

# Double\_720 详细设计说明书

拟写： \_\_\_\_\_ 日期： \_\_\_\_\_

审评人： \_\_\_\_\_ 日期： \_\_\_\_\_

批准： \_\_\_\_\_ 日期： \_\_\_\_\_

# 文件控制

## 变更记录

日期	修改人	版本	更改说明

## 审阅

日期	审阅者	意见

# 目录

1 引言.....	1
1.1 编写目的.....	1
1.2 背景.....	1
1.3 参考资料.....	1
1.4 专业术语说明.....	1
2 设计概述.....	1
2.1 任务和目标.....	1
2.2 设计要求 .....	1
3 需求规定.....	1
3.1 界面要求 .....	1
3.2 功能需求.....	1
3.3 通信协议.....	1
4 运行环境规定.....	2
4.1 设备.....	2
4.2 接口.....	2
5.代码设计.....	2
5.1 主要技术.....	2
5.2 设计难点.....	2
5.3 代码流程.....	2
5.4 函数功能.....	2
5.5 代码设计说明.....	2

6.实施计划.....	2
6.1 限制.....	2
6.2 实施内容和进度.....	2
6.3 实施条件和措施.....	2
7 测试计划.....	2
7.1 测试方案.....	2
7.2 测试结果.....	2
7.3 修改类容.....	2
8 验收标准.....	2

## 1. 引言

### 1.1 编写目的

对飞控项目的规格和要求做一些详细的说明和讲解,让开发人员能够更加清晰的明白该如何的设计 APP。文档主要包括了: APP 中应该实现的功能、飞机的控制指令、代码的设计模块、主要应用技术、初步的测试步骤、性能需求等。在实际开发的过程中能够随时做好项目的总结的,以便以后的修改和更深层次的开发。

### 1.2 背景

(1) 待开发软件的名称: Double\_720

(2) 待开发软件的描述: 该项目主要用于无人机的航拍中。无人机上的 WIFI 摄像头作为一个 WIFI 热点,通过智能手机连接到该热点,打开此 APP,即可获取 WIFI 摄像头拍摄的视频等信息,同时可以通过该 APP 对无人机进行飞行操控。WIFI 摄像头有两个服务器,一个是用来传输视频的视频服务器(RTSP 协议);另一个是用来传输命令的 UDP 协议。

WIFI 摄像头的 SSID: 以 Skycam 开头,例如: Skycam\_ab-9f-f4。

密码: 12345678。

(3) 适用人群: 玩具类,使用于 12 岁以上的人群。

(4) 开发人员: 周斌,邹宇。

(5) 项目的委托单位和开发单位: 武汉新联科技。

### 1.3 参考资料

FFmpeg 相关的文档(雷霄华相关的博客,FFmpeg 解码群,开源中国)

### 1.4 专业术语说明

FFmpeg: 是一套可以用来记录、转换数字音频、视频,并能将其转化为流的开

源计算机程序。

## 2. 设计概述

### 2.1 任务和目标

按照文档的要求在规定的时间内做好对 APP 的开发,APP 要求做到功能完善,操作界面简单明了,易于操作。对控制指令的准确性、录像、拍照、画面的延迟、是否有马赛等方面要做好测试和记录。

### 2.2 设计要求

(1) 4 个不同的摄像头模式: a.广角摄像头; b.非广角摄像头; c.主屏幕广角摄像头, 副屏幕非广角摄像头; d.主屏幕非广角摄像头, 副屏幕广角摄像头。

(2) 要求主屏幕显示的画面是 720P。

(3) 在切换摄像头模式的时候, 会等待三秒, 然后进入到下一个模式。

(4) 延迟在 500ms 以内; 图像显示无马赛克和破图现象, 在距离过远时, 图像卡死, 在摄像头向手机靠近时, 画面恢复, 延迟在 500ms 内。

(5) 主要功能: 在线 OTA 升级, 摄像头版本查询, 拍照, 录像等。

(6) 摇杆控制命令和实际的控制命令一致, 能够控制飞机的飞行。

(7) 在画面静止的时候, 有数据的发送和接收, 不会出现卡死的情况。

## 3. 需求规定

### 3.1 界面要求

图 1:

