Date: 2018-08-16

Author: Stefan Hagenmüller, CarMaker Service, Germany

Release No.: CM-7.0.1

Extracting RGB information from images provided by the Camera Raw Signal Interface

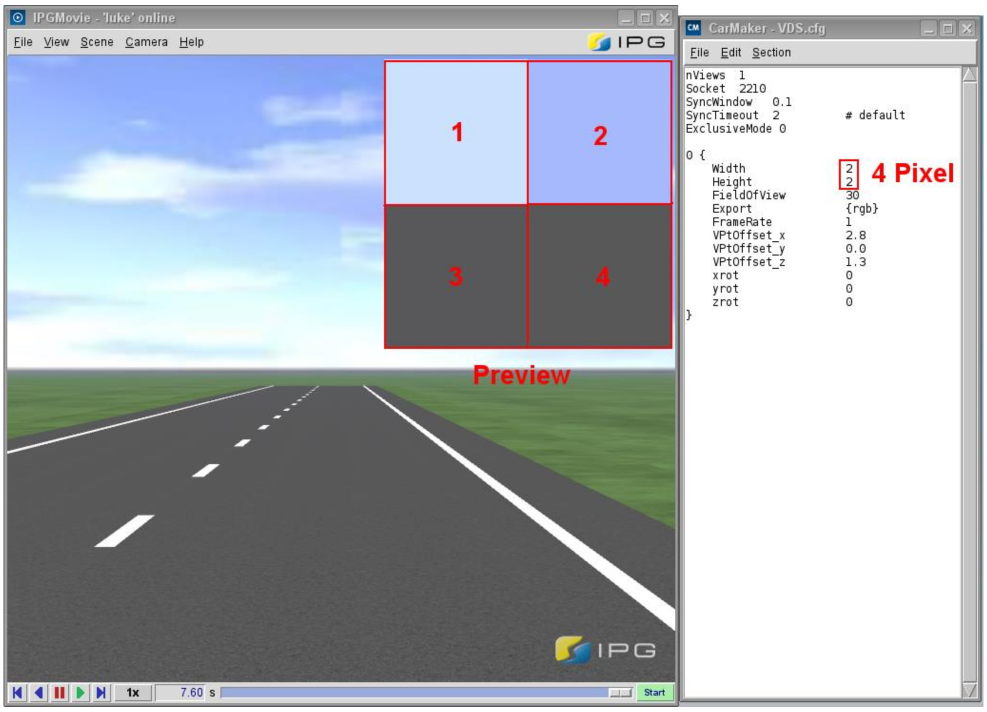
This document describes an easy way to extract the RGB information for each pixel from images generated by the camera module and transferred with the help of a VDS channel. This example is based on the VDS standalone client C-Code example provided with the CarMaker installation. There are just very few steps necessary.

Technical Background

The Video Data Stream provides through its channels a continuous stream of pixel data. This data can have several formats e.g. depth, RGB, raw and many more. This example works with RGB data.

Solution

For better comprehension the used environment should be configured as simple as possible. The configuration of a VDS channel is done within the VDS.cfg file of the *Movie* folder. If there's no such file available at the moment, you can let one be created by the *Vehicle Data Set* (*Sensors / Camera RSI / Edit VDS Configuration*). In terms of keeping it simple we'll use a pixel grid of just 4 pixels (width of 2 and height of 2) in total. In order to avoid getting to much data in a very short period of time, we'll set the frame rate to 1 fps. Please see the following picture (Fig.1) to figure out how this simple configuration could look like.



```
nViews  1
Socket  2210
SyncWindow   0.1
SyncTimeout  2            # default
ExclusiveMode 0

0 {
    Width              2
    Height             2
    FieldOfView        30
    Export             {rgb}
    FrameRate          1
    VPtOffset_x        2.8
    VPtOffset_y        0.0
    VPtOffset_z        1.3
    xrot               0
    yrot               0
    zrot               0
}
```

Fig. 1: VDS.cfg for the used example

To identify which channels contain which color tones, additional software tools (like GIMP) can be used. Therefore, the 4 pixels contain the following values for RGB:

