

****SHANGHAI JIAO TONG UNIVERSITY

**无线通讯及VR软件说明书**

|  |  |
| --- | --- |
| **项目名称：** | 类人机器人遥操作控制 |
|  |  |
| **学生姓名：** | 程 洁 |
|  |  |
| **指导教师：** | 贡 亮 |
| **教学班级：** | 机电4班 |
| **学 院：** | 机械与动力工程学院 |
| **项目时间：** | 2021年 4月 10日 至 2021年 8月5 日 |

目 录

[1 软件安装说明 1](#_Toc80799056)

[1.1 PC VR 头戴式设备软件设置 1](#_Toc80799057)

[1.2 浏览器选择 2](#_Toc80799058)

[2 软件部分源代码及说明 3](#_Toc80799059)

[2.1 数据传输 3](#_Toc80799060)

[2.2 视频流传输 5](#_Toc80799061)

# 软件安装说明

## PC VR 头戴式设备软件设置

由于现有设备为TCL品牌的头戴式VR设备，考虑其适配性，下载安装其支持软件HTCVive。链接如下：

<https://www.vive.com/cn/setup/>

头戴式设备的版本为VIVE Cosmos，即在该网页下载对应版本,显示界面如下：





注意：该软件对硬件设备有一定要求，具体要求如下：



## 浏览器选择

为实现浏览器接收视频直接投屏的功能，在查阅资料后选择了Firefox Reality。该软件可在上述HTC Viveport的软件库中直接下载安装。

VR设备界面如下：



# 软件部分源代码及说明

由于代码环节较多，过于复杂，在此列出及说明部分代码，完整源码请见附件。

## 数据传输

本文只说明控制器端，即Ubuntu端数据接收处理部分的代码。基于已有接口程序，可直接将WiFi传输程序写入该接口程序，代码如下：

#define SOCKET

void SubWindow::on\_startSockBt\_clicked() {

if (!isSync) {

onAddInfo("[ERROE] Not sync yet");

return;

}

#if 1

onAddInfo("start sock...");

RobotState s;

rm->get\_robot\_state(&s);

if (!s.servoEnabled) {

onAddInfo("start sock failed: robot is not enable");

return;

}

m\_isWillSidecarThreadTerminate.exchange(false);

m\_sidecarThread = std::thread([&](){

/\* TODO: open socket \*/

#ifdef SOCKET

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Socket Initialization \*\*\*\*\*\*\*\*\*\*\*\*\*/

int i, len, rc, on = 1;

int listen\_sd, max\_sd, new\_sd;

int desc\_ready, end\_server = FALSE;

int close\_conn;

// char buffer[80];

double bufferSock[7];

struct timeval timeout;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Create an AF\_INET6 stream socket to receive incoming \*/

/\* connections on \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

listen\_sd = socket(AF\_INET6, SOCK\_STREAM, 0);

if (listen\_sd < 0)

{

perror("socket() failed");

exit(-1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Allow socket descriptor to be reuseable \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

rc = setsockopt(listen\_sd, SOL\_SOCKET, SO\_REUSEADDR,

(char \*)&on, sizeof(on));

if (rc < 0)

{

perror("setsockopt() failed");

::close(listen\_sd);

exit(-1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Set socket to be nonblocking. All of the sockets for \*/

/\* the incoming connections will also be nonblocking since \*/

/\* they will inherit that state from the listening socket. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

rc = ioctl(listen\_sd, FIONBIO, (char \*)&on);

if (rc < 0)

{

perror("ioctl() failed");

::close(listen\_sd);

exit(-1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Bind the socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

memset(&addr, 0, sizeof(addr));

addr.sin6\_family = AF\_INET6;

memcpy(&addr.sin6\_addr, &in6addr\_any, sizeof(in6addr\_any));

addr.sin6\_port = htons(SERV\_PORT);

rc = bind(listen\_sd,

(struct sockaddr \*)&addr, sizeof(addr));

if (rc < 0)

{

perror("bind() failed");

::close(listen\_sd);