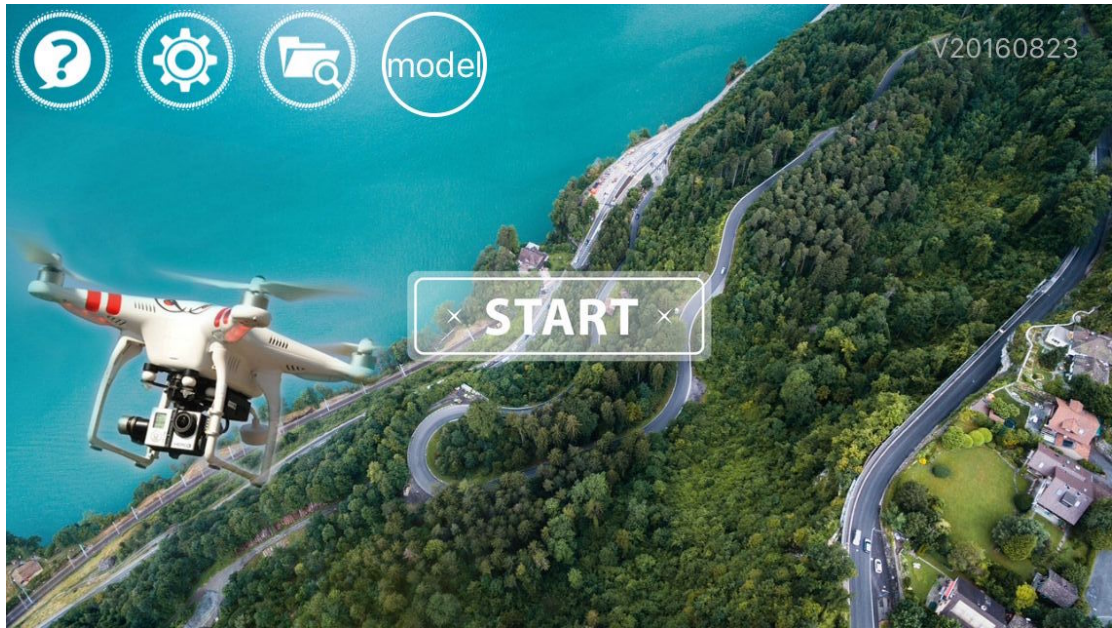
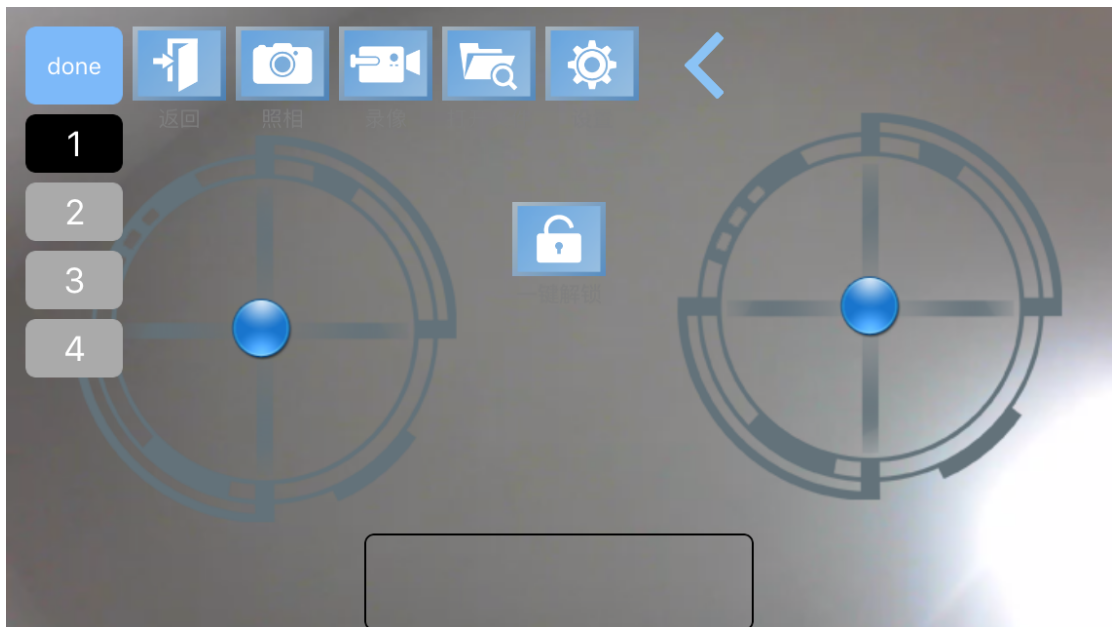


图 2:










图一是进入 APP 时显示的画面，在 WIFI 没有连接的情况下点击 START 按钮不会进入显示画面；连接到指定的 WIFI 摄像头后，等待三秒点击 START 按钮可以进入到图 2 所示的视频显示界面。





图 2 所示的画面用来显示实时的视频影像，上方若干按钮，左 1 按钮可以将 1-4 的按钮进行收放，右 1“<”按钮可以将上面的按钮进行收放。左右各有一个摇杆，

用来控制飞机的油门和飞行方向。进入时不显示摇杆，到手触碰到屏幕时，摇杆显示，离开时隐藏。底部的黑框用来接受 WIFI 摄像头返回的数据。当屏幕超过一分钟没有被触碰，隐藏所有的控制按钮，触碰一下就显示。

3.2 功能要求

	摇杆控制	点击后进入操作讲解界面。
	设置按钮	点击后进入设置界面：  (1) 设置网络 - 点击后到手机的网络设置中，选择 WIFI 摄像头的 WIFI；  (2) 无头模式 - 开启或关闭无头模式；  (3) 影像翻转 - 设置手机影像 180 度翻转；  (4) 右手操作 - 切换手机油门于左摇杆或右摇杆；  (5) 版本号查询 - 查询硬件的版本参数；  (6) OTA 版本升级 - 一键升级摄像头版本。
	文件夹	查看摄像头拍摄的照片和录制的视频。
	模式按钮 (UDP 和 TCP)	选择影像传输的模式，UDP 模式和 TCP 模式，用于测试哪种模式下影像传输的效果更好。
	返回按钮	返回到主界面。
	拍照按钮	截取手机端当前的视频截图，保存到手机中。
	录像按钮	录制手机端收到的视频，保存到手机中。



	文件按钮	查看摄像头拍摄的照片和录制的视频。
	设置按钮	<p>(1) 无头模式 - 开启或关闭无头模式；</p> <p>(2) 一键平衡 - 开启或关闭无头模式，油门为 0 时才能发送此平衡值；</p> <p>(3) 影像翻转 - 设置手机影像翻转 180 度；</p> <p>(4) 右手操作 - 切换手机油门于左摇杆或右摇杆；</p> <p>(5) 定高模式 - 定高模式的开启或关闭，若开启，中间则放一个解锁按钮；</p> <p>(6) 打开微调 - 开启或关闭微调按钮。</p>
	摇杆控制	<p>按住中间的按钮，可上下左右随意移动；在摇杆上方显示相对的坐标，并将坐标数据发送给 WIFI 摄像头。</p> <p>摇杆 1-油门 + 方向舵</p> <p>摇杆 2-副翼（侧飞）+ 升降舵（前后）</p>
	微调按钮	<p>细微调整遥控数据，按两端的按钮可进行微调，按一次其调整值为+/-2，并会发出按键声音，最高限制值为输出值的 50%。主要是来调节偏航微调，俯仰微调，侧滚微调。如果按钮调整至中间时，则会发出归零声音。</p>

