```
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "std_image_write.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <linux/videodev2.h>
#include <math.h>

#define cam_x 1920
#define cam_y 1080

unsigned char* arr[2];

int open_handle()
{
    // open the device handel
    int cameraHandle = open("/dev/video0",O_RDWR,0);

    // set a supported video format
    struct v4l2_format format;
    memset(&format,0,sizeof(format));
    format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    format.fmt.pix.width = cam_x;
    format.fmt.pix.height = cam_y;
    format.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;//MJPG
    format.fmt.pix.field= V4L2_FIELD_ANY;


    if (ioctl(cameraHandle, VIDIOC_S_FMT,&format)<0)
    {
        printf("Vidioc_s_fmt video format set fail\n");
        return -1;
    }
    return cameraHandle;
}

int request(int cameraHandle)
{
    // request creation of two buffers in the video driver
    struct v4l2_requestbuffers req;
    memset(&req,0,sizeof(req));
    req.count=2;
    req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    req.memory = V4L2_MEMORY_MMAP;
    if (ioctl(cameraHandle,VIDIOC_REQBUFS,&req)<0)
    {
        printf("VIDIOC_REQBUFS failed!\n");
        return -1;
    }
    return 0;
}
```

```
int set_query(int cameraHandle)
{

    for (int i=0;i<2;i++)
    {
    // query the created buffers

        struct v4l2_buffer buf;
        memset(&buf,0,sizeof(buf));
        buf.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory=V4L2_MEMORY_MMAP;
        buf.index=i;
        if (ioctl(cameraHandle,VIDIOC_QUERYBUF,&buf)<0)
        {
            printf("VIDIOC_QUERYBUF failed!\n");
            return -1;
        }
        arr[i] = mmap(NULL,buf.length,PROT_READ|
PROT_WRITE,MAP_SHARED,cameraHandle,buf.m.offset);
    }
    return 0;
}

int queue_up(int cameraHandle,int i)
{
    // queue up buffers that the driver should write to
    struct v4l2_buffer buf;
    memset(&buf,0,sizeof(buf));
    buf.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory=V4L2_MEMORY_MMAP;
    buf.index=i;
    if (ioctl(cameraHandle,VIDIOC_QBUF,&buf)<0)
    {
        printf("VIDEOC_OBUF failed!\n");
        return -1;
    }
    return 0;
}

int start_camera(int cameraHandle)
{
    // start camera
    enum v4l2_buf_type type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
    if (ioctl(cameraHandle,VIDIOC_STREAMON,&type)<0)
    {
        printf("VIDOC_STREAMON failed!\n");
        return -1;
    }
    return 0;
 }

typedef struct Pixel
    {
        unsigned char R;
        unsigned char G;
        unsigned char B;
        unsigned char A;

    } Pixel;
```

```
static void YUYVtoRGB(unsigned char y, unsigned char u, unsigned char v, Pixel*
_rgba)
    #define MIN(a, b) (((a) < (b)) ? (a) : (b))
    #define MAX(a, b) (((a) < (b)) ? (b) : (a))
    {
        int c = y - 16;
        int d = u - 128;
        int e = v - 128;

        _rgba->A = 255; //Alpha
        _rgba->R = MAX(0, MIN((298 * c + 409 * e + 128) >> 8, 255));
        _rgba->G = MAX(0, MIN((298 * c - 100 * d - 208 * e + 128) >> 8, 255));
        _rgba->B = MAX(0, MIN((298 * c + 516 * d + 128) >> 8, 255));
    }

int ProcessImage(const unsigned char *_yuv, int _size)
    {
        #define RGB_SIZE cam_x * cam_y
        static Pixel rgbConversion[RGB_SIZE];
        //Pixel* data=rgbConversion;//man behöver inte har en pekare funkar också
utan
        int rgbIndex = 0;
        for (int i = 0; i < _size; i += 4)

        {
            unsigned char y1 = _yuv[i + 0];
            unsigned char u = _yuv[i + 1];
            unsigned char y2 = _yuv[i + 2];
            unsigned char v = _yuv[i + 3];

            YUYVtoRGB(y1, u, v, &rgbConversion[rgbIndex++]);
            YUYVtoRGB(y2, u, v, &rgbConversion[rgbIndex++]);

        }
        int a=(cam_x/2);
        int b=(cam_y/2);
        int r1=40*40;
        int r2=50*50;

        // loopar genom alla pixlar

        for (int y=0;y<cam_y;y+=1)
        {
            // ekvationen (x+a)^2 + (y+b)^2 =r^2
            for (int x=0;x<cam_x;x+=1)
            {
                //printf("punkt: %d,%d \n",x,y);
                int rgb_size= (cam_x * (y))+(x);
                int v_led=(x-a)*(x-a)+(y-b)*(y-b);

                if (v_led>r1 && v_led<r2)
                {
                    //printf("rgb_size: %d,%d\n",x,y);
                    rgbConversion[rgb_size].R = 255;
                    rgbConversion[rgb_size].B = 0;
                    rgbConversion[rgb_size].G = 0;
```

```
                }


            }



        }

        stbi_write_png("bild.png",cam_x,cam_y,4,rgbConversion,cam_x*4);
        // rgbConversion now has the correct rgba data here (if you used
V4L2_PIX_FMT_YUYV). Use the data to create a PNG image that is saved to disk.
        return 0;
    }

int dequeue(int cameraHandle,int i)
{
    // if we want to start a video so we create a while loop here

    // dequeue a buffer(get the image data)
    struct v4l2_buffer buf;
    memset(&buf,0,sizeof(buf));
    buf.type= V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory= V4L2_MEMORY_MMAP;

    if (ioctl(cameraHandle,VIDIOC_DQBUF,&buf)<0)
    {
        return errno;
    }

    ProcessImage(arr[i],buf.bytesused);
    return 0;
}

int main()
{
    int camera = open_handle();
    request(camera);
    set_query(camera);
    queue_up(camera,0);
    start_camera(camera);
    dequeue(camera,0);
}
```