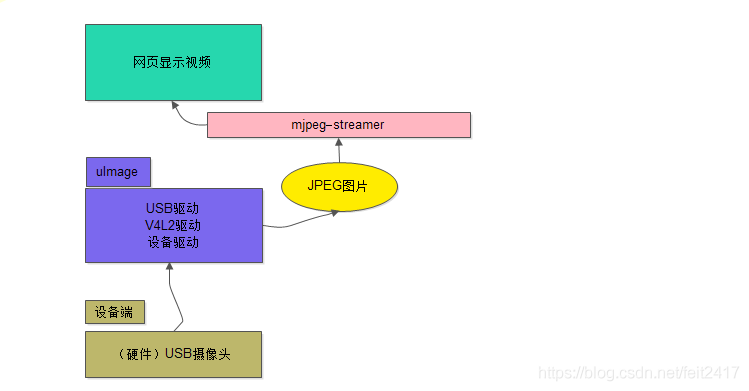
std::cout << "listen\_sd: " << listen\_sd << std::endl;

exit(-1);

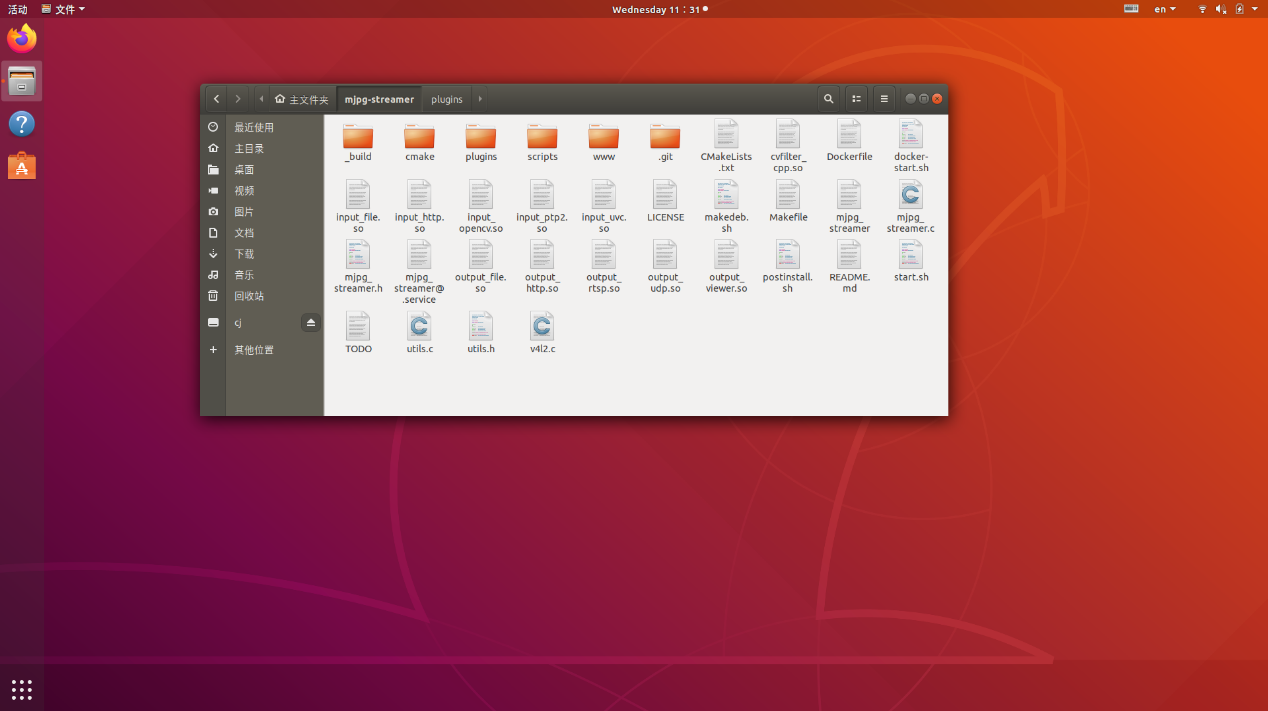
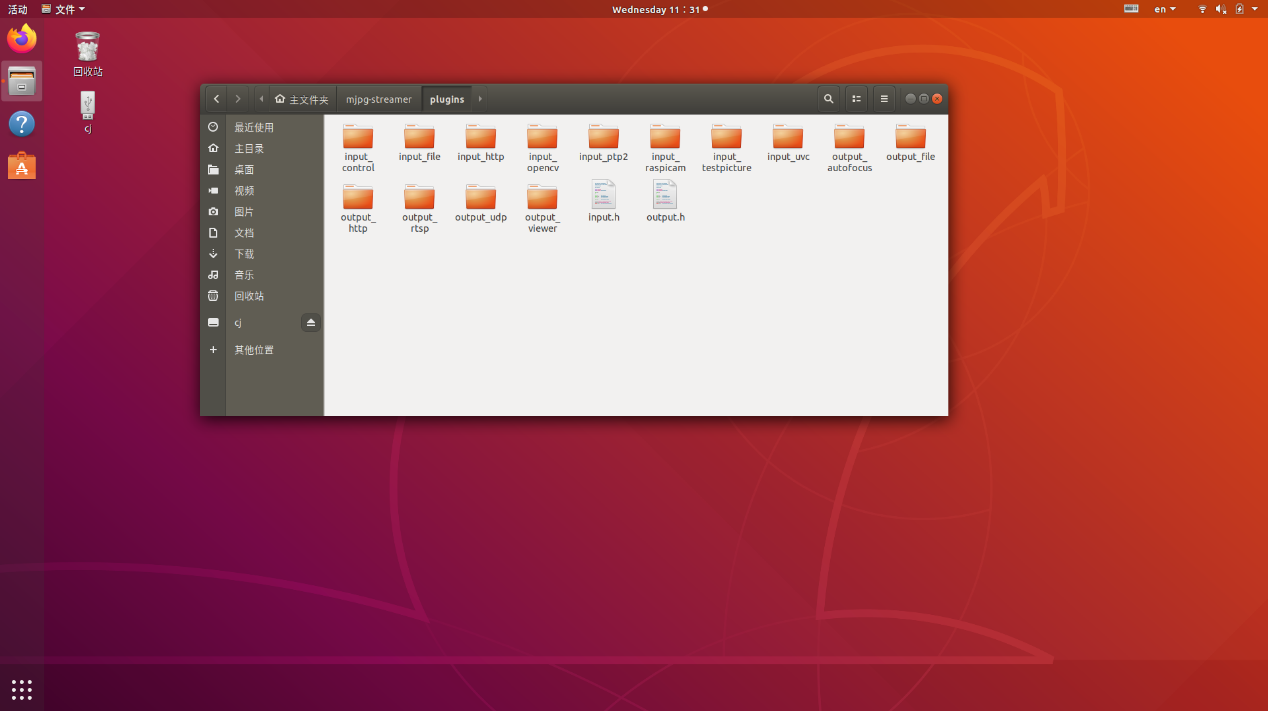
}