摇杆说明：

通过右手模式按钮来确定油门的左右手模式，以左手模式为例：

左手摇杆	油门	左手摇杆的上下为油门，控制飞行器的升降。最下端的油门值为 0，最上端的油门之最大。如果为定高模式，油门不具备回弹功能，推倒什么位置就停到什么位置。APP 启动时，油门应放到最低位置。若关闭定高模式，油门具有回弹功能。
	偏航方向	左手摇杆的左右为偏航，控制飞行器偏转。偏航摇杆具有回弹功能，手指松开，回弹到偏航的中心位置。
右手摇杆	俯仰升降舵	右手摇杆的上下为俯仰，控制飞行器俯仰（前后）运动。俯仰摇杆具有回弹功能，手指松开后，回弹到俯仰的中心位置。
	侧滚	右手摇杆的上下为侧滚，控制飞行器侧滚（左右）运动。侧滚摇杆具有回弹的功能，手指松开后，回弹到侧滚的中心位置。

### 3.3 通信协议

串口发送数据的格式（串口波特率 19200，一个起始位，一个停止位），APP 每 40ms 发送一个数据包，每个包为 8 个字节。

BYTE[0]	数据头	固定为 0x66。
BYTE[1]	侧滚 roll AIL 副翼	数据范围 0x00-0xff，中间值 0x7f，最左边为 0x00，最右为 0xff。
BYTE[2]	俯仰 pitch	数据范围 0x00-0xff，中间值 0x7f，后最大为 0x00，

	ELE 升降舵	前最大为 0xff。
BYTE[3]	油门 throttle	数据范围 0x00-0xff，中间值 0x7f，最下为 0x00，最上为 0xff。
BYTE[4]	偏航 yaw 方向舵	数据范围 0x00-0xff，中间值 0x7f，最左边为 0x00，最右边 0xff。
BYTE[5]	标志位	Bit0 – 保留位； Bit1 – 一键起飞（油门解锁），先置 1，1s 后置 0； Bit2 – 一键着陆，先置 1，1s 后置 0； Bit3 – 飞行器紧急停止，先置 1，1s 后置 0； Bit4 – 定高模式开启或关闭； Bit5 – 无头模式的开启或关闭； Bit6 – 一键翻转，点击后置 1，当方向键超过一半时置 0； Bit7 – 一键平衡，先置 1，1s 后置 0，油门为 0 时才能送出平衡值。
BYTE[6]		$(\text{BYTE}[1] \wedge \text{BYTE}[2] \wedge \text{BYTE}[3] \wedge \text{BYTE}[4] \wedge \text{BYTE}[5])$ &0xff
BYTE[7]	帧尾	固定为 0x99

#### 4. 运行环境规定

##### 4.1 设备

通过 UART 命令实现数据透传。APP 发送的遥控数据，WIFI 摄像头接受后，不做任何处理，直接透过 UART 接口发送给飞控板。

## 4.2 接口

手机和 WIFI 摄像头连接通过 http 的请求连接；控制命令的发送要求有返回值，改为 UDP 发送；发送的值需要设置成 {Authcode (4Bytes)} + {Data}，需要将 string 转化成 data 发送。

## 5. 代码设计

### 5.1 主要技术

- (1) 手机 APP 和 WIFI 摄像头的连接；
- (2) 手机 APP 给 WIFI 摄像头的数据发送以及接受接受返回数据；
- (3) 通过手机给 WIFI 摄像头上传.sh 文件实现摄像头升级；
- (4) FFmpeg 解码实现实时播放；
- (5) iOS 的 delegate、notification、kvo、单例设计模式等。

### 5.2 设计难点

(1) 由于摄像头传输的数据是由 WIFI 传输的，所以在超过一定的距离以后，可能会出现画面卡死的现象，需要解决的是，在 WIFI 摄像头从新在向手机靠近时，画面会恢复，并且延迟不会超过 500ms。

(2) 将.sh 文件转成转化输入流到 WIFI 摄像头端升级摄像头。

(3) 控制命令发送的时候，WIFI 摄像头接受的命令为 data，所以需要将原来的 string 转化成 data，以十六进制的形式发送出去。

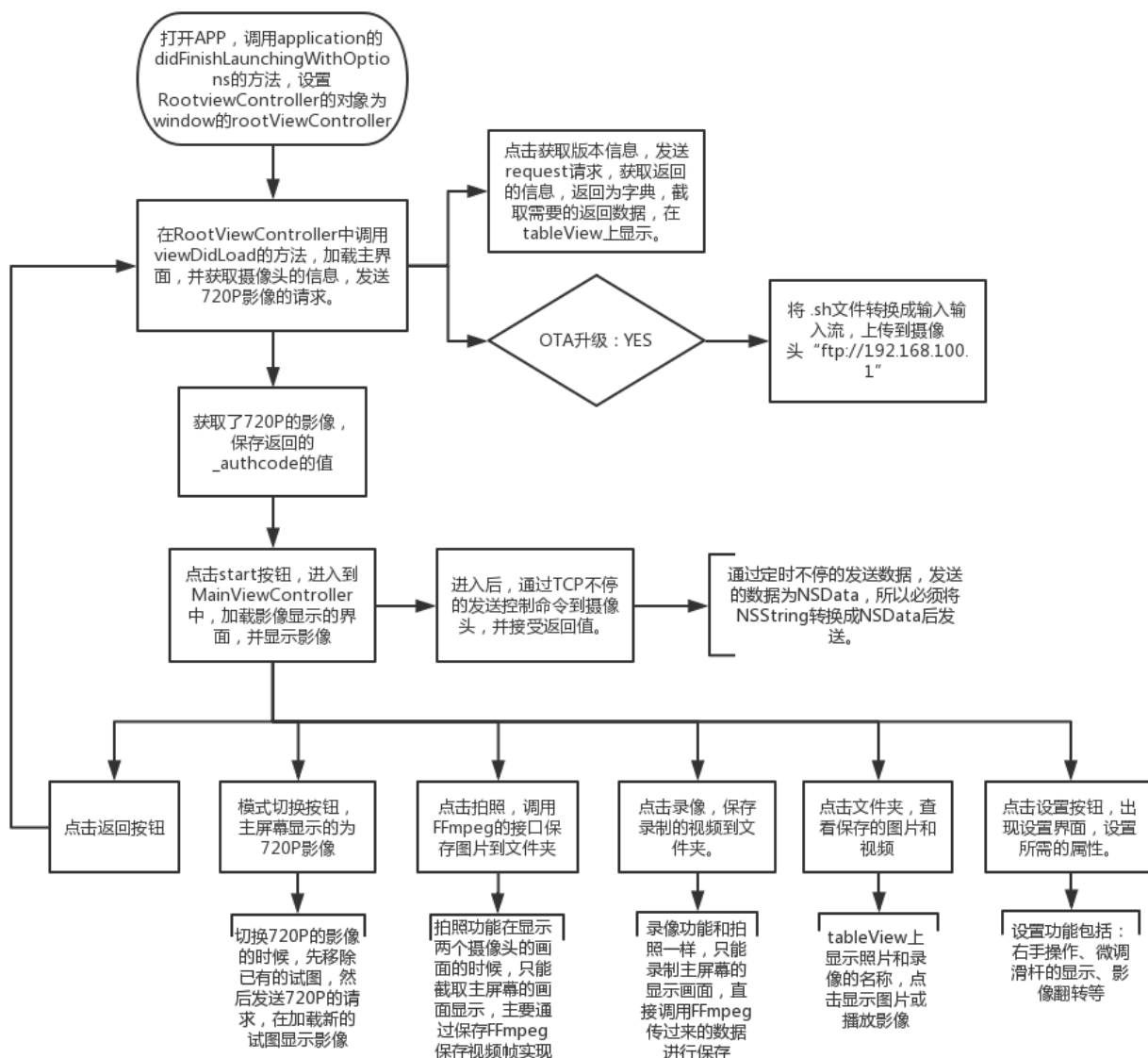
(4) 控制命令的拼接，由于 TCP 发送的控制命令是 NSData，那么我们需要将 NSString 转换成 NSData 再发送；例如：string 类型的“11223344”需要转成 data 类型的“11223344”。