1. Pixel: (203 / 226 / 255)
2. Pixel: (165 / 185 / 252)
3. Pixel (88 / 88 / 89)
4. Pixel (88 / 88 / 88)

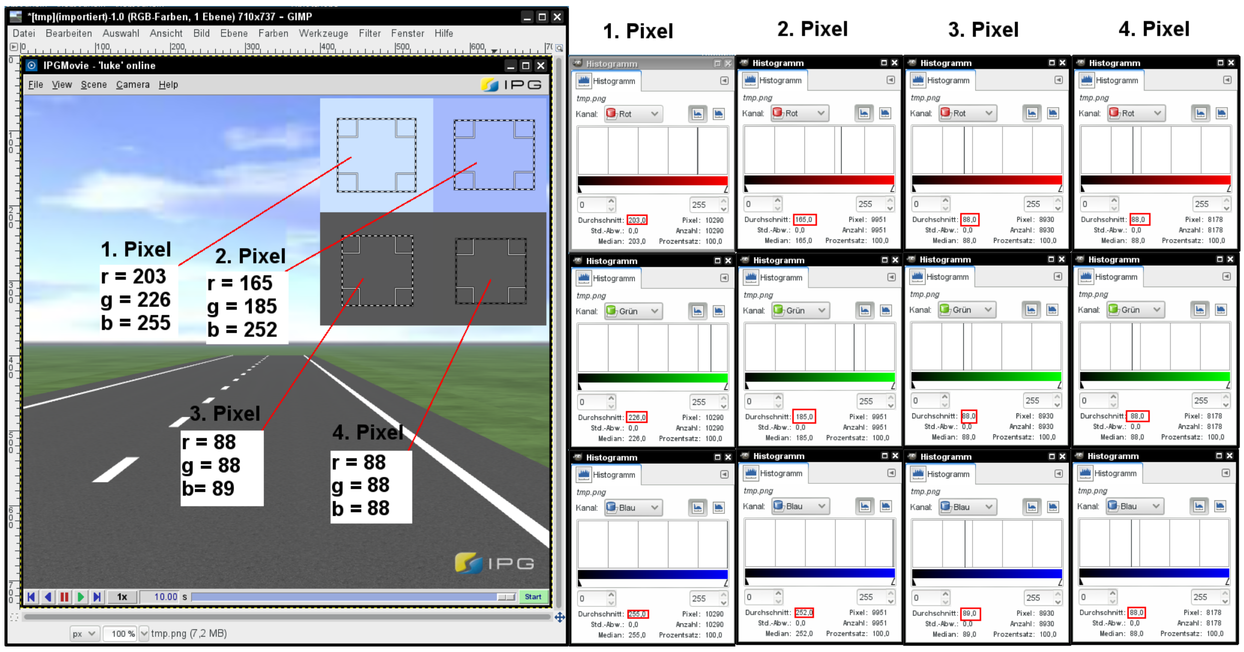


Fig. 2: RGB Code of each pixel measured with GIMP

The *IPGMovie Manual* describes how to compile and run a *VDS Standalone Client* (Chapter 5.4: *Video Data Stream Client*). The example client, which is located in the installation directory, can be found here:

/ <InstallDir> / hil / <version> / Examples / VDS / vds-client-standalone.c

The example standalone client is prepared to handle depth information, figuring out which is the closest pixel to the virtual camera lens. For the purpose of checking the RGB channel information, there are some small modifications in the example template to be made. Since the standard example uses depth information we have to modify the *strcmp* part so that the image type (*ImgType*) corresponds to “rgb” instead to “depth”. In addition, there are some slight changes within the Variables for Image Processing. The main data processing code is inside the if-case

```
29:                    if (len == ImgLen) {
… :
50:                    }
```

