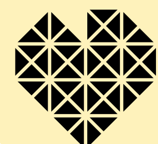
Test Image Generator ARMmovie.c

for a Vis-NIR spectral camera development

JARMO ALANDER | SEVERI SUTINEN | DÁNIEL TISZA



Vaasan yliopisto
UNIVERSITY OF VAASA

UNIVERSITY OF VAASA REPORTS 33

Test Image Generator ARMmovie.c: for a Vis-NIR spectral camera development.

Publisher:

University of Vaasa.

School of Technology and Innovations, Automation Technology.

Authors:

Jarmo Alander, Severi Sutinen, Dániel Tisza.

ISBN 978-952-395-040-5 (online)

URN http://urn.fi/URN:ISBN:978-952-395-040-5

ISSN 2489-2580 (University of Vaasa Reports 33, print)

Contents

III

1 Introduction 1
1.1 FPGA . . . 1
1.2 Testing . . . 3
1.3 Tools . . . 4
1.4 Related work . . . 5

2 Camera 6
2.1 Interference filter . . . 6

3 `ARMmovie.c`, a test image generator 10
3.1 Control . . . 10
3.2 Simulation of the camera . . . 11
3.3 Execution profile . . . 12
3.4 Parameter input . . . 12
3.5 Image comparison and animation . . . 13
3.6 Camera simulation . . . 14
3.7 LED simulation . . . 21
3.7.1 Camera with LEDs simulation . . . 22
3.8 RGB sensitivity analysis . . . 22

4 Running ARMmovie.c 25
4.1 Running procedure . . . 25

5 Conclusions and Future 27
5.1 Future . . . 27

References 28

A Program files 32

List of Figures

1 ICT energy use forecast . . . 2
2 The multispectral camera imaging a 20€ note . . . 6
3 The camera body daA2500-14uc . . . 7
4 Camera sensor sensitivities of each RGB channel pixels (Bayer). . . . 8
5 The principle of Fabry-Pérot interferometer. . . . 8
6 The transmission of etalon as the function of wavelength . . . 9
7 The transmission of etalon as the function of wavenumber . . . 10

8 Test image examples . . . 11
9 Parameter file for `ARMmovie.c` . . . 14
10 Comparing two images thru Magnifier glass. . . . 15
11 Frame of the animation of the spectral camera by ARMmovie.c. . . . 16
12 Sensor sensitivity curves drawn on the original image. . . . 17
13 Sensor sensitivity curves drawn on using AUTOpoint. . . . 17
14 Fitting the original draw data on the original image. . . . 18
15 Fitting the averaged draw data on the original image. . . . 18
16 Fitting final filtered and interpolated draw data on the original image. . . . 19
17 Test image (frame 0) . . . 19
18 Test image (frame 6: 650nm filter setting). . . . 20
19 The sum of channel quantum efficiencies and its inverse. . . . 23
20 Test image leg (frame 17: 800nm filter setting). . . . 24
21 LED spectra simulation for $\lambda_0 = 450$, 500, 600, and 800nm. . . . 24
22 LED spectra simulation for HLMP-3750/-3390/-1340 Series LED Lamps. . . . 25

List of Tables

1 Properties of the camera. . . . 7
2 Execution time profile of `ARMmovie.c`. . . . 13
3 Parameters of the model of high efficiency red LED (Fig. 22). . . . 21

Abstract

This report describes a C language program that can be used in both offline or online generation of test images for a special spectral camera prototype software that is run on a Field Programmable Gate Array (FPGA). The FPGA has two ARM processor cores on the same chip where the program can be run under an operating system, such as Linux, or actually its reduced version called Petalinux, or immediately in a 'baremetal' mode without any operating system as a stand-alone ARM assembler program. It was this flexibility of running modes and the limited memory resources of the FPGA boarsd, which were the main reasons why C language realisation was chosen. The program is designed to be used for both supporting software development and for online selftest type operations in the camera support software run on an FPGA that contains also two traditional ARM processors. The program generates purely synthetic images, or patterns, or can blend real images read from files with synthetic patterns. There are a set of parameters controlling the generation details and they can be input from a file, or they can be introduced via an internal data structure that can be manually tailored before the compilation. The program can generate single images or a sequence of images that can be e.g. externally be combined into a gif animation file.

Keywords: embedded systems, FPGA, image filters, image processing, simulation, SoC-FPGA, testing.

Acknowledgements

This research was funded by the Academy of Finland, grant number 314522: Spectral Imaging of Complex Surface Tomographies (SICSURFIS) project in RADDESS program. Acknowledgements to project companions Heikki Saari, and Roberts Trops at VTT, Anna-Maria Raita-Hakola, Leevi Annala, and Ilkka Pölönen at University of Jyväskylä; and Annamari Ranki at Helsinki University Hospital. Acknowledgements to Petri Välisuo and Juhani Puska for their invaluable help with the boards and software.

1 Introduction

In this project called SICSURFIS developing and applying a filter based spectral camera (Fig. 2), we have concentrated on the development of the software for the configuration of the FPGA hardware used with the camera in general, and in this report especially on the image generation for testing the FPGA software. Before going into the details of the software we will here briefly introduce the camera hardware, mainly the optical aspects of it and the FPGA technology in general.

A conventional RGB (color) camera is imaging using three spectral ranges, red, green, and blue, which is practically enough to record colors as perceived by the human vision system[16]. Spectral cameras are able to image on several to even thousands of spectral wavelength ranges. It means that each pixel is a spectrum of its field of view (FoV). Our spectral camera is actually an ordinary RGB camera that has an interference filter to select the wavelength range to be recorded on an RGB sensor. Therefore its pixel is still an RGB vector, but a filtered one having three channels (R, G, and B) giving optically encoded spectral information that needs careful post processing to be interpreted as intensities as the function of wavelength i.e. as the values of the spectra. Observe, that to record the whole spectrum several images with different filter settings must be taken. The most obvious applications for this type of spectral camera are those in which you need only a very limited number of narrow wavelength bands of the spectra, ideally a couple, not the whole spectra, which would need hundreds of images to be taken. However, due to its principle of operation there can be considerable crosstalk between the RGB channels. It is therefore advisable to filter out those wavelengths that are not needed to be measured or change the camera, actually its cell, to a higher quality one that has less crosstalk[18;35].

1.1 FPGA

The huge volume of digital data that has to be processed puts more and more concern on the energy consumption by digital processing[8;15]. (Fig. 1) This is especially true for mobile devices, which should have both long operation times and light weight batteries. The current share of ICT's carbon footprint is around 2% of global emissions that is comparable with that of the aviation industry[15].

