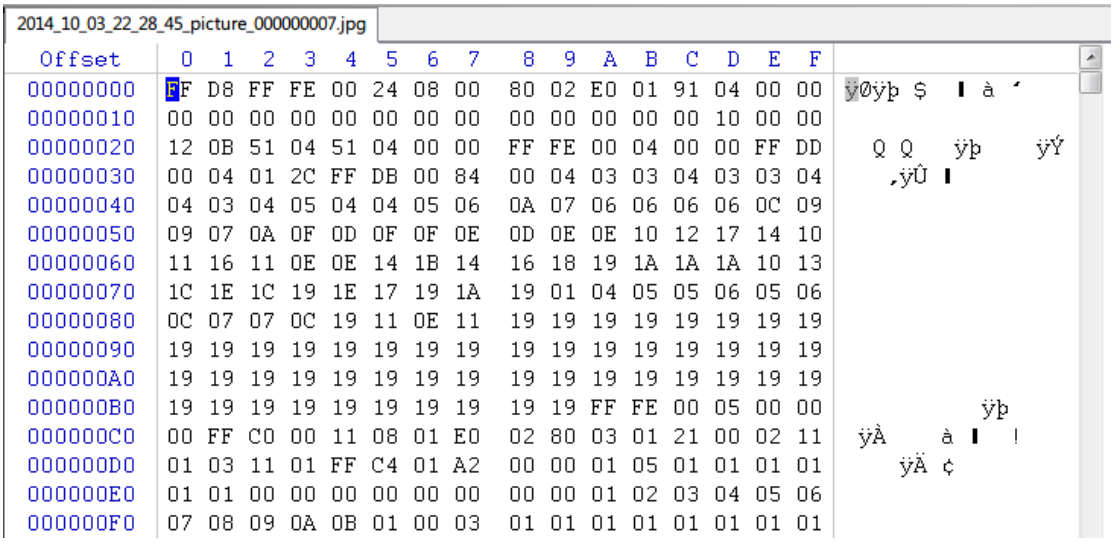
这个网上有详细的说明，我这里只说明两点，jpg 编码是以 0xff 开头的，是以 0xff 0xd9 结束的。