(5) 在接受数据的时候可能会出现丢帧的现象，画面会出现马赛克，如何实

现播放的画面没有马赛克。

### 5.3 代码流程

该 APP 主要用在实时的显示 WIFI 摄像头拍摄的画面和控制飞机的运行状态，根据所需的功能，可以确定 APP 的功能流程，大致如下：



(1) RootViewController：设置成打开 APP 的时候直接进入到这个类中；

RootViewController 主要包括 start 界面的界面设计，获取摄像头的版本信息，

OTA 升级功能, 打开文件夹, 操作说明, 功能的设置, 获取 720P 的影像等功能。

(2) MainViewContrller: 点击 start 按钮的时候直接 present 到这个类创建的对象中; MainViewController 主要包括影像显示界面的设计, 不同模式的切换; TCP 控制命令的发送, 设置功能的实现 (包括影像翻转、右手模式等)。该类主要涉及到了界面的加载, 所以只需要在主线程中实现即可, 无须开启新的线程。

(3) KxMovieViewController: 主要用于调用 FFmpeg 解码和解码后视频的显示, 为了防止解码造成的主线成堵塞, 所以需要开启新的线程来完成解码和的工程。

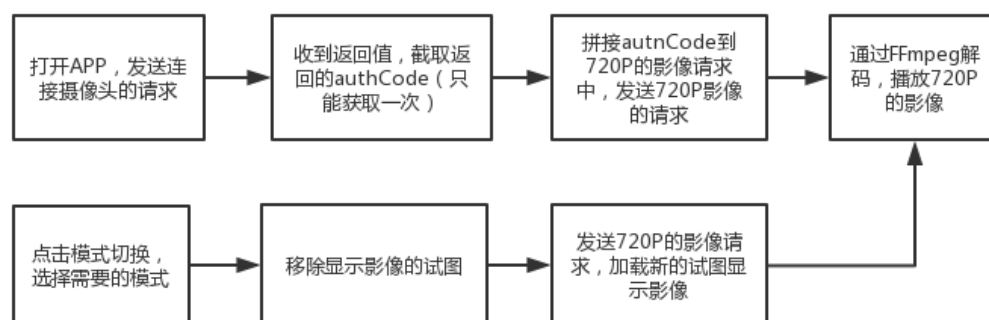
(4) FKFileViewController: 文件夹界面的设计和保存录制影像和拍摄的照片。

(5) HelpViewController: 操作说明的界面设计, 主要通过 scrollerView 来完成。

## 5.4 软件结构

飞控有关项目的 APP 大致分为两个模块, 功能模块 (及要求实现的功能, 比喻拍照、录像等功能) 和实时的播放 (FFmpeg 的解码)。

(1) 720P 画面的显示:



```
NSURLRequest * request = [NSURLRequest requestWithURL:[NSURL
URLWithString:[NSString stringWithFormat:@"http://192.168.100.1/information"]]]
```

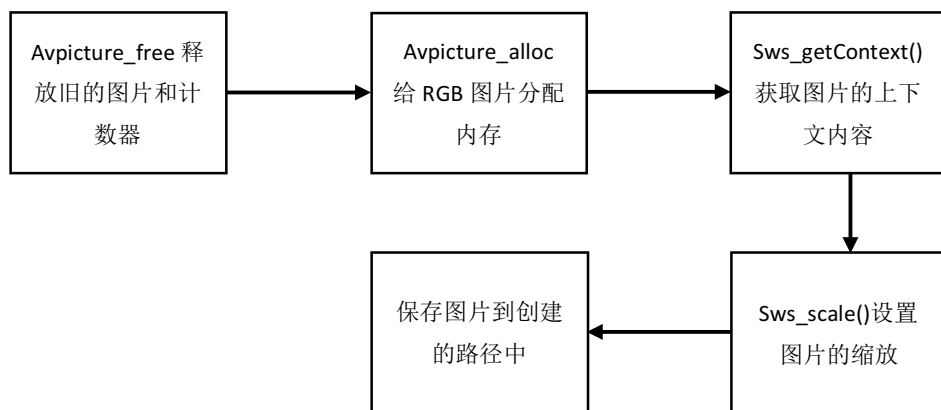
将 string 转换成 NSURLRequest，转成可以被接受的请求。

```
[NSURLConnection senfAsynchronousRequest:request queue:[NSOperationQueue
currentQueue] completionHandler:^(NSURLResponse *response, NSData *data,
NSError *connectionError){…………}];
```