## 视频流传输

在本项目中，视频传输是基于mjpeg-streamer平台，并在原有代码中进行了修改，使其可在传输视频流的同时，以图片格式保存视频流至本地，传输逻辑如下：



摄像头+驱动+JPEG图库+mjpeg-streamer+HTML 完成了对用户远程提供视屏展示的服务。首先底层通过摄像头采集图片，通过驱动将数据流存储到JPEG图库，最终通过html交给用户的浏览器来解析，而mjpeg-streamer用来实现对这一流程的的控制。构成mjpeg-streamer视频流服务器。



输出改进部分代码如下：（完整源码见附件）

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/ioctl.h>

#include <errno.h>

#include <signal.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <getopt.h>

#include <pthread.h>

#include <syslog.h>

#include <linux/types.h> /\* for videodev2.h \*/

#include <linux/videodev2.h>

#include "../../mjpg\_streamer.h"

#include "../../utils.h"

#include "httpd.h"

#include <fcntl.h>

#include <time.h>

#include <syslog.h>

#include <dirent.h>

#include "output\_file.h"

static pthread\_t worker;

static globals \*pglobal;

static int fd, delay=50, ringbuffer\_size = -1, ringbuffer\_exceed = 0, max\_frame\_size;

static char \*folder = "/home/cj/picture";

static unsigned char \*frame = NULL;

static char \*command = NULL;

static int input\_number = 0;

static char \*mjpgFileName = NULL;

static char \*linkFileName = NULL;

#define OUTPUT\_PLUGIN\_NAME "HTTP output plugin"

#define OUTPUT\_PLUGIN2\_NAME "FILE output plugin"

void worker\_cleanup(void \*arg)

{

static unsigned char first\_run = 1;

if (mjpgFileName != NULL) {

close(fd);

}

if(!first\_run) {

DBG("already cleaned up resources\n");

return;

}

first\_run = 0;

OPRINT("cleaning up resources allocated by worker thread\n");

if(frame != NULL) {

free(frame);

}

close(fd);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Description.: compares a directory entry with a pattern

Input Value.: directory entry

Return Value: 0 if string do not match, 1 if they match

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int check\_for\_filename(const struct dirent \*entry)

{

int rc;

int year, month, day, hour, minute, second;

unsigned long long number;

/\*

\* try to scan the string using scanf

\* I would like to use a define for this format string later...