想进一步了解的看这里：

http://www.cnblogs.com/leaven/archive/2010/04/06/1705846.html

使用 winhex 随便打开一张 jpg 图片查看头和尾。

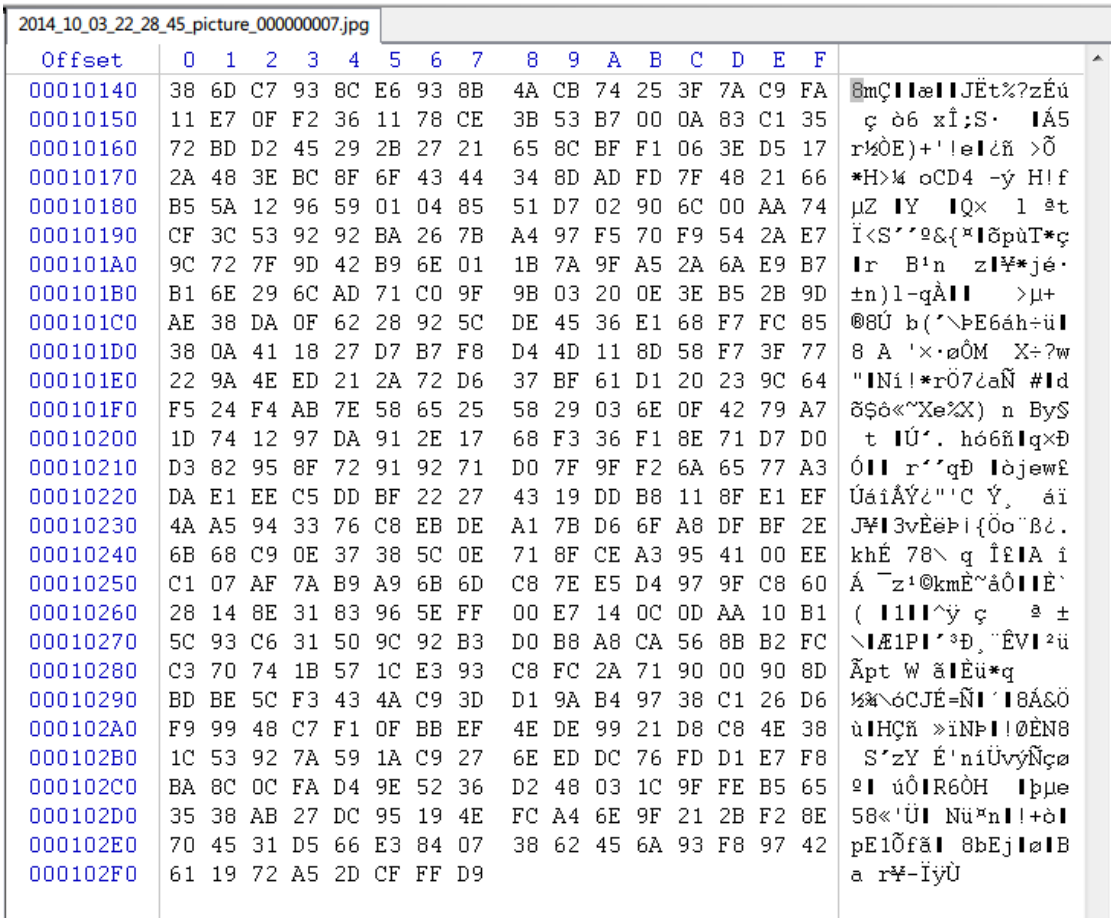
jpg 帧头截图，以 0xff 开始



2014_10_03_22_28_45_picture_000000007.jpg

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	FF	D8	FF	FE	00	24	08	00	80	02	E0	01	91	04	00	00
00000010	00	00	00	00	00	00	00	00	00	00	00	00	00	10	00	00
00000020	12	0B	51	04	51	04	00	00	FF	FE	00	04	00	00	FF	DD
00000030	00	04	01	2C	FF	DB	00	84	00	04	03	03	04	03	03	04
00000040	04	03	04	05	04	04	05	06	0A	07	06	06	06	06	0C	09
00000050	09	07	0A	0F	0D	0F	0F	0E	0D	0E	0E	10	12	17	14	10
00000060	11	16	11	0E	0E	14	1B	14	16	18	19	1A	1A	1A	10	13
00000070	1C	1E	1C	19	1E	17	19	1A	19	01	04	05	05	06	05	06
00000080	0C	07	07	0C	19	11	0E	11	19	19	19	19	19	19	19	19
00000090	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19
000000A0	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19
000000B0	19	19	19	19	19	19	19	19	19	19	FF	FE	00	05	00	00
000000C0	00	FF	C0	00	11	08	01	E0	02	80	03	01	21	00	02	11
000000D0	01	03	11	01	FF	C4	01	A2	00	00	01	05	01	01	01	01
000000E0	01	01	00	00	00	00	00	00	00	00	01	02	03	04	05	06
000000F0	07	08	09	0A	0B	01	00	03	01	01	01	01	01	01	01	01