调用此方法，在当前的主线程中发送异步请求，实现 720P 的画面显示。

## (2) 拍照：

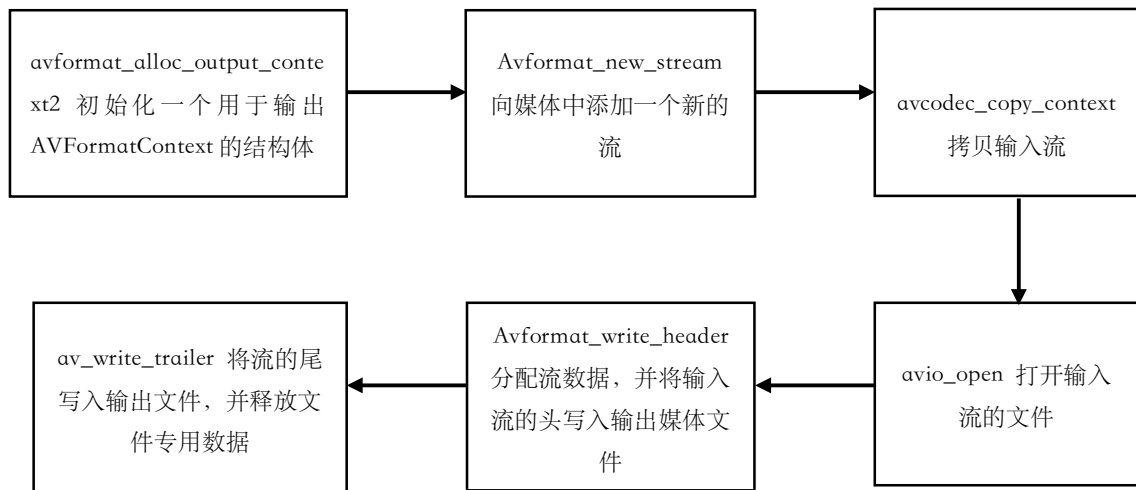
本地截图的处理在 kxmove 中作乐扩展，直接调用 FFmpeg 的接口对传来的帧数直接保存为 RGB 图片。