But on what functions or operations the energy in computing is actually used for? Somewhat surprisingly, it is actually not used in the processing of data, but just to move the data around, mainly from memory to the processing element also called the Central Processing Unit (CPU) and then back very soon after the processing. Memory access operations are repeated millions and millions of times e.g. to process an image consisting of millions of pixels. The longer the distance the data is moved the more energy must be used. The energy needed to move a pair of numbers from memory to a processor is roughly about one thousand ($\approx 1,000$) times the energy needed to do a simple operation like addition of the same pair by the processor. It really pays to think and design how to avoid unnecessary moving data and do as much as possible processing near the data

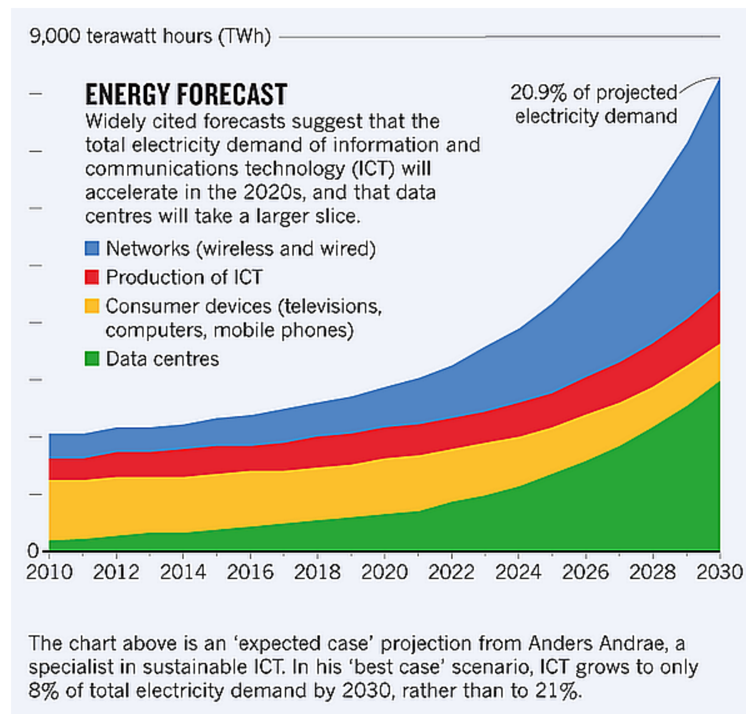


Figure 1: ICT energy use forecast by [15].

source, ideally when the data is still in the registers of the processing element. This is the edge computing paradigm. [11]

Running the test image generator is in line with this approach, because we are avoiding unnecessary data movements over longer distances. The test images needed are generated when ever needed and by the processing element very locally.

Actually we should get rid of the simple program execution altogether because it also causes unnecessarily busy memory traffic. This problem is solved by an FPGA, that has the program all the time already ready within the processing element and not in the memory while running. The down side of this approach is that the programs moved to FPGA should be static and not extremely complex. Program change is possible but it is quite slow and complicated so that most FPGA-applications are based on a single fixed program for simplicity and efficiency. In our case we are trying to do most low level image processing computations within an FPGA connected to the image source, a handheld spectral camera.

However, the test image generation is so complex computational operation and not so frequently needed when compared to other image operations that we are currently not doing it on the FPGA side but by the processor located on the same chip and having a fast data bus access to the FPGA. If later there will be a need for more frequent test image generation then an FPGA based generator might be considered.

Programming an FPGA is called configuring and the compilation into a functioning circuit as synthesis. It means that the hardware elements to do the basic digital processing, like logic gates, multiplexers, adders etc already exists on the FPGA chip. What is missing is the blueprint how to connect those primitive elements to realise a computing device. In practice the programming is done using such hardware design languages as VHDL or Verilog that are used to implicitly model the connections between a huge number of primitive logic circuits and modules already realized on the chip by the chip maker. The expression implicit model here means that typically the model being textual is quite abstract resembling more a traditional programming language code that must be compiled into an explicit model than any explicit circuit diagram drawing or blueprint. We are here not going any more details of the FPGA programming; that is left for a few other reports published by our project.

FPGA is ideal when a lot of processing power is needed energy efficiently with the option of reprogrammability and when the product volume is not extremely high. A huge number of industrial electronics and automation devices, like cameras, protection relays, frequency controllers, medical devices, belong to this in many ways very important category of potential FPGA applications. An prominent opposite example is a mobile phone, when the volume is so high that an ASIC is the right choice.

With the introduction of FPGAs with one or more standard (ARM) processors, called SoC-FPGAs, has finally made FPGAs a really powerful and flexible element in modern instrumentation, providing both high computational efficiency (FPGA-part) and flexible user and system interfaces via ARM and Linux or similar operating systems. In our project we are using SoC-FPGAs made by Xilinx, which is nowadays a part of the Advanced Micro Devices, Inc (AMD).

In conclusion of FPGAs, we can say that they can both save energy and give excellent processing speed in certain data intensive applications, like many such found in signal and image processing fields.

1.2 Testing

Testing is a key activity to produce quality software[27]. In embedded software production there are two main activities causing approximately equally high expenses: the programming and the testing. Actually in many cases the testing can even need some more resources that the programming. This is especially true for application having high quality and safety standards, like those in aerospace, military, and medical areas. Here we are dealing with applications that belong mainly the last one: a handheld diagnostic device that hopefully helps medical doctors to make right diagnosis of melanoma[32].

Reference[29] gives the following seven reasons why software testing is important:

- economy: the earlier a bug is fixed the lower the costs,
- security: a well tested program should be more secure and reliable than a less tested,

- quality: is the software running smoothly also in different environments,
- customer satisfaction: a thorough testing also includes user points of views,
- faster development: testing may be run in parallel to development,
- adaptation to new features, when rerunning the tests helps avoid accidental bugs caused by revising and updating the code, and
- last but not least: the determination of the performance of the software.

One famous program development approach called Test Driven Development (TDD) even turns the view upside down: testing is done first and it is not before each test has been implemented that the implementation of the program functions is proceed. TDD approach leads to well tested and documented programs, so to say automatically, because it encourages the programmers to make simple designs easy to test and inspires confidence because testing is done rigorously.[10]

In this report we are concentrating on one aspect of testing image processing software, namely on the generation of synthetic images that can not only be used in the design process of the device but also in the final product to help in calibration and selftesting.

1.3 Tools

The definition of software development tool is of practical nature. In a broad sense it is any software that the programmer is using when creating or maintaining software. Here it mainly means a set of web pages that are used to generate HDL modules and files to be run on FPGA or SoC-FPGA. The first tool presented here is actually a C language program that can be run on the SoC processor or any external PC to generate test images for the system under development.

The current set of the main tools aiding FPGA programming includes

- `ARMmovie.c`: test image generator, the topic of this report,
- `CSD6.html`: constant multiplication code generator for VHDL and SystemVerilog,
- `SpektriMarvin3.html`: Principle Component Analysis (PCA) of images, PCA matrix generation for `CSD6.html`,
- `SaturaViisari.html`: color, like saturation, processing,
- `GAGui.html`: segmentation filter search and optimisation by genetic algorithm, and
- `OpenInsta.html`: multispectral image compression.

Here we are presenting the first one, `ARMmovie.c`, a test image generator.