jpg 帧尾截图，以 0xff 0xd9 结束



2014_10_03_22_28_45_picture_000000007.jpg

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00010140	38	6D	C7	93	8C	E6	93	8B	4A	CB	74	25	3F	7A	C9	FA
00010150	11	E7	0F	F2	36	11	78	CE	3B	53	B7	00	0A	83	C1	35
00010160	72	BD	D2	45	29	2B	27	21	65	8C	BF	F1	06	3E	D5	17
00010170	2A	48	3E	BC	8F	6F	43	44	34	8D	AD	FD	7F	48	21	66
00010180	B5	5A	12	96	59	01	04	85	51	D7	02	90	6C	00	AA	74
00010190	CF	3C	53	92	92	BA	26	7B	A4	97	F5	70	F9	54	2A	E7
000101A0	9C	72	7F	9D	42	B9	6E	01	1B	7A	9F	A5	2A	6A	E9	B7
000101B0	B1	6E	29	6C	AD	71	C0	9F	9B	03	20	0E	3E	B5	2B	9D
000101C0	AE	38	DA	0F	62	28	92	5C	DE	45	36	E1	68	F7	FC	85
000101D0	38	0A	41	18	27	D7	B7	F8	D4	4D	11	8D	58	F7	3F	77
000101E0	22	9A	4E	ED	21	2A	72	D6	37	BF	61	D1	20	23	9C	64
000101F0	F5	24	F4	AB	7E	58	65	25	58	29	03	6E	0F	42	79	A7
00010200	1D	74	12	97	DA	91	2E	17	68	F3	36	F1	8E	71	D7	D0
00010210	D3	82	95	8F	72	91	92	71	D0	7F	9F	F2	6A	65	77	A3
00010220	DA	E1	EE	C5	DD	BF	22	27	43	19	DD	B8	11	8F	E1	EF
00010230	4A	A5	94	33	76	C8	EB	DE	A1	7B	D6	6F	A8	DF	BF	2E
00010240	6B	68	C9	0E	37	38	5C	0E	71	8F	CE	A3	95	41	00	EE
00010250	C1	07	AF	7A	B9	A9	6B	6D	C8	7E	E5	D4	97	9F	C8	60
00010260	28	14	8E	31	83	96	5E	FF	00	E7	14	0C	0D	AA	10	B1
00010270	5C	93	C6	31	50	9C	92	B3	D0	B8	A8	CA	56	8B	B2	FC
00010280	C3	70	74	1B	57	1C	E3	93	C8	FC	2A	71	90	00	90	8D
00010290	BD	BE	5C	F3	43	4A	C9	3D	D1	9A	B4	97	38	C1	26	D6
000102A0	F9	99	48	C7	F1	0F	BB	EF	4E	DE	99	21	D8	C8	4E	38
000102B0	1C	53	92	7A	59	1A	C9	27	6E	ED	DC	76	FD	D1	E7	F8
000102C0	BA	8C	0C	FA	D4	9E	52	36	D2	48	03	1C	9F	FE	B5	65
000102D0	35	38	AB	27	DC	95	19	4E	FC	A4	6E	9F	21	2B	F2	8E
000102E0	70	45	31	D5	66	E3	84	07	38	62	45	6A	93	F8	97	42
000102F0	61	19	72	A5	2D	CF	FF	D9								

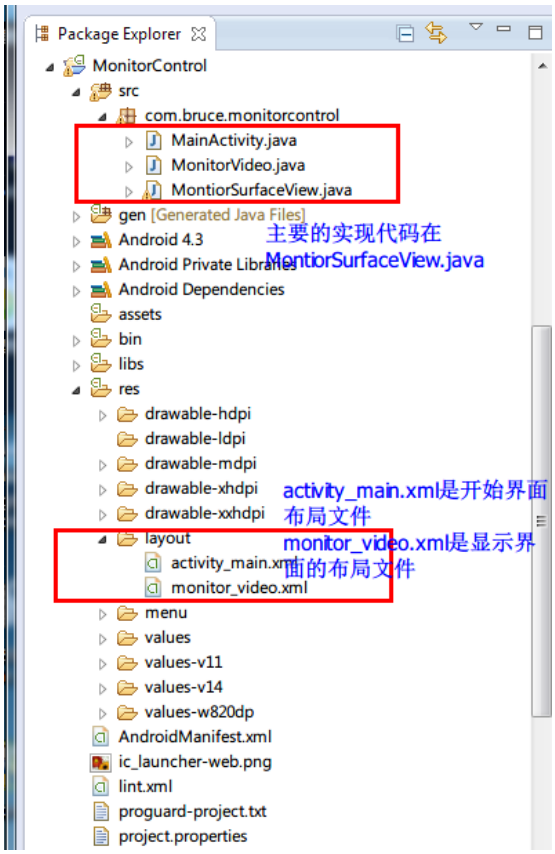
三、Android 编程

实现的界面如下，开始界面提供一个设备地址 EditText 和两个按钮，设备地址改完自动保存，下次打开会自动载入。

点击登录，程序自动连接设备地址，成功后显示 jpg 数据流到 SurfaceView 控件上。这

里用到两个线程， 一个线程下载 jpg 数据帧，然后保存保存到 BlockingQueue 中；另外一个是 SurfaceView 控件自己实现的线程，这个线程从 BlockingQueue 中获取数据，然后刷新界面。注意 SurfaceView 已经实现了双缓冲，无需我们手动干预。

