Picture Grabber

Listing D-1 provides the complete code discussed in the “A Program to Capture Images” section in Chapter 7. The parts required by webcam C270 are highlighted in bold.

Listing D-1. picture_grabber.c

```
/* V4L2 video picture grabber
```

```
Copyright (C) 2009 Mauro Carvalho Chehab <mchehab@infradead.org>
```

```
This program is free software; you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation version 2 of the License.
```

```
This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.
```

```
Changed by Manoel Ramon:
```

- added command line support see usage() function for details.
- added YUYV or RGB pixel format support
- added conversion function from yuyv_to_rgb24() function extracted from OpenCV, copied without any changes under Intel licenses from the file cvcap_v4l.cpp.

```
For reference the whole file can be visualized on this link:
```

```
https://code.ros.org/trac/opencv/browser/trunk/opencv/src/highgui/  
cvcap\_v4l.cpp
```

```
*/
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <fcntl.h>  
#include <errno.h>  
#include <sys/ioctl.h>
```

```

#include <sys/types.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <linux/videodev2.h>
#include <libv4l2.h>
#include <getopt.h>           /* getopt_long() */
#include <errno.h>
#include <unistd.h>

int images_count;

static const char short_options[] = "W:H:yc:d:h";

static const struct option
long_options[] = {
    { "width",  required_argument, NULL, 'W' },
    { "height", required_argument, NULL, 'H' },
    { "yuyv",   no_argument,      NULL, 'y' },
    { "count",  required_argument, NULL, 'c' },
    { "device", required_argument, NULL, 'd' },
    { "help",   no_argument,      NULL, 'h' },
    { 0, 0, 0, 0 }
};

#define CLEAR(x) memset(&(x), 0, sizeof(x))

static char *dev_name;

/* convert from 4:2:2 YUYV interlaced to RGB24 */
/* based on ccvt_yuyv_bgr32() from camstream */
#define SAT(c) \
    if (c & (~255)) { if (c < 0) c = 0; else c = 255; }

static void
yuyv_to_rgb24 (int width, int height, unsigned char *src, unsigned char *dst)
{
    unsigned char *s;
    unsigned char *d;
    int l, c;
    int r, g, b, cr, cg, cb, y1, y2;

    l = height;
    s = src;
    d = dst;

```

```

while (l--) {
    c = width >> 1;
    while (c--) {
        y1 = *s++;
        cb = ((*s - 128) * 454) >> 8;
        cg = (*s++ - 128) * 88;
        y2 = *s++;
        cr = ((*s - 128) * 359) >> 8;
        cg = (cg + (*s++ - 128) * 183) >> 8;

        r = y1 + cr;
        b = y1 + cb;
        g = y1 - cg;
        SAT(r);
        SAT(g);
        SAT(b);

        *d++ = b;
        *d++ = g;
        *d++ = r;

        r = y2 + cr;
        b = y2 + cb;
        g = y2 - cg;
        SAT(r);
        SAT(g);
        SAT(b);

        *d++ = b;
        *d++ = g;
        *d++ = r;
    }
}

struct buffer {
    void *start;
    size_t length;
};

static void xioctl(int fh, int request, void *arg)
{
    int r;

    do {
        r = v4l2_ioctl(fh, request, arg);
    } while (r == -1 && ((errno == EINTR) || (errno == EAGAIN)));
}

```

```

    if (r == -1) {
        fprintf(stderr, "error %d, %s\n", errno, strerror(errno));
        exit(EXIT_FAILURE);
    }
}

```

```

static void usage(FILE *fp, int argc, char **argv)
{
    fprintf(fp,
        "Usage: %s [options]\n\n"
        "Version adapted to Intel Galileo\n"
        "Options:\n"
        "-d | --device name    Video device name\n"
        "-W | --width          Sets image width\n"
        "-H | --height         Sets image height\n"
        "-y | --yuyv           Sets YUYV encode, if not used the\n"
        "                      encode if RGB24\n"
        "-c | --count          Sets number of images to be\n"
        "                      captured\n"
        "-h | --help           Prints this help\n"
        "", argv[0]);
}

```

```

static void errno_exit(const char *s)
{
    fprintf(stderr, "%s error %d, %s\n", s, errno, strerror(errno));
    exit(EXIT_FAILURE);
}

```

```

int main(int argc, char **argv)
{
    struct v4l2_format          fmt;
    struct v4l2_buffer          buf;
    struct v4l2_requestbuffers  req;
    enum v4l2_buf_type          type;
    fd_set                     fds;
    struct timeval              tv;
    int                         r, fd = -1;
    unsigned int                i, n_buffers;
    char                        out_name[256];
    FILE                        *fout;
    struct buffer                *buffers;
    unsigned char               isYUYV = 0;
    int                         width = 1280;
    int                         height = 720;

    dev_name = "/dev/video0";
    images_count = 10;
}