1.4 Related work

Frolov et al.[14] gives a review of using adversarial generative neural networks to synthesise images from text. That is an Artificial Intelligence (AI) based technology to generate images from textual descriptions and example images used to train the networks. However, that is mainly used for fancy illustrations not for online testing of image processing systems, but it certainly have potential for also some image processing testing approaches. The webpage[2] reviews ten AI based image generators.

Fajar Suryawan has recently presented an FPGA based geometric pattern test image generator for VGA displays[36].

ZIPcores markets IP core for video test pattern generation[41]. IP cores for test image generators are also provided by FPGA vendors, like Xilinx[40].

Our group has been one of the pioneers of using evolutionary computing in the context of software testing, including test image generation[3;4;5;6;7;20;21;22;23]. However, in this work due to simplicity and software restrictions caused by the embedding hardware, we are not using evolutionary computing here, but in some other topics of our project reported elsewhere.

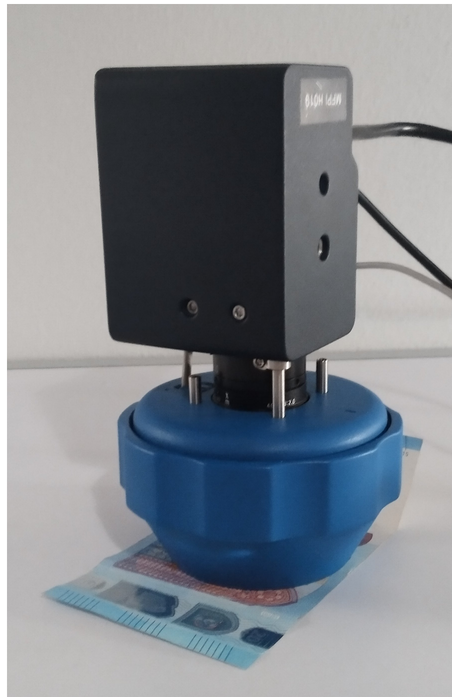


Figure 2: The multispectral camera imaging a 20€ note (photo by Dániel Tisza)

2 Camera

The camera is Basler daA2500-14uc with CS-Mount, but the sensor is from another maker and you can find more information on the sensor from the datasheet [28]. The typical spectral characteristics of our spectral camera sensor is shown in Figure 4. For more information on the camera see e.g. [9;25;26;30;31;32;33;34;37;38;39].

2.1 Interference filter

The spectra are filtered by a mirror coated cavity, called etalon, the length of which is electrically controlled to select wavelengths. This is the so called Fabry-Pérot interferometer, the principle of operation of which is shown in Figure 5. It is a simple but very precise instrument to filter wavelengths i.e the spectrum of light. Unfortunately its raw output is somewhat complex to interpret, but luckily we can write programs that can transform the optical information of the device into a more useful one. The details of that procedure are described elsewhere.

In Figure 5 light comes from left and goes thru two partially reflecting mirrors, of which the left one is fixed and the right one is moved by a control voltage. The device is quite small MicroElectroMechanical System (MEMS), which can be used in small cameras

Table 1: Properties of the camera.

Make	Basler
model	daA2500-14uc
type	RGB / CMOS
sensor	Mouser MT9P031
image size	2592×1944 pixels
bits/pixel	8 or 12
max speed	14 fps
interface	USB 3.0
image size	5.7×4.3 mm^2
pixel size	2.2×2.2 μm^2
power (typical)	1.3 W
weight (typical)	15 g
lens mount	CS-mount
operating temperature	0-50 °C

Figure 3: The camera body daA2500-14uc (source: Basler AG)

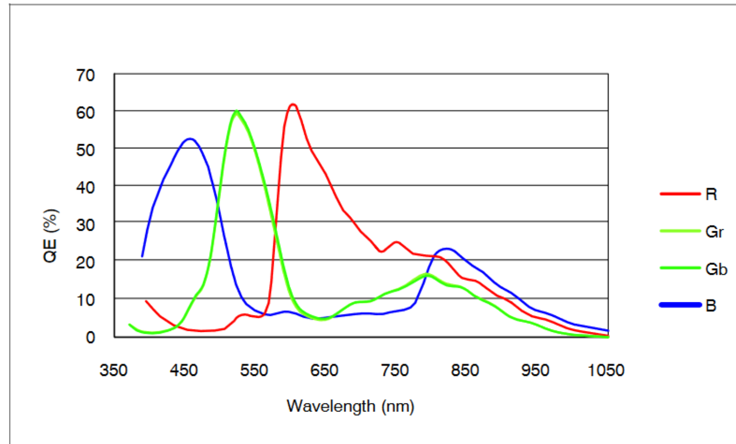


Figure 4: Camera sensor sensitivities of each RGB channel pixels (Bayer). Notations: Gr = Green pixel on Red pixel row, Gb = Green pixel on Blue pixelrow. (source: `https://www.mouser.fi/datasheet/2/308/MT9P031-D-1103275.pdf`)

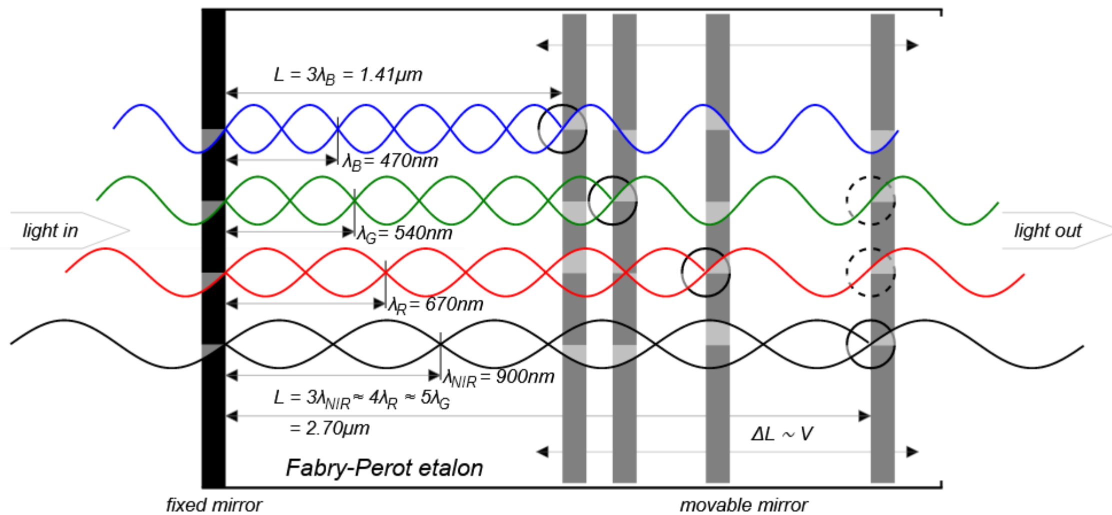


Figure 5: The principle of Fabry-Pérot interferometer. For each wavelength λ_c the length of the etalon is set $L = n(\lambda/2) = 6(\lambda/2) = 3\lambda_c$, i.e. $n = 6$, $c = R, G, B, NIR$.