代码看工程，下面是工程的文件简要介绍：



下面的截图显示帧数和下载数据的速度。

关于 Android 程序的说明：

1）大家可能已经注意到我这里 jpg 下载的速度非常快，达到 700KB/s 左右。

其实我设置的参数并不高，jpg 帧数 10fps，jpg 分辨率为最小的 640*480。大家在用 4Mbps 宽带看 720p 网络视频一般不卡，网络视频下载速度大概在 400KB/s，而实际的视频码率比这个低的多。为什么我们这里的速度达到 700KB/s 呢？这是因为他们的编码方式的区别，jpg 不是用来传输视频的，h.264 传输视频比较合适，但是使用 jpg 传输比较简单，这也是网络上大部分开源视频监控都使用 mjpg-streamer 的原因。

如果你的路由器用的人比较多，或者速度比较慢，那么你会感觉到视频画面延时很严重。我一直都以为是程序问题，优化了好几天，将原始的程序效率差的地方替换掉（原始程序使用 for 循环一个字节一个字节的处理。其实完全没必要，因为我们从 mjpg-streamer 数据包的头就能得到 jpg 的大小，只需要找到 jpg 的头部，然后接收这么多的数据即可）。但是这部分改完后效果不明显，最后实在没办法了，我把之前买的 30 块包邮的路由器（600MHz 的 7620n CPU，300Mbps 的无线）拿出来单独测试，一测还真跟路由器有关。速度一下就上去了，画面流畅了，基本上没延时了。

2）从上面可以看出，视频传输使用 mjpg-streamer 效率不高，后面考虑使用专门的视频传输协议：H.264 实现视频传输。

3）原始程序 BUG 比较多，这个程序我在 MX 上经过几天的测试没发现什么问题，如果你有兴趣玩玩，遇到问题请反馈给我，谢谢！

4）为了方便大家，这里提供原始版的 WiFi 小车源代码和我已经实现的源代码，如果你对代码有什么疑问，欢迎交流。

5）这个程序比较简单，适合入门。我从刚学 Android 编程到现在写完这个小程序用了差不多 2 个星期的时间，都是利用晚上的时间搞的。所以我感觉 Android 编程入门还是蛮简单的，只要你花时间，肯定能学会，这玩意比嵌入式容易多了。在编写的过程中你会发现 Google 的 API 做的太完善了，功能基本都给你做好了，你只需要调用一下就可以了，像在搭积木，经验足的就是使用这些 API 比你熟悉，呵呵。