```

```

for (;;) {
    int idx;
    int c;

    c = getopt_long(argc, argv,
                    short_options, long_options, &idx);

    if (-1 == c)
        break;

    switch (c) {
    case 0: /* getopt_long() flag */
        break;

    case 'W':
        errno = 0;
        width = strtol(optarg, NULL, 0);
        if (errno)
            errno_exit(optarg);
        break;

    case 'H':
        errno = 0;
        height = strtol(optarg, NULL, 0);
        if (errno)
            errno_exit(optarg);
        break;

    case 'y':
        isYUYV = 1;
        break;

    case 'c':
        errno = 0;
        images_count = strtol(optarg, NULL, 0);
        if (errno)
            errno_exit(optarg);
        break;

    case 'd':
        dev_name = optarg;
        break;

    case 'h':
        usage(stdout, argc, argv);
        exit(EXIT_SUCCESS);
    }
}

```

```

        default:
            usage(stderr, argc, argv);
            exit(EXIT_FAILURE);
    }
}

fd = v4l2_open(dev_name, O_RDWR | O_NONBLOCK, 0);
if (fd < 0) {
    perror("Cannot open device");
    exit(EXIT_FAILURE);
}

CLEAR(fmt);
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.width = width;
fmt.fmt.pix.height = height;

if (!isYUVV)
{
    printf("Encode RGB24\n");
    fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_RGB24;
}
else
{
    printf("Encode YUYV\n");
    fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
}
fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;
xiocctl(fd, VIDIOC_S_FMT, &fmt);
if (fmt.fmt.pix.pixelformat != V4L2_PIX_FMT_RGB24 &&
    fmt.fmt.pix.pixelformat != V4L2_PIX_FMT_YUYV) {
    printf("Libv4l didn't accept RGB24 or YUYV format. Can't
    proceed.\n");
    exit(EXIT_FAILURE);
}
if ((fmt.fmt.pix.width != width) || (fmt.fmt.pix.height != height))
    printf("Warning: driver is sending image at %dx%d\n",
        fmt.fmt.pix.width, fmt.fmt.pix.height);

CLEAR(req);
req.count = 5;
req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
req.memory = V4L2_MEMORY_MMAP;
xiocctl(fd, VIDIOC_REQBUFS, &req);

```

```

buffers = calloc(req.count, sizeof(*buffers));
if (!buffers) {
    fprintf(stderr, "Out of memory\n");
    exit(EXIT_FAILURE);
}

for (n_buffers = 0; n_buffers < req.count; ++n_buffers) {
    CLEAR(buf);

    buf.type      = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory    = V4L2_MEMORY_MMAP;
    buf.index     = n_buffers;

    xioctl(fd, VIDIOC_QUERYBUF, &buf);

    buffers[n_buffers].length = buf.length;
    buffers[n_buffers].start = v4l2_mmap(NULL, buf.length,
        PROT_READ | PROT_WRITE, MAP_SHARED,
        fd, buf.m.offset);

    if (MAP_FAILED == buffers[n_buffers].start) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }
}

for (i = 0; i < n_buffers; ++i) {
    CLEAR(buf);
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory = V4L2_MEMORY_MMAP;
    buf.index = i;
    xioctl(fd, VIDIOC_QUEBUF, &buf);
}
type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

xioctl(fd, VIDIOC_STREAMON, &type);

for (i = 0; i < images_count; i++) {
    do {
        FD_ZERO(&fds);
        FD_SET(fd, &fds);

        // Timeout.
        tv.tv_sec = 2;
        tv.tv_usec = 0;
    } while (select(fd+1, &fds, NULL, NULL, &tv) < 0);
}

```

```

        r = select(fd + 1, &fds, NULL, NULL, &tv);
    } while ((r == -1 && (errno = EINTR)));
    if (r == -1) {
        perror("select");
        return errno;
    }

    CLEAR(buf);
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory = V4L2_MEMORY_MMAP;
    xioctl(fd, VIDIOC_DQBUF, &buf);

    sprintf(out_name, "out%03d.ppm", i);
    printf("Creating image: %s\n", out_name);
    fout = fopen(out_name, "w");
    if (!fout) {
        perror("Cannot open image");
        exit(EXIT_FAILURE);
    }
    fprintf(fout, "P6\n%d %d 255\n", fmt.fmt.pix.width, fmt.fmt.
    pix.height);

    if (isYUVV)
    {
        // each pixel 3 bytes in RGB 24
        int size = fmt.fmt.pix.width * fmt.fmt.pix.height *
        sizeof(char) * 3;
        unsigned char * data = (unsigned char *) malloc(size);

        yuyv_to_rgb24(fmt.fmt.pix.width,
            fmt.fmt.pix.height,
            (unsigned char*)(buffers[buf.index].start),
            data);

        fwrite(data, size, 1, fout);

        free (data);
    }
    else
    {
        fwrite(buffers[buf.index].start, buf.bytesused, 1, fout);
    }
    fclose(fout);

    xioctl(fd, VIDIOC_QBUF, &buf);
}

```



```
type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
xioctl(fd, VIDIOC_STREAMOFF, &type);
for (i = 0; i < n_buffers; ++i)
    v4l2_munmap(buffers[i].start, buffers[i].length);
v4l2_close(fd);

return 0;
}
```