\*/

rc = sscanf(entry->d\_name, "%d%d%d\_%d%d%d\_%09llu.jpg", &year, \

&month, \

&day, \

&hour, \

&minute, \

&second, \

&number);

DBG("%s, rc is %d (%d, %d, %d, %d, %d, %d, %llu)\n", entry->d\_name, \

rc, \

year, \

month, \

day, \

hour, \

minute, \

second, \

number);

/\* if scanf could find all values, it matches our filenames \*/

if(rc != 7) return 0;

return 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Description.: delete oldest files, just keep "size" most recent files

This funtion MAY delete the wrong files if the time is not valid

Input Value.: how many files to keep

Return Value: -

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void maintain\_ringbuffer(int size)

{

struct dirent \*\*namelist;

int n, i;

char buffer[1<<16];

/\* do nothing if ringbuffer is not set or wrong value is set \*/

if(size < 0) return;

/\* get a sorted list of directory items \*/

n = scandir(folder, &namelist, check\_for\_filename, alphasort);

if(n < 0) {

perror("scandir");

return;

}

DBG("found %d directory entries\n", n);

/\* delete the first (thus oldest) number of files \*/

for(i = 0; i < (n - size); i++) {

/\* put together the folder name and the directory item \*/

snprintf(buffer, sizeof(buffer), "%s/%s", folder, namelist[i]->d\_name);

DBG("delete: %s\n", buffer);

/\* mark item for deletion \*/

if(unlink(buffer) == -1) {

perror("could not delete file");

}

/\* free allocated memory for name \*/

free(namelist[i]);

}

/\* keep the rest, but we still have to free every result \*/

for(i = MAX(n - size, 0); i < n; i++) {

DBG("keep: %s\n", namelist[i]->d\_name);

free(namelist[i]);

}

/\* free last just allocated resources \*/

free(namelist);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Description.: this is the main worker thread

it loops forever, grabs a fresh frame and stores it to file

Input Value.:

Return Value:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void \*worker\_thread(void \*arg)

{

int ok = 1, frame\_size = 0, rc = 0;

char buffer1[1024] = {0}, buffer2[1024] = {0};

unsigned long long counter = 0;

time\_t t;

struct tm \*now;

unsigned char \*tmp\_framebuffer = NULL;

/\* set cleanup handler to cleanup allocated resources \*/

pthread\_cleanup\_push(worker\_cleanup, NULL);

while(ok >= 0 && !pglobal->stop) {

DBG("waiting for fresh frame\n");

pthread\_mutex\_lock(&pglobal->in[input\_number].db);

pthread\_cond\_wait(&pglobal->in[input\_number].db\_update, &pglobal->in[input\_number].db);

/\* read buffer \*/

frame\_size = pglobal->in[input\_number].size;

/\* check if buffer for frame is large enough, increase it if necessary \*/

if(frame\_size > max\_frame\_size) {

DBG("increasing buffer size to %d\n", frame\_size);

max\_frame\_size = frame\_size + (1 << 16);

if((tmp\_framebuffer = realloc(frame, max\_frame\_size)) == NULL) {

pthread\_mutex\_unlock(&pglobal->in[input\_number].db);

LOG("not enough memory\n");

return NULL;

}

frame = tmp\_framebuffer;

}

/\* copy frame to our local buffer now \*/

memcpy(frame, pglobal->in[input\_number].buf, frame\_size);

/\* allow others to access the global buffer again \*/

pthread\_mutex\_unlock(&pglobal->in[input\_number].db);

if (mjpgFileName == NULL) { // single files with ringbuffer mode

/\* prepare filename \*/

memset(buffer1, 0, sizeof(buffer1));

memset(buffer2, 0, sizeof(buffer2));