(e.g. satellites, drones, mobile) and fast; the mass of the moving mirror is really small allowing rapid accelerations.

When the length of the cavity L is an integer multiple of $\lambda/2$, the beam of light is going thru the interferometer at highest intensity and when $L = \lambda(n+1)/2$, then the transmission T_e of the etalon is at its minimum. The value of transmission depends on several parameters: the reflectivity of the mirrors R, wavelength λ, length of the etalon L, and the angle of the beam w.r.t the normal of the mirrors Θ:

$$T_e(R, L, \lambda, \Theta) = \frac{1}{1 + F(R)\sin^2(\sigma(L, \lambda)/2)},$$

where $F(R) = 4R/(1-R)^2$ is called the coefficient of finesse, $\sigma = (2\pi/\lambda)2nL\cos\Theta$ is the phase difference of each successive transmitted beams, and Θ is the angle between the beam and the normal of the mirrors. In the center of the image $\Theta = 0$, so that $\sigma = 4n\pi L/\lambda$. An example of transmission curves calculated by the above equation for several finesse values ($F = 10, 30, 50, 70, 90$) is shown in Figure 6. The higher the finesse value the sharper the transmission peak and the lower the leaking between the peaks, i.e. in the stop band. Observe, that the curves are shown as the function of wavelength. However, in spectroscopy the rule is to use **wavenumber** $k = 1/\lambda$ because its properties are often more handy for processing the spectral data as shown in Figure 7, where it gives equal distances between the transmission peaks. Observe also, that often the unit used for k is not 1/m but 1/cm.

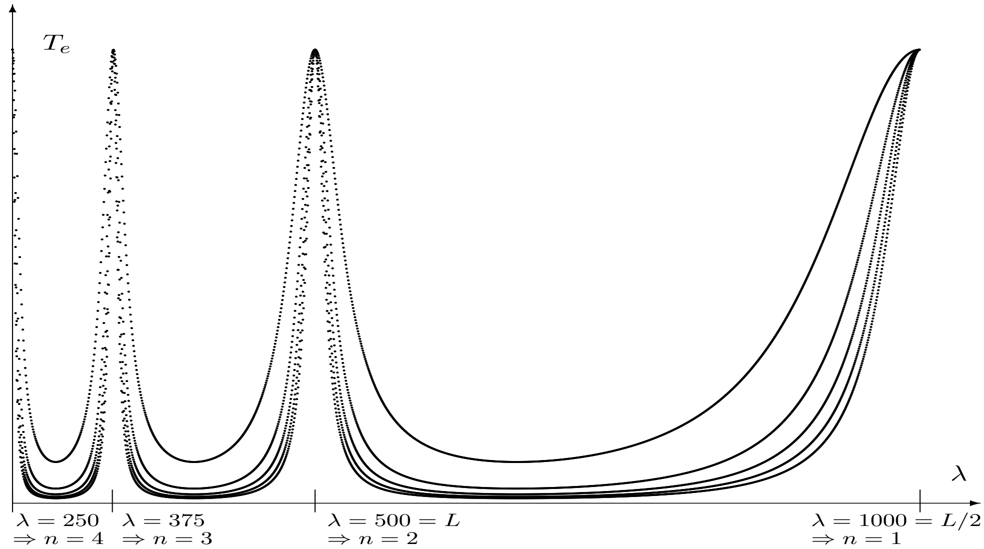


Figure 6: The transmission of an ideal etalon of length $L = 500$ for different finesses $F = 10, 30, 50, ...$ as the function of wavelength. The higher the fitness the sharper the peak and the lower the stop band.

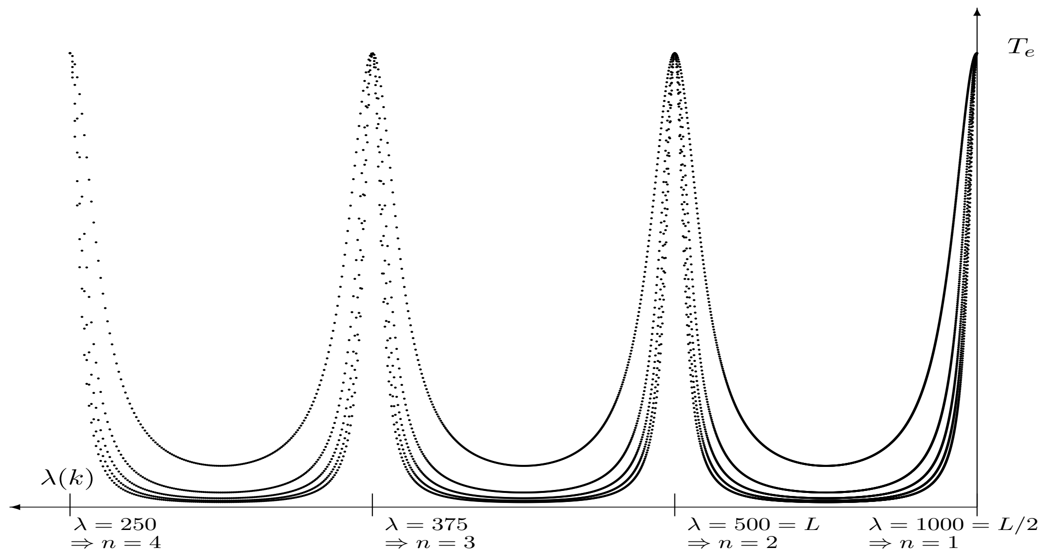


Figure 7: The transmission of an ideal etalon of length $L = 500$ for different finesses $F = 10, 30, 50, ...$ as the function of **wavenumber** $k = 1/\lambda$.

3 `ARMmovie.c`, a test image generator

The C programming language[17] was chosen in order to be able to run the generator in the embedded environment of SoC-FPGA, where there is not so much language alternatives in practice mainly due to the operating system restrictions. A C language program can be run on practically any main frame computer and most PCs and microcontrollers with or without an operating system. Here we are running it in three main computing environments: on a university server computer where most of our programs, documents, and data are saved, on personal and shared PCs, and finally on the ARM processor in the SoC-FPGAs.

The name `ARMmovie` comes from the ARM processor family and the fact that the program is able to generate image sequences i.e. video or movie. In practice the image sequence can be converted into a gif format image animation by Unix/Linux `convert` command. It is also possible to call a C language program from e.g. a Java -language program. A demonstration software where a Java -language program calls a C -language function has been implemented for Guided Image Filter implementation reported elsewhere.

3.1 Control

`ARMmovie`'s control input consists of a set of parameters that control the generation and an optional set of images used as background of the generated test images. The idea has been that the small program can be used in testing of the camera system also in cases when

there is no image source present or when doing a selftest for the system. `ARMmovie.c` is able to produce test images based entirely on the parameters input to it. It is also able to generate image sets to be joined into `.gif` animation files by external programs like the Unix/Linux command line command `convert`.