主要调用了 FFmpeg 中对应的函数，具体函数在函数解析下面做了解释。

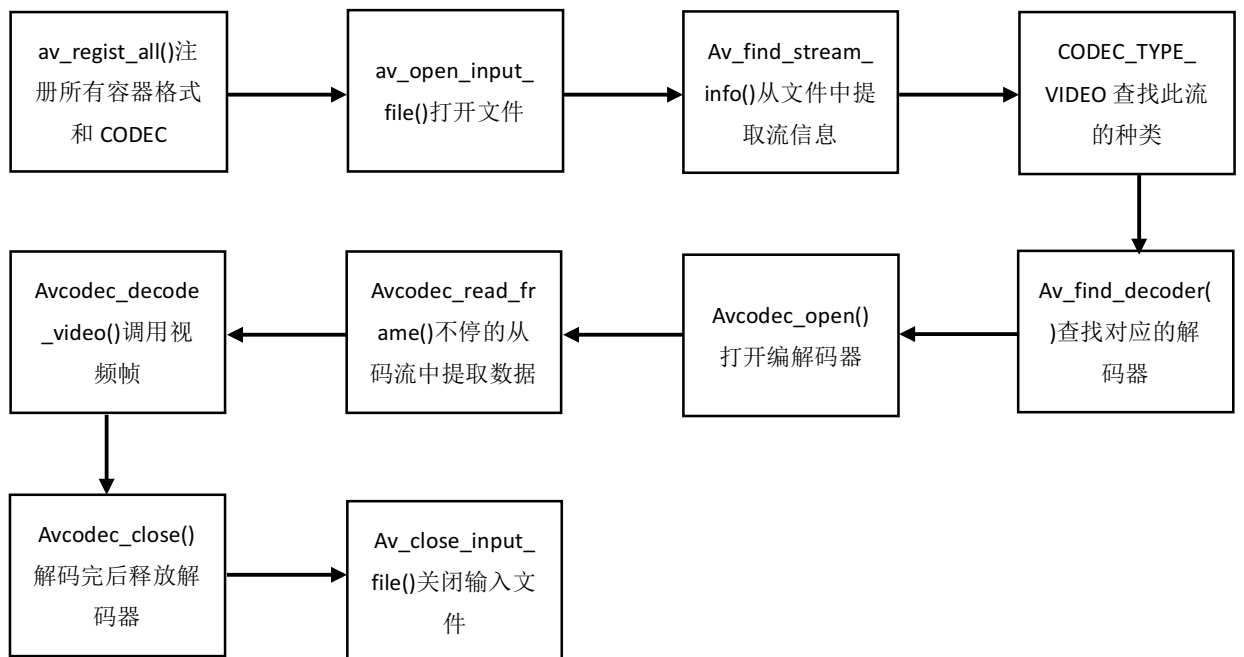
## (3) 录像

本地录制视频的处理实在 kxmove 中作了扩展，直接调用 ffmpeg 的借口对传送的数据直接保存为 mp4 的视频。



#### (4) FFmpeg 解码

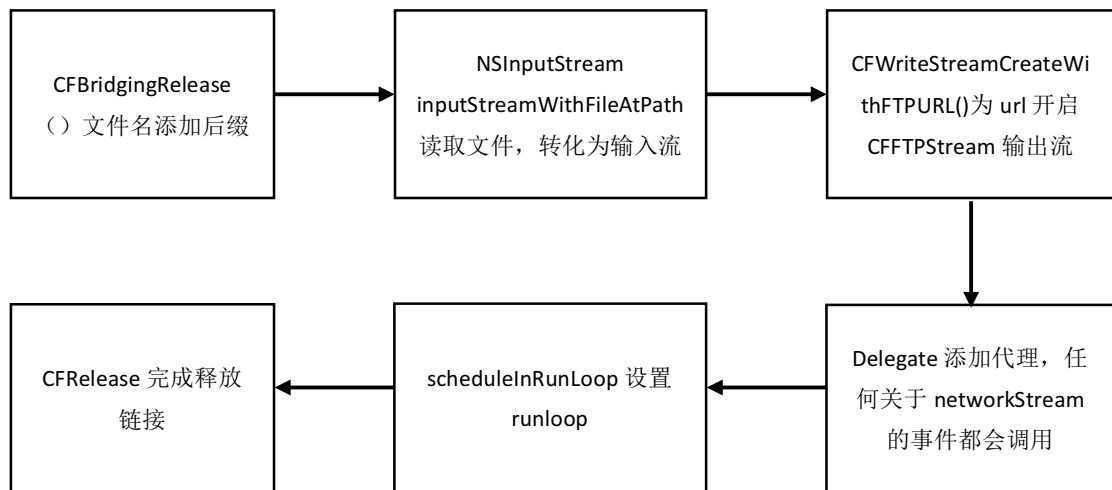
将摄像头传来的数据通过 FFmpeg 解码成实时播放的视频，为了防止线程的堵塞，需要从新的开启新线程，FFmpeg 的大致解码流程如下：



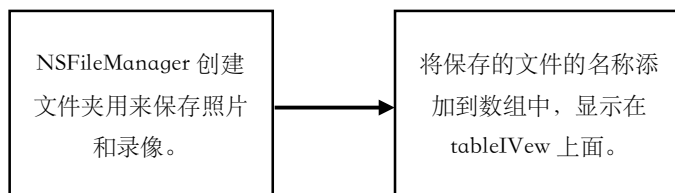
#### (5) OTA 升级

将摄像头的升级文件放在 APP 中，通过摄像头的 URL 将升级文件上传到摄像头，实现摄像头的在线升级功能。

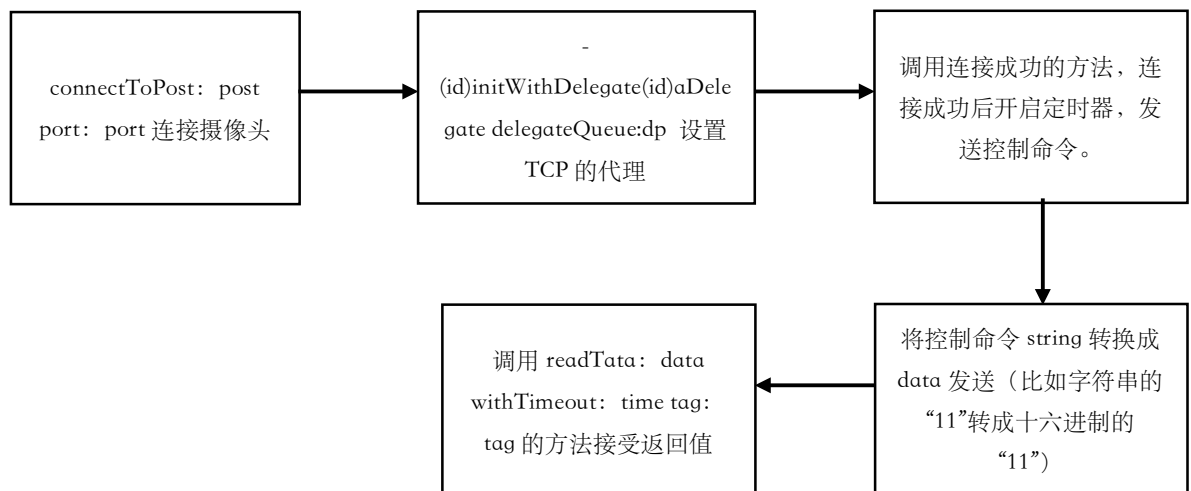




## (6) 文件保存和文件夹的创建



## (7) TCP 控制命令的发送



## 5.5 函数说明

对于不同的函数，将其分别放到了不同的模块中进行了分类说明。

### (1) FFmpeg 解码：

<1> av\_register\_all (void): 注册所有容器格式和编解码器;

```
<2> avformat_open_input(AVFormatContext **ps,  
  
                        const char *url,  
  
                        AVInputFormat *fmt,  
  
                        AVDictionary **options): 打开需要解码的文件;
```

ps: 函数调用成功之后处理过的 AVFormatContext;

url: 打开的视频流的 URL;

fmt: 强制制定 AVFormatContext 中 AVInputFormat 的类型, 这个参数一般设置为 NULL, 这样 FFmpeg 可以自动检测 AVinputFormat;

options: 附加的一些选项, 一般情况下设置为 NULL。

```
<3> av_find_stream_info(AVFormatContext *ic,  
  
                        AVDictionary **options): 从文件中提取流信息;
```

ic: 输入的 AVFormatContext;

options: 额外的选项, 一般设置为空。

<4> avcodec\_find\_decoder(enum AVCodecID id): 查找对应的解码器;

id: 解码器的 ID, 返回查找到的解码器 (没有找到就返回 NULL)。

```
<5> avcodec_open2(AVCodecContext *avctx,  
  
                  const AVCodec *codec,  
  
                  AVDictionary **options): 打开编解码器;
```

avctx: 需要初始化的 AVCodecContext;

codec: 输入的 AVCodec;

options: 额外的选项, 一般设置为 NULL; 当使用 libx264 编码的时候, “preset”,

“tune”等都可以通过该参数设置。

<6> avcodec\_alloc\_frame(void): 为解码帧分配内存;

<7>av\_read\_frame(AVFormatContext \*s, AVPacket \*pkt): 不停的从码流中提取帧数据;

s: 输入的 AVFormatContext;

pkt: 输出的 AVPacket。

h. avcodec\_decode\_video2((AVCodecContext \*avctx, AVFrame \*picture,  
int \*got\_picture\_ptr,  
const AVPacket \*avpkt): 判断帧的类型, 对视频帧

进行调用;

avctx: 编解码器的环境;

picture: 解码后视频帧被储存的格式;

got\_picture\_ptr: 检查视频帧的宽、高是否正确;

avpkt: 输出的AVPacket。

<8> avcodec\_close(AVCodecContext \*avctx): 解码完后, 释放解码器;

avctx: 解码完后需要释放的 AVCodecContext。

<9> av\_free\_packet(AVPacket \*pkt);关闭输入文件;

pkt: 需要关闭的 AVPacket 包。

<10> avformat\_network\_init(void): 如果是打开的是网络流的话, 在函数

avformat\_open\_input()前面加上此函数;