/\* get current time \*/

t = time(NULL);

now = localtime(&t);

if(now == NULL) {

perror("localtime");

return NULL;

}

/\* prepare string, add time and date values \*/

if(strftime(buffer1, sizeof(buffer1), "%%s/%Y%m%d\_%H%M%S\_%%09llu.jpg", now) == 0) {

OPRINT("strftime returned 0\n");

free(frame); frame = NULL;

return NULL;

}

/\* finish filename by adding the foldername and a counter value \*/

snprintf(buffer2, sizeof(buffer2), buffer1, folder, counter);

counter++;

DBG("writing file: %s\n", buffer2);

/\* open file for write \*/

if((fd = open(buffer2, O\_CREAT | O\_RDWR | O\_TRUNC, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH)) < 0) {

OPRINT("could not open the file %s\n", buffer2);

return NULL;

}

/\* save picture to file \*/

if(write(fd, frame, frame\_size) < 0) {

OPRINT("could not write to file %s\n", buffer2);

perror("write()");

close(fd);

return NULL;

}

close(fd);

/\* link the picture as fixed name file \*/

if (linkFileName) {

snprintf(buffer1, sizeof(buffer1), "%s/%s", folder, linkFileName);

unlink(buffer1);

(void) link(buffer2, buffer1);

}

/\* call the command if user specified one, pass current filename as argument \*/

if(command != NULL) {

memset(buffer1, 0, sizeof(buffer1));

/\* buffer2 still contains the filename, pass it to the command as parameter \*/

snprintf(buffer1, sizeof(buffer1), "%s \"%s\"", command, buffer2);

DBG("calling command %s", buffer1);

/\* in addition provide the filename as environment variable \*/

if((rc = setenv("MJPG\_FILE", buffer2, 1)) != 0) {

LOG("setenv failed (return value %d)\n", rc);

}

/\* execute the command now \*/

if((rc = system(buffer1)) != 0) {

LOG("command failed (return value %d)\n", rc);

}

}

/\*

\* maintain ringbuffer

\* do not maintain ringbuffer for each picture, this saves resources since

\* each run of the maintainance function involves sorting/malloc/free operations

\*/

if(ringbuffer\_exceed <= 0) {

/\* keep ringbuffer excactly at specified siOUTPUT\_PLUGIN\_NAMEze \*/

maintain\_ringbuffer(ringbuffer\_size);

} else if(counter == 1 || counter % (ringbuffer\_exceed + 1) == 0) {

DBG("counter: %llu, will clean-up now\n", counter);

maintain\_ringbuffer(ringbuffer\_size);

}

} else { // recording to MJPG file

/\* save picture to file \*/

if(write(fd, frame, frame\_size) < 0) {

OPRINT("could not write to file %s\n", buffer2);

perror("write()");

close(fd);

return NULL;

}

}

/\* if specified, wait now \*/

if(delay > 0) {

usleep(1000 \* delay);

}

}

/\* cleanup now \*/

pthread\_cleanup\_pop(1);

return NULL;

}

//----------------------------------------------------------------------

/\*

\* keep context for each server

\*/

context servers[MAX\_OUTPUT\_PLUGINS];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Description.: print help for this plugin to stdout

Input Value.: -

Return Value: -

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void help(void)

{

fprintf(stderr, " ---------------------------------------------------------------\n" \

" Help for output plugin..: "OUTPUT\_PLUGIN\_NAME"\n" \

" Help for output plugin..: "OUTPUT\_PLUGIN2\_NAME"\n" \

" ---------------------------------------------------------------\n" \

" The following parameters can be passed to this plugin:\n\n" \

" [-w | --www ]...........: folder that contains webpages in \n" \

" flat hierarchy (no subfolders)\n" \

" [-p | --port ]..........: TCP port for this HTTP server\n" \

" [-l ] --listen ]........: Listen on Hostname / IP\n" \

" [-c | --credentials ]...: ask for \"username:password\" on connect\n" \

" [-n | --nocommands ]....: disable execution of commands\n" \

" [-f | --folder ]........: folder to save pictures\n" \

" [-m | --mjpeg ].........: save the frames to an mjpg file \n" \

" [-l | --link ]..........: link the last picture in ringbuffer as this fixed named file\n" \

" [-d | --delay ].........: delay after saving pictures in ms\n" \

" [-i | --input ].........: read frames from the specified input plugin\n" \

" The following arguments are takes effect only if the current mode is not MJPG\n" \

" [-s | --size ]..........: size of ring buffer (max number of pictures to hold)\n" \

" [-e | --exceed ]........: allow ringbuffer to exceed limit by this amount\n" \

" [-c | --command ].......: execute command after saving picture\n"\

" ---------------------------------------------------------------\n");

}

/\*\*\* plugin interface functions \*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Description.: Initialize this plugin.

parse configuration parameters,

store the parsed values in global variables

Input Value.: All parameters to work with.