The corresponding changes are marked as bold letters in the following C-Code example.

```c
static int
VDS_GetData(void)
{
  int len = 0;
  int res = 0;

  /* Variables for Image Processing */
  char    ImgType[64];
  int     ImgWidth, ImgHeight, Channel;
  float   SimTime;
  int     ImgLen;
  int     Pixel;
  int     ColorChannel;
  float  *f_img;

  if (sscanf(VDScfg.sbuf, "*VDS %d %s %f %dx%d %d", &Channel,
        ImgType, &SimTime, &ImgWidth, &ImgHeight, &ImgLen) == 6) {
    if (0)
      printf ("%6.3f %d: %8s %dx%d %d\n",
      SimTime, Channel, ImgType, ImgWidth, ImgHeight, ImgLen);
    if (ImgLen > 0) {
      if (strcmp(ImgType, "rgb") == 0) {
        char *img = (char *)malloc(ImgLen);
        for (len=0; len<ImgLen; len+=res) {
          if ((res=recv(VDScfg.sock, img+len, ImgLen-len, VDScfg.RecvFlags)) < 0) {
            printf("VDS: Socket Reading Failure\n");
            break;
          }
        }
        if (len == ImgLen) {

         /* Initialize Pixel and ColorChannel information */
         Pixel = 1;
         ColorChannel = 0;

         /* Walk through all ColorChannel of each pixel */
          while (ColorChannel < ImgWidth * ImgHeight * 3) {

          /* Get the information of each ColorChannel */
           unsigned char R = img[ColorChannel + 0];
           unsigned char G = img[ColorChannel + 1];
           unsigned char B = img[ColorChannel + 2];

          /* Print the information */
          printf("Time = %f, Pixel No. = %d,
              Channels: R = %d, G = %d, B = %d \n",
              SimTime, Pixel, R, G, B);

          /* Prepare for next loop step */
          ColorChannel = ColorChannel + 3;
          Pixel++;
          }
        }
        free (img);
      }
    }
    VDSIF.nImages++;
    VDSIF.nBytes += len;
  } else {
    printf ("VDS: not handled: %s\n",VDScfg.sbuf);
  }
```

The distribution of the pixel information is shown in the IPGMovie Manual, Chapter 5.5 *“Detailed Description of the Output Formats”.* Therefore, the lines

```
while (ColorChannel < ImgWidth * ImgHeight * 3.0) {

/* Get the information of each ColorChannel */
  unsigned char R = img[ColorChannel + 0];
  unsigned char G = img[ColorChannel + 1];
  unsigned char B = img[ColorChannel + 2];
```

are used to walk through each color channel (channel information of R, G and B) and save the information within the unsigned char variables R, G and B.

The printf function additionally prints the output to the shell

```
/* Print the information */
printf("Time = %f, Pixel No. = %d,
        Channels: R = %d, G = %d, B = %d \n",
        SimTime, Pixel, R, G, B);
```

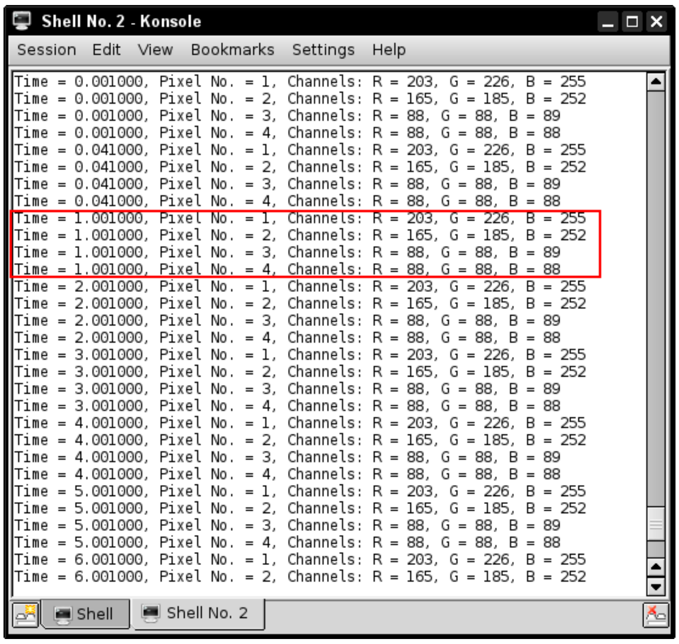
After compiling the new executable via the command