6）存在的问题：下载线程和显示线程没有安全的终止，退出时直接强制关闭当前的 Activity，可能有问题。

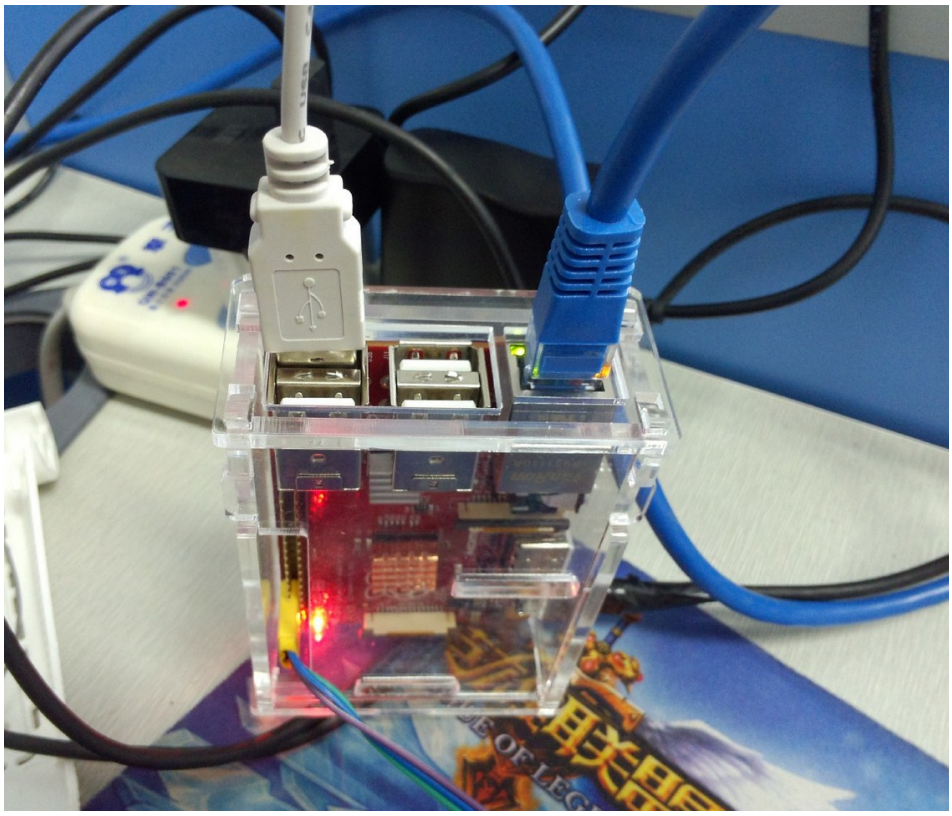
7）我的摄像头输出 jpg 格式支持的最小分辨率为 640*480

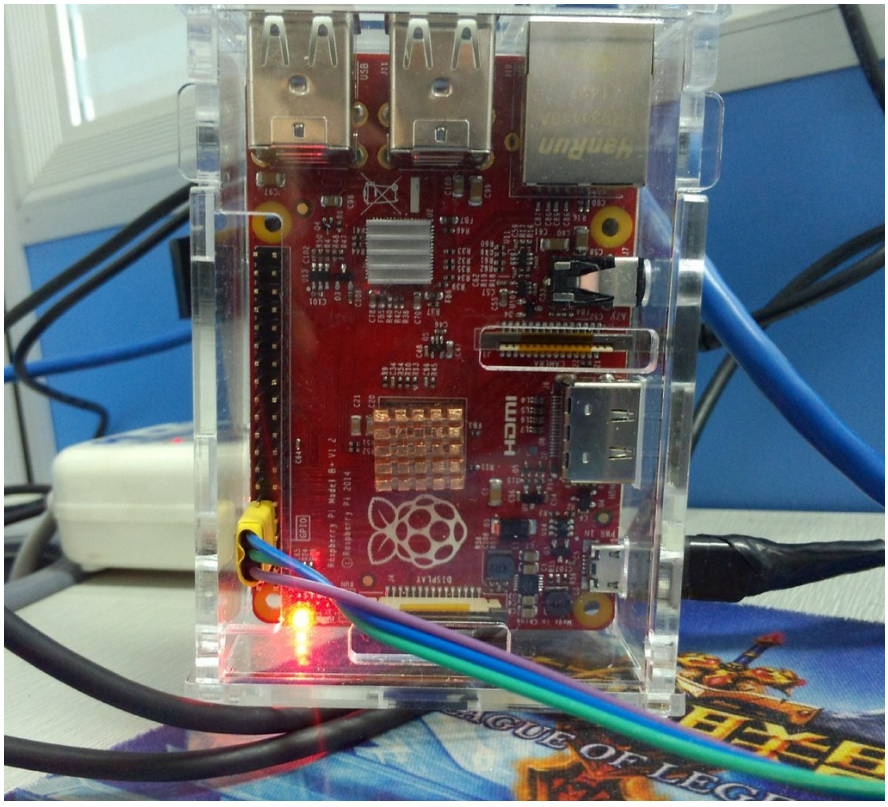
使用命令查看摄像头信息：

sudo luvcview -d /dev/video0 -L

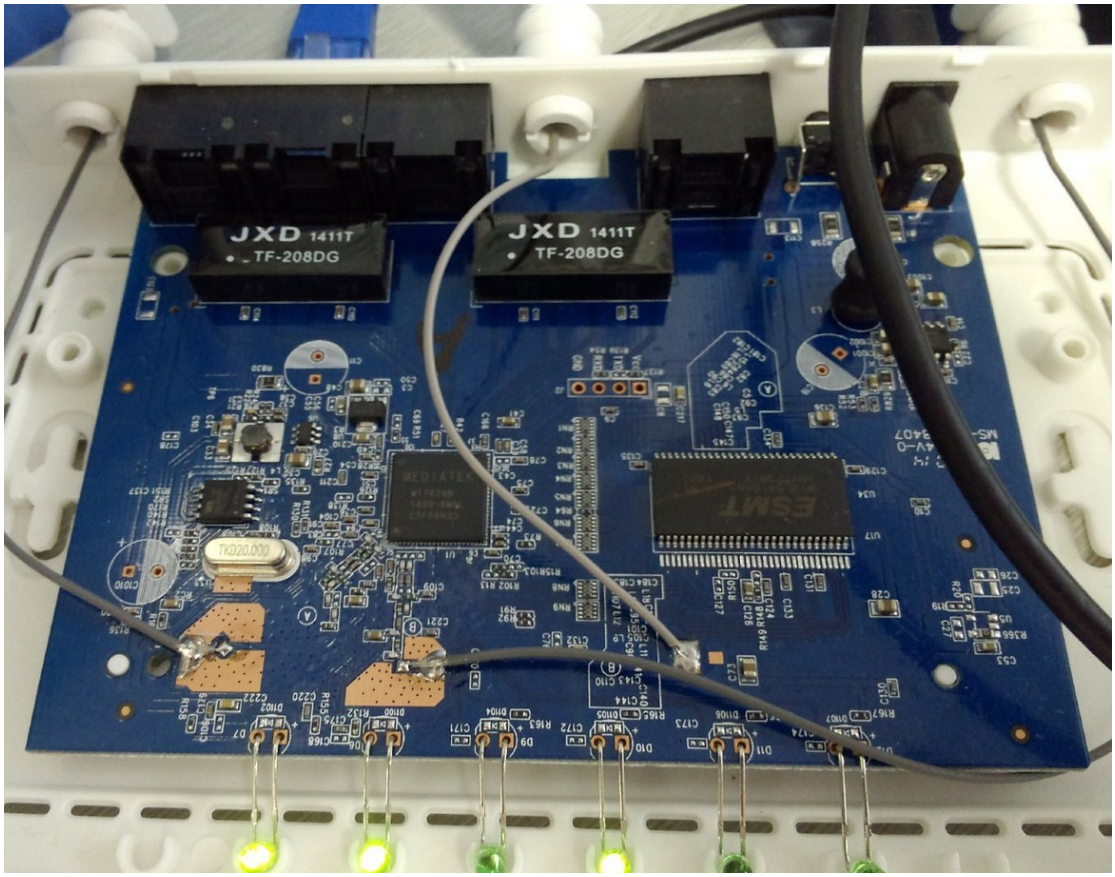
```
{ pixelformat = 'MJPG', description = 'MJPEG' }
{ discrete: width = 640, height = 480 }
       Time interval between frame: 1/60, 1/30, 2/55, 1/25, 2/45, 1/20, 2/35, 1/15, 2/25, 1/10, 2/15, 1/5,
{ discrete: width = 720, height = 480 }
       Time interval between frame: 1/60, 1/30, 2/55, 1/25, 2/45, 1/20, 2/35, 1/15, 2/25, 1/10, 2/15, 1/5,
{ discrete: width = 800, height = 600 }
       Time interval between frame: 1/30, 2/55, 1/25, 2/45, 1/20, 2/35, 1/15, 2/25, 1/10, 2/15, 1/5,
{ discrete: width = 1024, height = 576 }