Among many other variables the "param->id" is quite important -

it is used to distinguish between several server instances

Return Value: 0 if everything is OK, other values signal an error

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int output\_init(output\_parameter \*param, int id)

{

int i;

int port;

char \*credentials, \*www\_folder, \*hostname = NULL;

char nocommands;

DBG("output #%02d\n", param->id);

port = htons(8080);

credentials = NULL;

www\_folder = NULL;

nocommands = 0;

delay = 1000;

pglobal = param->global;

pglobal->out[id].name = malloc((1+strlen(OUTPUT\_PLUGIN2\_NAME))\*sizeof(char));

sprintf(pglobal->out[id].name, "%s", OUTPUT\_PLUGIN2\_NAME);

DBG("OUT plugin %d name: %s\n", id, pglobal->out[id].name);

param->argv[0] = OUTPUT\_PLUGIN2\_NAME;

/\* show all parameters for DBG purposes \*/

for(i = 0; i < param->argc; i++) {

DBG("argv[%d]=%s\n", i, param->argv[i]);

}

reset\_getopt();

while(1) {

int option\_index = 0, c = 0;

static struct option long\_options[] = {

{"h", no\_argument, 0, 0

},

{"help", no\_argument, 0, 0},

{"p", required\_argument, 0, 0},

{"port", required\_argument, 0, 0},

{"l", required\_argument , 0, 0},

{"listen", required\_argument, 0, 0},

{"c", required\_argument, 0, 0},

{"credentials", required\_argument, 0, 0},

{"w", required\_argument, 0, 0},

{"www", required\_argument, 0, 0},

{"n", no\_argument, 0, 0},

{"nocommands", no\_argument, 0, 0},

{"f", required\_argument, 0, 0},

{"folder", required\_argument, 0, 0},

{"d", required\_argument, 0, 0},

// {"delay", required\_argument, 0, 0},

{"s", required\_argument, 0, 0},

{"size", required\_argument, 0, 0},

{"e", required\_argument, 0, 0},

{"exceed", required\_argument, 0, 0},

{"i", required\_argument, 0, 0},

{"input", required\_argument, 0, 0},

{"m", required\_argument, 0, 0},

{"mjpeg", required\_argument, 0, 0},

//{"l", required\_argument, 0, 0},

{"link", required\_argument, 0, 0},

// {"c", required\_argument, 0, 0},

{"command", required\_argument, 0, 0},

{0, 0, 0, 0}

};

c = getopt\_long\_only(param->argc, param->argv, "", long\_options, &option\_index);

/\* no more options to parse \*/

if(c == -1) break;

/\* unrecognized option \*/

if(c == '?') {

help();

return 1;

}

switch(option\_index) {

/\* h, help \*/

case 0:

case 1:

DBG("case 0,1\n");

help();

return 1;

break;

/\* p, port \*/

case 2:

case 3:

DBG("case 2,3\n");

port = htons(atoi(optarg));

DBG("case 2,3\n");

folder = malloc(strlen(optarg) + 1);

strcpy(folder, optarg);

if(folder[strlen(folder)-1] == '/')

folder[strlen(folder)-1] = '\0';

break;

/\* Interface name \*/

case 4:

case 5:

DBG("case 4,5\n");

hostname = strdup(optarg);

//delay = atoi(optarg);

break;

/\* c, credentials \*/

case 6:

case 7:

DBG("case 6,7\n");

credentials = strdup(optarg);

ringbuffer\_size = atoi(optarg);

break;

/\* w, www \*/

case 8:

case 9:

DBG("case 8,9\n");

www\_folder = malloc(strlen(optarg) + 2);

strcpy(www\_folder, optarg);

if(optarg[strlen(optarg)-1] != '/')

strcat(www\_folder, "/");

ringbuffer\_exceed = atoi(optarg);

break;