<11> avformat\_alloc\_context(void) &

avformat\_free\_context(AVFormatContext \*s): 上下文内存的分配和释放;

s: 需要释放的 AVFormatContext 对象。

```
<12>avcodec_decode_audio4(AVCodecContext *avctx, AVFrame *frame,  
  
                           int *got_frame_ptr,  
  
                           const AVPacket *avpkt): 解码音频流, 返回值为解  
码数量;
```

avctx: 需要解码的 AVCodecContext;

frame: 传入需要解码的 frame;

got\_frame\_ptr: 检查音频帧的宽、高是否正确;

avpkt: 储存编解码数据相关的结构体。

```
<13> avcodec_decode_subtitle2(AVCodecContext *avctx,  
  
                              AVSubtitle *sub,  
  
                              int *got_sub_ptr,  
  
                              AVPacket *avpkt): 解码视频的字幕留言;
```

avctx: 需要解码的 AVCodecContext;

frame: 传入需要解码的 frame;

got\_frame\_ptr: 检查字幕的宽、高是否正确;

avpkt: 储存编解码数据相关的结构体。

(2) 拍照功能:

```
<1>avpicture_free (AVPicture * picture): 释放存储图片信息的内存;
```

picture: 需要释放的 AVPicture 对象。

```
<2> avpicture_alloc(AVPicture *picture,
```

```
enum AVPixelFormat pix_fmt,

int width,

int height): 初始化 RGB 图片并分配内存;
```

picture: 需要初始化的 AVPicture 对象;

pix\_fmt: 初始化 AVPixelFormat 对象, 该参数设置图片帧的格式;

width: 设置图片帧的宽度;

height: 设置图片帧的长度。

```
<3> sws_getContext(int srcW, int srcH, enum AVPixelFormat srcFormat,

int dstW, int dstH, enum AVPixelFormat dstFormat,

int flags, SwsFilter *srcFilter,

SwsFilter *dstFilter, const double *param): 调用缩放器, 设置
```

图片的缩放;

srcW: 源图像的宽;

srcH: 源图像的高;

srcFormat: 源图像的像素格式;

dstW: 目标图像的宽;

dstH: 目标图像的高;

dstFormat: 目标图像的像素格式;

flags: 设置图像拉伸使用的算法;

成功执行的话返回生成的 SwsContext, 否则返回为 NULL。

```
<4> -(nullable NSArray<NSString*> *)contentsOfDirectoryAtPath:
```

```
(NSString *)path
```

`error:(NSError **)error NS_AVAILABLE(10_5, 2_0)`: 将路径的内容保存到沙盒;

`path`: 文件的保存路径。

`<5>-(NSString *)stringByAppendingPathComponent:(NSString *)str`: 设置保存文件的名称;

`<6>UIImageWriteToSavedPhotosAlbum(UIImage *image, __nullable id completionTarget, __nullable SEL completionSelector, void * __nullable contextInfo)`: 将照片添加到已保存的照片相册。

`image`: 要写入相册的图片;

`completionTarget`: 可选项, 保存完成后, 回调方法所在的对象;

`completionSelector`: 要调用的 `completionTarget` 的回调方法。

`contextInfo`: 可选参数, 保存了一个指向 `context` 数据的指针, 他将传递给回调方法。

### (3) 录像功能:

`<1> avformat_alloc_output_context2(AVFormatContext **ctx, AVOutputFormat *oformat, const char *format_name, const char *filename)`: 初始化一个用于输出的 `AVFormatContext` 结构体的对象;

ctx: 函数调用成功之后创建的 AVFormatContext;

oformat: 指定 AVFormatContext 中的 AVOutputFormat, 用于确定输出格式, 如果指定为 NULL, 可以设定后两个参数(format\_name 或则 filename)由 FFmpeg 猜测输出格式;

PS: 使用该参数需要自己手动获取 AVOutputFormat, 相对于使用后两个参数来说要麻烦一些;

Format\_name: 指定输出格式名称。根据格式名称, FFmpeg 会推测出格式, 输出格式可以是“flv”、“mkv”等等;

Filename: 指定输出文件的名称。根据文件名称, FFmpeg 会推测输出格式。文件名称可以是“xx.flv”、“yy.mkv”等等;

函数执行成功的话, 其返回值大于等于零。

<2>avformat\_new\_stream(AVFormatContext \*s, const AVCodec \*c): 创建一个新的流通道;

s: 新的流通道中的 AVFormatContext;

c: 创建新的流通道中的编解码器。

<3>avio\_open(AVIOContext \*\*s, const char \*url, int flags): 创建并初始化一个 AVIOContext 以 url 指向的资源;

s: 函数调用成功后创建的 AVIOContext 结构体;

url: 输入输出协议的地址 (文件也是一种“广义”的协议, 对于文件来说就是文件的路径);

flags: 打开地址的方式。可以选择只读, 只写, 或则读写。

<4>avformat\_write\_header(AVFormatContext \*s, AVDictionary \*\*options): 分配

流数据，并将流的头写入输出媒体文件；

s: 用于输出的 AVFormatContext;

options: 额外的选项，一般情况下设置为 NULL。

<5>av\_interleaved\_write\_frame(AVFormatContext \*s, AVPacket \*pkt): 将数据包写入输出媒体文件，确保写入正确；

s: 用于输入的 AVFormatContext;

pkt: 用于输出的 AVPacket。

<6>av\_write\_trailer(AVFormatContext \*s): 将流的尾写入输出媒体文件并释放文件专用数据；

s: 关闭用于输入的 AVFormatContext。

#### (4) TCP 发送控制指令:

<1>-(BOOL)connectToHost:(NSString\*)host onPort:(uint16\_t)port error:(NSError \*)errPtr: 连接到对应的端口，返回值为 BOOL 值，YES 表示成功；