The simulator is written in C so that it may be run even without operating system in a small FPGA-SoC having ARM processor. That is why file operations are limited to minimum and so that they can be turned off when run on the object device, on bare metal. E.g. the camera calibration tables are not read from files but are included as a part of the program as an `.h` file. However, the program can be run on UNIX/Linux operating systems e.g. to produce data for documentation purposes, like this report. In addition there are several simple AWK-language[1] programs that are used to process the calibration files and/or graphics also for documentation.

The generated image can contain the following features:

- color gradients of size (256 × 256) that are regularly tiled within larger images (see Fig. 8), the test image size is controllable by the user,
- an animated magnifier glass for showing details in controlled locations, and
- optionally a background image input as a `.ppm` file.

The output is a one dimensional array containing one generated test image. In the OS-mode the program is able to save the images as proper image files in a given directory.

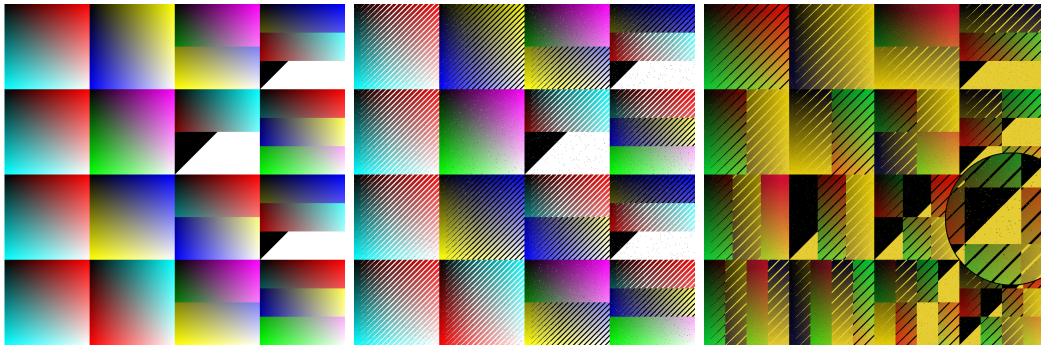


Figure 8: Color gradient tiling example (left), Stripes and salt-and-pepper noise on gradients (middle), and animated magnifier glass and color filtering (right). Observe, that the right image has a different tiling than the left and middle ones. The animation of the right most image can be seen at `http://lipas.uwasa.fi/~TAU/ICAT1040/slides.php?File=5100TestImage.txt&Page=18`

3.2 Simulation of the camera

As told above the spectral camera uses an interference filter to select certain wavelengths. We can roughly simulate that by adding the fourth step to the image generation algorithm:

multiplication of each channel (except alpha) by a given constant (from table RGBw[]):

```
// Step 4: filtering of channels (VTT MEMS filter):
if (RGBw!=NULL) C = (char) (*(RGBw+c)*C); // camera filter
```

The rightmost image of Figure 8 has been filter by multiplying with

$$\texttt{RGBw} = [0.9, 0.8, 0.1].$$

As can be seen blue color has almost disappeared because it is multiplied by 0.1, while red and green are only slightly attenuated. A more thorough presentation of the camera system simulation is given later in this report.

3.3 Execution profile

To see how much time is used in the main functions of `ARMmovie.c` we took the execution time profile shown in Table 2. As can be seen, most of the time is used in generating the testimage (`pgmOutput` and `generateVideoFrameFaster`) just as anticipated. Outputting .ppm (binary) instead of .pgm (ascii text) image files will reduce the output time about to half and file size to one quarter saving about 75% energy use in file saving.

The next most time is used by `Magnifier` and `interpolate`. That are the functions making the image within the magnifying glass. The `interpolate` function is very simple, only one line:

```
return t*A + (1-t)*B;
```

How this oneliner can consume so much time? It must be the two multiplications. Unfortunately multiplication is quite essential in linear interpolation. Actually the magnifier glass calls this function twice: for x and y directions. Hence it seems that the four multiplications per pixel consume quite much time, and energy. For a more efficient interpolation algorithm on FPGA see e.g. [19].

After several small refactorings and other modifications, such as reducing the area of the magnifier glass processing and random noise generation, the speed of the test image generation was over 20 frames per second on the server.

3.4 Parameter input

`ARMmovie.c` is controlled by giving it parameters in a string or in a parameter file, which both contains one command per line starting with one letter command followed by the parameters of the command. In Figure 9 is is an example of such a file or string. Observe, that the line after `#` is a comment. The one letter commands are the following:

- `S`: Size of the generated image,
- `F`: File (output) base name,

Table 2: An example of the execution time profile of `ARMmovie.c`.

$\%t$	$\sum t$(s)	t(s)	*calls*	ms/call	function
33.37	0.31	0.31	21	14.78	`pgmOutput`
29.06	0.58	0.27	21	12.87	`generateVideoFrameFaster`
15.07	0.72	0.14	6666516	0.00	`Magnifyer`
8.61	0.80	0.08	11990010	0.00	`interpolate`
8.61	0.88	0.08	8388608	0.00	`generateRGBcolorPattern`
4.31	0.92	0.04	2	20.02	`generateTestImage`
1.08	0.93	0.01	1	10.01	`copyImage`
0.00	0.93	0.00	6210324	0.00	`min`
0.00	0.93	0.00	20056	0.00	`nextTime`
0.00	0.93	0.00	21	0.00	`intToStr`
0.00	0.93	0.00	20	0.00	`videoFrameToDMA`
0.00	0.93	0.00	2	0.00	`memoryAllocation`
0.00	0.93	0.00	1	0.00	`showMovie`

- `I`: Image file name for the background,
- `W`: Weights of camera channels,
- `B`: Background pattern feature parameters: S, Stripe parameter, and Salt&Pepper noise parameter,
- `L`: Lens (magnifier) parameters: Radius, ΔRadius, X0, Y0, ITC, and Magnification,
- `V`: Video parameters: FrameNumber, FrameNumber max, and Δtime between frames in animation,
- `X`: X-coordinate list for the magnifier glass animation, and
- `Y`: Y-coordinate list for the magnifier glass animation.

The last two parameters (vectors) are used for moving the animated magnifier glass lens over the image. To see the 'movie' or animation, convert the .ppm files generated into a `.gif` file e.g. by the versatile `convert` aka ImageMagick^{TM} UNIX command:

```
convert -delay 200 -loop 0 MunRaami*.ppm myVideo.gif
```

where `delay` option gives the time delay between each animation frame and `loop 0` starts animation repeatedly at the first frame (index=0).