/\* n, nocommands \*/

case 10:

case 11:

DBG("case 10,11\n");

DBG("case 12,13\n");

input\_number = atoi(optarg);

nocommands = 1;

break;

/\* m mjpeg \*/

case 12:

case 13:

DBG("case 12,13\n");

mjpgFileName = strdup(optarg);

break;

/\* l link \*/

case 14:

case 15:

DBG("case 14,15\n");

linkFileName = strdup(optarg);

break;

/\* c command \*/

case 16:

case 17:

DBG("case 16,17\n");

command = strdup(optarg);

break;

}

}

servers[param->id].id = param->id;

servers[param->id].pglobal = param->global;

servers[param->id].conf.port = port;

servers[param->id].conf.hostname = hostname;

servers[param->id].conf.credentials = credentials;

servers[param->id].conf.www\_folder = www\_folder;

servers[param->id].conf.nocommands = nocommands;

OPRINT("www-folder-path......: %s\n", (www\_folder == NULL) ? "disabled" : www\_folder);

OPRINT("HTTP TCP port........: %d\n", ntohs(port));

OPRINT("HTTP Listen Address..: %s\n", hostname);

OPRINT("username:password....: %s\n", (credentials == NULL) ? "disabled" : credentials);

OPRINT("commands.............: %s\n", (nocommands) ? "disabled" : "enabled");

param->global->out[id].name = malloc((strlen(OUTPUT\_PLUGIN\_NAME) + 1) \* sizeof(char));

sprintf(param->global->out[id].name, OUTPUT\_PLUGIN\_NAME);

if(!(input\_number < pglobal->incnt)) {

OPRINT("ERROR: the %d input\_plugin number is too much only %d plugins loaded\n", input\_number, param->global->incnt);

return 1;

}

OPRINT("output folder.....: %s\n", folder);

OPRINT("input plugin.....: %d: %s\n", input\_number, pglobal->in[input\_number].plugin);

OPRINT("delay after save..: %d\n", delay);

if (mjpgFileName == NULL) {

if(ringbuffer\_size > 0) {

OPRINT("ringbuffer size...: %d to %d\n", ringbuffer\_size, ringbuffer\_size + ringbuffer\_exceed);

} else {

OPRINT("ringbuffer size...: %s\n", "no ringbuffer");

}

} else {

char \*fnBuffer = malloc(strlen(mjpgFileName) + strlen(folder) + 3);

sprintf(fnBuffer, "%s/%s", folder, mjpgFileName);

OPRINT("output file.......: %s\n", fnBuffer);

if((fd = open(fnBuffer, O\_CREAT | O\_RDWR | O\_TRUNC, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH)) < 0) {

OPRINT("could not open the file %s\n", fnBuffer);

free(fnBuffer);

return 1;

}

free(fnBuffer);

}

param->global->out[id].parametercount = 2;

param->global->out[id].out\_parameters = (control\*) calloc(2, sizeof(control));

control take\_ctrl;

take\_ctrl.group = IN\_CMD\_GENERIC;

take\_ctrl.menuitems = NULL;

take\_ctrl.value = 1;

take\_ctrl.class\_id = 0;

take\_ctrl.ctrl.id = OUT\_FILE\_CMD\_TAKE;

take\_ctrl.ctrl.type = V4L2\_CTRL\_TYPE\_BUTTON;

strcpy((char\*) take\_ctrl.ctrl.name, "Take snapshot");

take\_ctrl.ctrl.minimum = 0;

take\_ctrl.ctrl.maximum = 1;

take\_ctrl.ctrl.step = 1;

take\_ctrl.ctrl.default\_value = 0;

param->global->out[id].out\_parameters[0] = take\_ctrl;

control filename\_ctrl;

filename\_ctrl.group = IN\_CMD\_GENERIC;

filename\_ctrl.menuitems = NULL;

filename\_ctrl.value = 1;

filename\_ctrl.class\_id = 0;

filename\_ctrl.ctrl.id = OUT\_FILE\_CMD\_FILENAME;

filename\_ctrl.ctrl.type = V4L2\_CTRL\_TYPE\_STRING;

strcpy((char\*) filename\_ctrl.ctrl.name, "Filename");

filename\_ctrl.ctrl.minimum = 0;

filename\_ctrl.ctrl.maximum = 32;

filename\_ctrl.ctrl.step = 1;

filename\_ctrl.ctrl.default\_value = 0;

param->global->out[id].out\_parameters[1] = filename\_ctrl;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Description.: this will stop the server thread, client threads

will not get cleaned properly, because they run detached and

no pointer is kept. This is not a huge issue, because this

funtion is intended to clean up the biggest mess on shutdown.