       Time interval between frame: 1/30, 2/55, 1/25, 2/45, 1/20, 2/35, 1/15, 2/25, 1/10, 2/15, 1/5,
{ discrete: width = 1024, height = 768 }
       Time interval between frame: 1/30, 2/55, 1/25, 2/45, 1/20, 2/35, 1/15, 2/25, 1/10, 2/15, 1/5,
{ discrete: width = 1280, height = 720 }
       Time interval between frame: 1/30, 2/55, 1/25, 2/45, 1/20, 2/35, 1/15, 2/25, 1/10, 2/15, 1/5,
{ discrete: width = 1280, height = 960 }
       Time interval between frame: 1/30, 2/55, 1/25, 2/45, 1/20, 2/35, 1/15, 2/25, 1/10, 2/15, 1/5,
{ discrete: width = 1280, height = 1024 }
       Time interval between frame: 1/30, 2/55, 1/25, 2/45, 1/20, 2/35, 1/15, 2/25, 1/10, 2/15, 1/5,
```

最后附一些测试图，欣赏一下吧。
树莓派通过有线连接路由器，手机通过 WiFi 连接无线路由器。





30 块包邮的路由器，CPU 7620N（600MHz），300Mpbs，1MB Flash，8MB 的 SDRAM。有时间换个 16MB 的 Flash，64MB 的 SDRAM，刷个 Openwrt，立马高大上☺



摄像头通过 USB 和树莓派连接。