3.5 Image comparison and animation

The magnifier glass can be used in image comparison. You know the traditional set up where two nearly similar images are shown side by side for comparison. Here we can compare images via an animated magnifier glass (Fig. 10), which is halved into two semicircles,

```
# image size:
S 2048 1080
# frame file name
F MunRaami
# background image name (.ppm)
I tmp2
# camera channel weights
W 400 0.9 0.8 0.2
# background image parameters S, Stripes, SaltPepper
B 2 -64 -10000
# magnifyer glass parameters: R, dR, X0, Y0, ITC, M
L 200 20 0 0 50 2.0
# video, frame, and time difference: FN, FNmax, dt
V 1 10 10
# Lens center coordinates:
X 500 500 500 500 500 500 700 900 1100 1300 1500 1700 1700 1700 1700 1700 1700 1700 1700
Y 300 400 500 600 700 800 800 700 600 500 400 300 300 400 500 600 700 800 900
```

Figure 9: An example parameter file for `ARMmovie.c`

the upper half showing the input image while the lower part shows the processed image. Giving proper coordinates of points of interest to the magnifier (`X` and `Y` parameter vectors), we can see these points magnified in the animation within one image in one glance without the need to move eyes between the two compared images.

Also spectral imaging can be animated. Figure 11 shows 8 frames of the animation, at wavelengths $\lambda = [450, 500, 550, \ldots, 800]$nm. For an animation of spectral camera imaging test image Red Admiral see `https://lipas.uwasa.fi/~TAU/ICAT1040/Slides/video/myVideoVTTamiral.gif`. More testimages at: `https://lipas.uwasa.fi/~TAU/gallery/test/`.

3.6 Camera simulation

The sensitivity graphs of the sensor RGB-pixels are shown in Fig. 4. Unfortunately the datasheet has no further information of the sensitivities than that figure. How to obtain the numerical values of the graphs, that is described next.

We must somehow scan the image and extract the points. AUTOpoint, the author's lecture slide browser, has the option to draw with mouse. The original sensitivity curves has been drawn on the figure shown in an AUTOpoint slide window by mouse as shown in Fig. 13. The mouse points are written in real time to browser console using the given mouse color as background so that the data is easily found within the other text in console for copy-pasting.

Figure 12 shows the result over drawn by mouse on the original curves of the datasheet. The mouse points are the coordinates of the image on display window. They have three main issues that must be solved to get useful data:

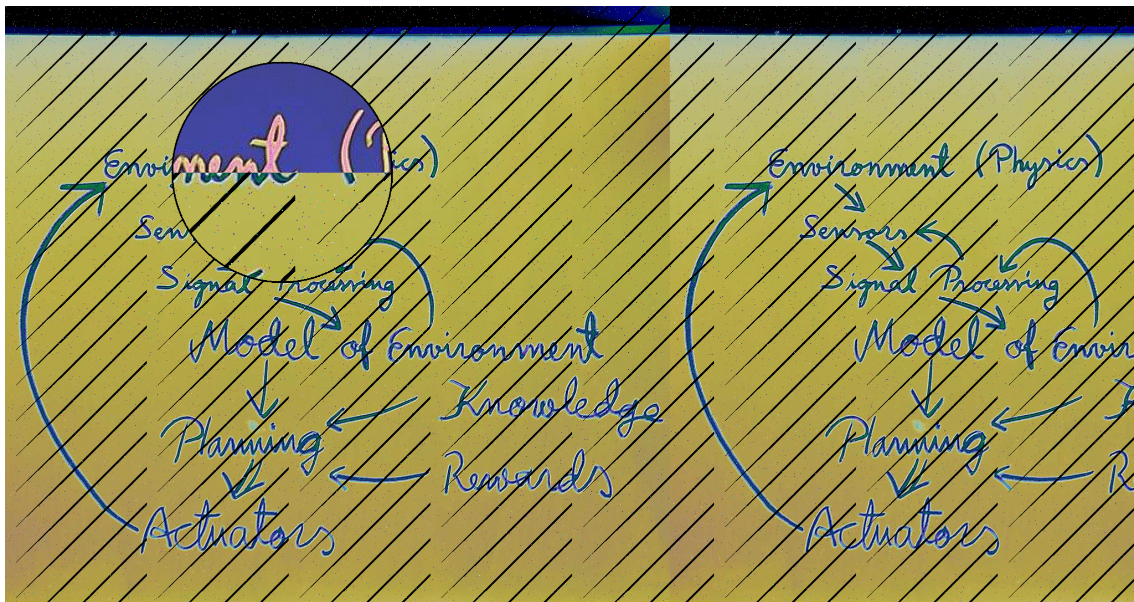


Figure 10: Comparing two images thru Magnifier glass: upper semicircle the original image and the lower semicircle the generated image. The animation can be seen at `http://lipas.uwasa.fi/~TAU/ICAT1040/slides.php?File=5100TestImage.txt&Page=44`

- the data points are irregularly spaced (manual mouse movement),
- the data points are more or less erroneous i.e noisy, and
- the data is in image coordinates of Windows not in the original graph wavelength × percent coordinates.

These issues were solved by a few simple `.awk` programs that cleaned the data, made averaging i.e. filtering, and finally manually fixing some extra/missing points. The last issue of mapping the display window to the real data was solved by drawing the original image on which the drawn data was plotted on using the data coordinates (Fig. 14-16). After a few iterations the match seemed good enough for the simulation experiments here and for testing. Anyway, remember, that for the final production use each camera sensor is unique and careful calibration must be done in order to get the best possible imaging precision. Finally linear interpolation was used to get a point for each integer value of wavelength in the range [400,1050] nm.

The numerical data was further transformed to a C language table, which is further used in `ARMmovie.c` to produce test image sequences for the image processing pipeline.