Input Value.: id determines which server instance to send commands to

Return Value: always 0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int output\_stop(int id)

{

DBG("will cancel server thread #%02d\n", id);

pthread\_cancel(servers[id].threadID);

pthread\_cancel(worker);

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Description.: This creates and starts the server thread

Input Value.: id determines which server instance to send commands to

Return Value: always 0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int output\_run(int id)

{

DBG("launching server thread #%02d\n", id);

/\* create thread and pass context to thread function \*/

pthread\_create(&(servers[id].threadID), NULL, server\_thread, &(servers[id]));

pthread\_detach(servers[id].threadID);

pthread\_create(&worker, 0, worker\_thread, NULL);

pthread\_detach(worker);

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Description.: This is just an example function, to show how the output

plugin could implement some special command.

If you want to control some GPIO Pin this is a good place to

implement it. Dont forget to add command types and a mapping.

Input Value.: cmd is the command type

id determines which server instance to send commands to

Return Value: 0 indicates success, other values indicate an error

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int output\_cmd(int plugin, unsigned int control\_id, unsigned int group, int value)

{

char valueStr[] = "/home/cj/picture";

DBG("command (%d, value: %d) for group %d triggered for plugin instance #%02d\n", control\_id, value, group, plugin);

int i = 0; switch(group) {

case IN\_CMD\_GENERIC:

for(i = 0; i < pglobal->out[plugin].parametercount; i++) {

if((pglobal->out[plugin].out\_parameters[i].ctrl.id == control\_id) && (pglobal->out[plugin].out\_parameters[i].group == IN\_CMD\_GENERIC)) {

DBG("Generic control found (id: %d): %s\n", control\_id, pglobal->out[plugin].out\_parameters[i].ctrl.name);

switch(control\_id) {

case OUT\_FILE\_CMD\_TAKE: {

if (valueStr != NULL) {

int frame\_size = 0;

unsigned char \*tmp\_framebuffer = NULL;

if(pthread\_mutex\_lock(&pglobal->in[input\_number].db)) {

DBG("Unable to lock mutex\n");

return -1;

}

/\* read buffer \*/

frame\_size = pglobal->in[input\_number].size;

/\* check if buffer for frame is large enough, increase it if necessary \*/

if(frame\_size > max\_frame\_size) {

DBG("increasing buffer size to %d\n", frame\_size);

max\_frame\_size = frame\_size + (1 << 16);

if((tmp\_framebuffer = realloc(frame, max\_frame\_size)) == NULL) {

pthread\_mutex\_unlock(&pglobal->in[input\_number].db);

LOG("not enough memory\n");

return -1;

}

frame = tmp\_framebuffer;

}

/\* copy frame to our local buffer now \*/

memcpy(frame, pglobal->in[input\_number].buf, frame\_size);

/\* allow others to access the global buffer again \*/

pthread\_mutex\_unlock(&pglobal->in[input\_number].db);

DBG("writing file: %s\n", valueStr);

int fd;

/\* open file for write \*/

if((fd = open(valueStr, O\_CREAT | O\_RDWR | O\_TRUNC, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH)) < 0) {

OPRINT("could not open the file %s\n", valueStr);

return -1;

}

/\* save picture to file \*/

if(write(fd, frame, frame\_size) < 0) {

OPRINT("could not write to file %s\n", valueStr);

perror("write()");

close(fd);

return -1;

}

close(fd);

} else {

DBG("No filename specified\n");

return -1;

}

} break;

case OUT\_FILE\_CMD\_FILENAME: {

DBG("Not yet implemented\n");

return -1;

} break;

default: {

DBG("Unknown command\n");

return -1;

} break;

}

DBG("Ctrl %s new value: %d\n", pglobal->out[plugin].out\_parameters[i].ctrl.name, value);

return 0;

}

}

DBG("Requested generic control (%d) did not found\n", control\_id);

return -1;

break;

}

return 0;

}