gcc vds-client-standalone.c -o vds-client-standalone

for Linux or

gcc vds-client-standalone.c -o vds-client-standalone -lws2_32

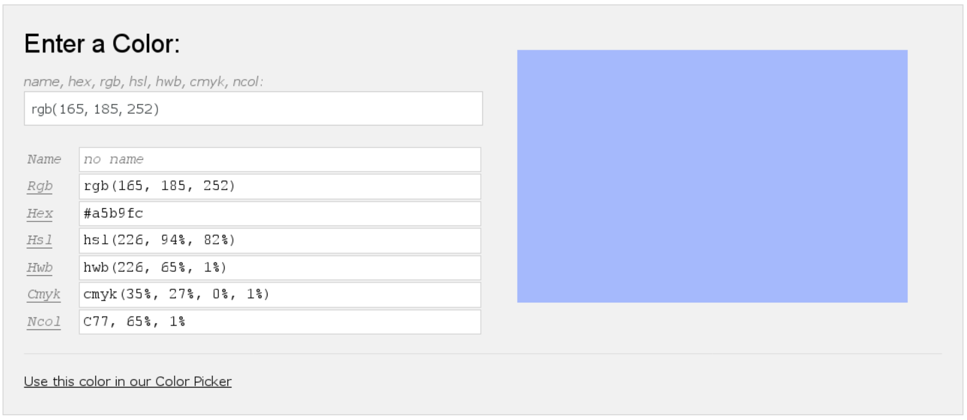
for Windows we can run the program in the shell and obtain the contents of the RGB channels. The gathered information in the shell equals that which we’ve measured in the picture previously in GIMP. Fig. 3 shows the console output generated by the *printf*-calls.



```
Time = 0.001000, Pixel No. = 1, Channels: R = 203, G = 226, B = 255
Time = 0.001000, Pixel No. = 2, Channels: R = 165, G = 185, B = 252
Time = 0.001000, Pixel No. = 3, Channels: R = 88, G = 88, B = 89
Time = 0.001000, Pixel No. = 4, Channels: R = 88, G = 88, B = 88
Time = 0.041000, Pixel No. = 1, Channels: R = 203, G = 226, B = 255
Time = 0.041000, Pixel No. = 2, Channels: R = 165, G = 185, B = 252
Time = 0.041000, Pixel No. = 3, Channels: R = 88, G = 88, B = 89
Time = 0.041000, Pixel No. = 4, Channels: R = 88, G = 88, B = 88
Time = 1.001000, Pixel No. = 1, Channels: R = 203, G = 226, B = 255
Time = 1.001000, Pixel No. = 2, Channels: R = 165, G = 185, B = 252
Time = 1.001000, Pixel No. = 3, Channels: R = 88, G = 88, B = 89
Time = 1.001000, Pixel No. = 4, Channels: R = 88, G = 88, B = 88
Time = 2.001000, Pixel No. = 1, Channels: R = 203, G = 226, B = 255
Time = 2.001000, Pixel No. = 2, Channels: R = 165, G = 185, B = 252
Time = 2.001000, Pixel No. = 3, Channels: R = 88, G = 88, B = 89
Time = 2.001000, Pixel No. = 4, Channels: R = 88, G = 88, B = 88
Time = 3.001000, Pixel No. = 1, Channels: R = 203, G = 226, B = 255
Time = 3.001000, Pixel No. = 2, Channels: R = 165, G = 185, B = 252
Time = 3.001000, Pixel No. = 3, Channels: R = 88, G = 88, B = 89
Time = 3.001000, Pixel No. = 4, Channels: R = 88, G = 88, B = 88
Time = 4.001000, Pixel No. = 1, Channels: R = 203, G = 226, B = 255
Time = 4.001000, Pixel No. = 2, Channels: R = 165, G = 185, B = 252
Time = 4.001000, Pixel No. = 3, Channels: R = 88, G = 88, B = 89
Time = 4.001000, Pixel No. = 4, Channels: R = 88, G = 88, B = 88
Time = 5.001000, Pixel No. = 1, Channels: R = 203, G = 226, B = 255
Time = 5.001000, Pixel No. = 2, Channels: R = 165, G = 185, B = 252
Time = 5.001000, Pixel No. = 3, Channels: R = 88, G = 88, B = 89
Time = 5.001000, Pixel No. = 4, Channels: R = 88, G = 88, B = 88
Time = 6.001000, Pixel No. = 1, Channels: R = 203, G = 226, B = 255
Time = 6.001000, Pixel No. = 2, Channels: R = 165, G = 185, B = 252
```

Fig. 3: Shell output for pixel 1-4

To verify the color code for one particular pixel the internet provides a lot of color code generators that are able to show the proper color (Fig. 4).



Enter a Color:

name, hex, rgb, hsl, hwb, cmyk, ncol:

rgb(165, 185, 252)

Name	no name
Rgb	rgb(165, 185, 252)
Hex	#a5b9fc
Hsl	hsl(226, 94%, 82%)
Hwb	hwb(226, 65%, 1%)
Cmyk	cmyk(35%, 27%, 0%, 1%)
Ncol	C77, 65%, 1%

Use this color in our Color Picker

Fig. 4: Source: https://www.w3schools.com