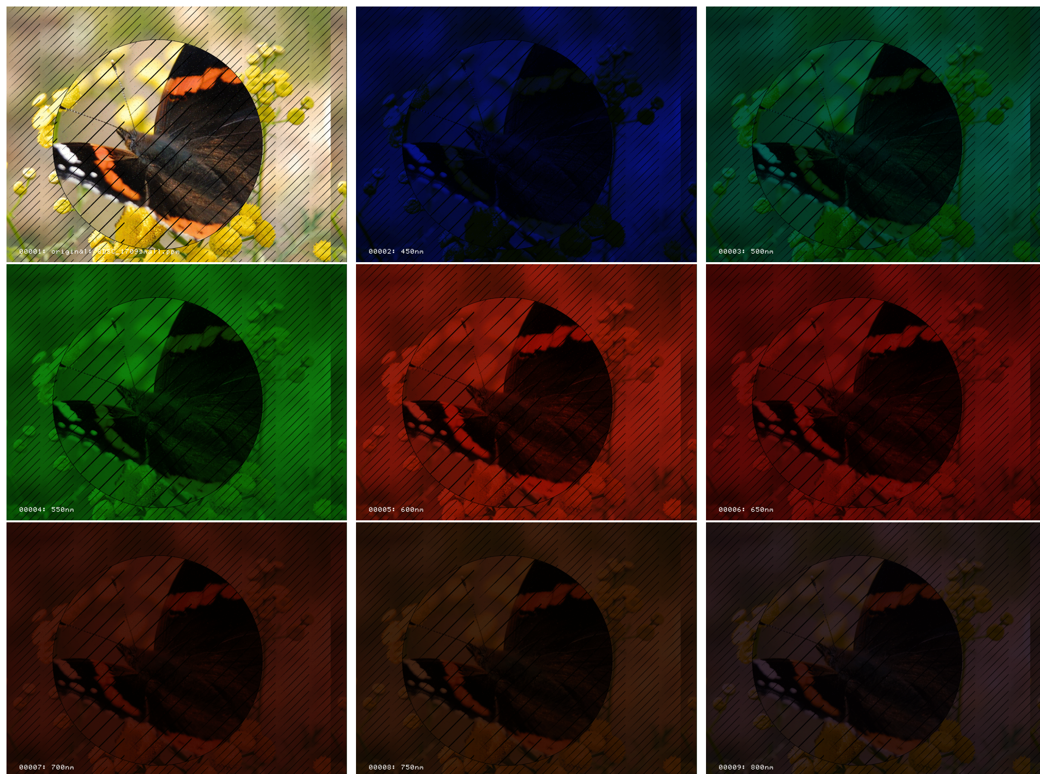


Figure 11: Upper left: the original image Red Admiral on tansy, images at simulated wavelengths 450, 500, 550, 600, 650, 700, 750, and 800 nm. Observe the decreasing intensity due to the decrease of the sensor sensitivity towards both ends of the spectra. The .gif animation can be seen at `http://lipas.uwasa.fi/~TAU/ICAT1040/slides.php?File=5100TestImage.txt&Page=52`

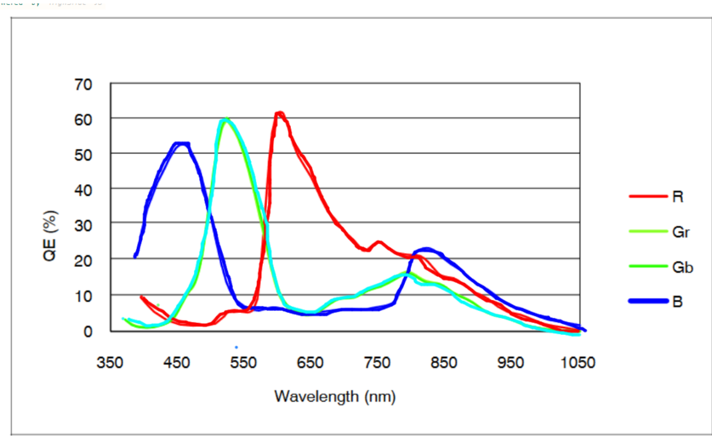


Figure 12: Sensor sensitivity curves drawn on the original image using `AUTOpoint`. The colours for drawing has been chosen to be more or less like the underlying curve colors. (background image source: `https://www.mouser.fi/datasheet/2/308/MT9P031-D-1103275.pdf`)

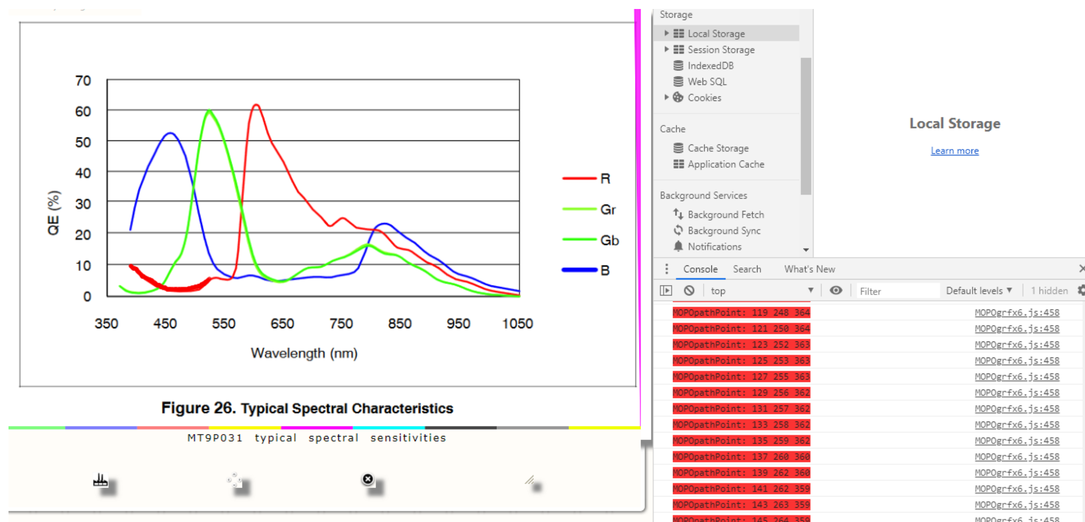


Figure 13: Sensor sensitivity curves drawn on using AUTOpoint and saving the points to browser (here Chrome) console. The colours for drawing the curves are also used as the background colors of the corresponding console printing. (background image source: `https://www.mouser.fi/datasheet/2/308/MT9P031-D-1103275.pdf`)

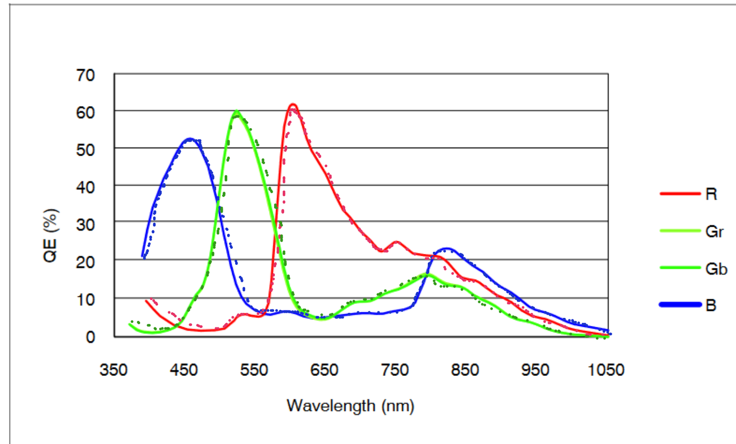


Figure 14: Fitting the original draw data on the original image. (background image source: `https://www.mouser.fi/datasheet/2/308/MT9P031-D-1103275.pdf`)

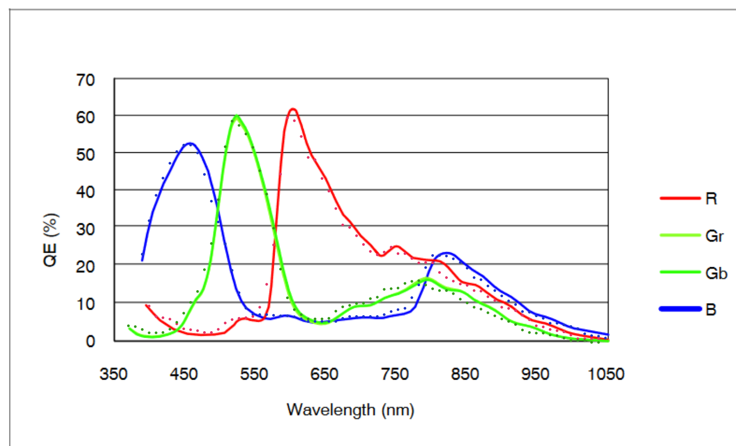


Figure 15: Fitting the averaged draw data on original image. (background image source: `https://www.mouser.fi/datasheet/2/308/MT9P031-D-1103275.pdf`)