<2>-(void)socket:(GCDAsyncSocket \*)sock didConnectToHost:(NSString \*)host port:(uint16\_t)port: 连接成功后调用这个代理方法；

<3>-(void)socketDidDisconnect:(GCDAsyncSocket \*)sock withError:(NSError \*)err: 连接中断时调用此方法；

<4>-(void)socket:(GCDAsyncSocket \*)sock didReadData:(NSData \*)data withTag:(long)tag: 此方法用于接受服务器的返回值；

<5>-(void)writeData:(NSData \*)data withTimeout:(NSTimeInterval)timeout

tag(long)tag: 发送数据给服务器，timeout 为超时设置，-1 表示无时间设置。



<6>-(void)readDataWithTimeout(NSTimeInterval)timeout tag(long)tag: 调用接受返回值的代理方法时，在连接成功的方法中调用此方法；timeout 设置为 -1 即为不设置超时，一直接受返回值。

## 6. 实施计划

### 6.1 限制

该 APP 是基于单摄像头上做的扩展开发，很多功能和条件上的限制和单摄像头的相同。

(1) 由于双摄像头模式下需要同时显示两个摄像头的画面，因此需要开辟两个流来进行编解码；在摄像头的距离距离 APP 越远时，由于传输的影像数据比单摄像头的大，所以传输的距离不会有单摄像头远。如果单摄像头在 50m 左右才会出现卡顿的情况，那么双摄像头在 25m 左右就可能出现卡顿的情况。

(2) 由于没有飞机或者小车来实际的测试 APP 的控制命令，所有只能由串口的打印数据来判断控制命令的发送是否和说明书一致。

(3) APP 摇杆发送数据，由于手机屏幕的限制，摇杆的灵敏度过高，不能很好的控制飞机或者玩具车运动。

(4) 在切换摄像头模式的时候，由于图像显示的是 720P 的影像，APP 发送 720P 影像请求的时候会出现三秒的延迟，所以在切换的时候，屏幕会白屏三秒才会显示影像。同样的，在进入 APP 后，需等待三秒后才能点击开始按钮进入影像显示的界面。

(5) 由于 WIFI 信号会受到外部环境的影响，所以测试的时候尽量选择晴天，并且到操场等开阔的地方测试。

### 6.2 实施内容和进度安排

由于该 APP 是基于单摄像头的代码上做的开发，所以很多的功能可以借鉴单摄像头的模版，只需要在单摄像头的代码上加上所需的新的功能即可。大致可以分为以下几个模块：

(1) 不同模式下镜头的切换；主要分为四种模式：a. 广角镜头；b. 非广角镜头；c. 广角镜头为主屏幕，非广角给副屏幕；d. 非广角镜头为主屏幕，广角镜头为副屏幕。

(2) 720P 影像的显示；在切换到任何模式下，主屏幕显示的图像都是 720P 的影像。

(3) OTA 升级，将.sh 文件上传到摄像头上，实现通过 APP 升级摄像头。

(4) 控制命令的发送；通过 TCP 给摄像头发送控制命令，同时能够接受到摄像头的返回值。

(5) 原有功能的修改和优化：拍照和录像根据选择模式的不同，对不同的摄像头进行拍照和录像；摇杆在触摸屏幕的时候显示，手指离开的时候隐藏等；根据需求的说明，对现有的功能进行修改。

(6) 功能测试：按照测试要求对完成的 APP 进行测试并做好记录。

(7) 修改 BUG 和维护优化；对应测试的结果，对没达到要求的的功能做进一步的修改和优化。

### **6.3 实施条件和措施**

开发中需要其他开发人员配合测试。

## **7. 测试计划**

### **7.1 测试方案**

该 APP 主要是用手机实现对玩具无人机的操控和实时的获取 WIFI 摄像头上

的影像；要确保手机能够操控飞机的正常飞行，手机显示的影像无马赛克、破图、卡顿等现象。所以需要测试的项目有：

- (1) 手机控制玩具无人机飞行的最远距离；
- (2) 手机显示画面的延迟大小；
- (3) 手机近距离显示画面是否出现马赛克、破图、卡顿等现象；
- (4) 手机远距离显示画面是否出现马赛克、破图、卡顿等现象，最远可到多少米；
- (5) 拍照和录像功能是否能够正常的使用。

针对上面需要测试内容，制定的测试方案：

- (1) 延迟方面：将摄像头对准秒表，手机图像显示出秒表的时间，用另外的手机进行拍照，将秒表和显示秒表的画面都拍摄到，对应拍摄照片上的时间，用手机图像上秒表的时间减去实际秒表上的时间，即可得到画面的延迟，反复的多次测试，求的平均值，即为画面的延迟。
- (2) 图像显示方面：分别在 0m、5m、10m、15m……分别进行测试，在不同的距离下看图像是否出现破图、马赛克、卡顿、延时等现象；同时在不同的距离下，手机是否能够控制飞机的正常起飞和停止。

## 7.2 测试结果

以表格的形式纪录测试的结果，主要测试的有：影像延时、影像卡顿、影像马赛克、影像流畅性、摄像头拍照、摄像头录像、查看手机文件、播放录像、删除播放录像、手机影像翻转、侧滚摇杆的控制、俯仰摇杆的控制、油门摇杆的控制、偏航摇杆的控制、侧滚微调、俯仰微调、偏航微调、定高模式、左右手切换。看这些功能是否能够正常的操作。

对影像方面要在不同距离、不同手机的情况下分别做好测试，并对比：

日期	测试距离	手机型号	测试结果

大致如上图所示，制作出测试记录表格，以便测试时记录测试结果。

## 8. 验收标准

针对双摄像头的实际运行环境，验收标准如下：

（1）用 TCP 给双摄像头发送数据和接收摄像头的返回数据，用串口调试的时候，确定发送的数据和接收的数据一样。

（2）确保 APP 能够完美的实现需求书的所有功能，比如：拍照、录像、右手模式等。

（3）APP 实时的显示 WIFI 摄像头的影像，要求影像不出现马赛克、破图、画面不流畅等现象，画面的延迟不得超过 200ms。

（4）APP 长时间接受静止的画面时，不会出现画面卡死的现象。