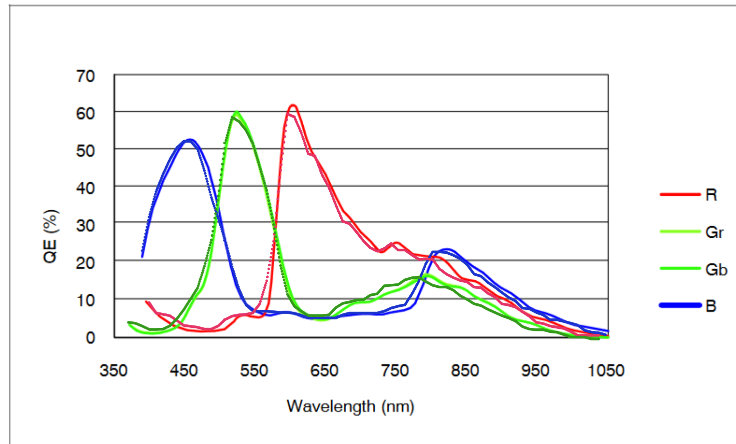


Figure 16: Fitting the final filtered and interpolated draw data on the original image. (background image source: `https://www.mouser.fi/datasheet/2/308/MT9P031-D-1103275.pdf`)

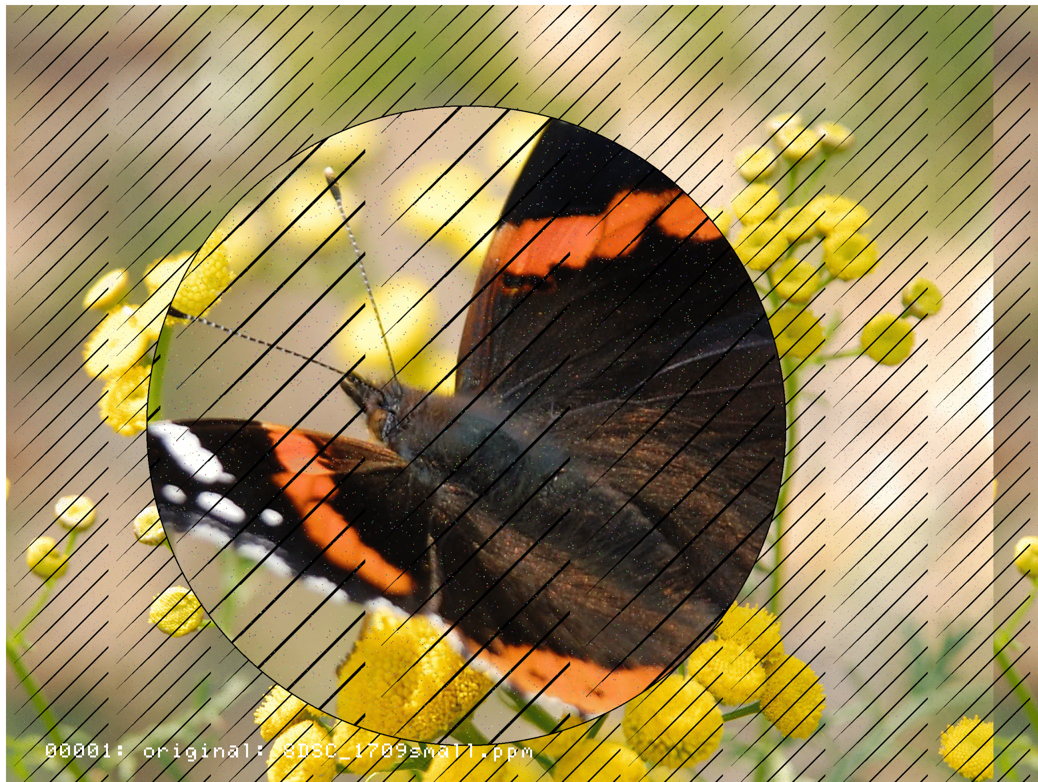


Figure 17: Test image (frame 0: the original image of a red admiral *Vanessa atalanta* (L. 1758) feeding on tansy *Tanacetum vulgare* (L.).

Figure 18: Test image (frame 6: 650nm filter setting).

3.7 LED simulation

An essential part of any imaging system is the illumination. In our case the illumination for the camera is done using LEDs. There are triplets of seven spectrally different LEDs having relatively narrow wavelength bands. Each LED of the triplet can be individually controlled giving the possibility to illuminate the object from three different fixed directions. The LED spectra can be modeled using Gaussian curve $e^{-(\lambda-\lambda_0)^2/\Delta\lambda^2}$, where λ is the wavelength, λ_0 is the wavelength of the intensity maximum, and $\Delta\lambda$ is the width of the band. An example of LED light spectrum is shown in Figure 21. Having more than one Gaussians as a **weighted sum**, we can broaden the band boundaries and modify the shape of the spectrum to better conform the true LED spectra (Figure 22 HLMP-3750/-3390/-1340 Series Lamps). The parameters of the Gaussian sum is given in Table 3.

Table 3: Parameters of the model of high efficiency red LED (Fig. 22). Observe, that due to imperfect (manual) scaling of the images, the sum of the weights is 113%.

λ_0	$\Delta\lambda$	*weight [%]*
630	27	75
655	53	24
663	23	10
597	29	4

The calculations were done using `LED.awk` program that is a small AWK language program. AWK is a scripting language intended for simple calculations and text processing[1]. It is one of the basic tools of UNIX/Linux operating system and can be used from command line as well alone as within pipes. AWK programs consists typically of three parts: begin block, input block, and end block. Begin block starts with keyword `BEGIN` and is followed by a block within braces. It is the natural place for setting default values and other initialisations. The most useful block is the input block which **automatically** reads records, one at a time and also extracts fields withing the records. This makes scanning fixed format text easy with a handful of automatically set variables, like $0 = whole record, $1 = first field of record, `NF` = number of fields, `NR` = number of records, etc. The syntax of AWK is similar to C (C++, Java, JavaScript, ...) so that it is easy to learn once you know one of those more well known programming languages.

The last block is the `END` block, that is a good place for ending the task. Here it is used to output the graph as a LaTeX `picture` environment commands that plot the graph on this report. In addition the program outputs the LED efficiency curve as a C language table for each nanometer as lines starting with '%@' so that the data can be extracted and further included in the camera system simulator program `ARMmovie.c` as a part of the file `LEDs.h`.

Our LED model of using Gaussians is actually similar to the more general Artificial Intelligence (AI) concept called Radial Basis Function (RBF) neural network, which are based

on weighted Gaussians in n dimensions[12].

3.7.1 Camera with LEDs simulation

Now we have both the model of the camera sensor and the LED used as its illuminator. Combining their functions can be done using the dot product:

$$I_c(i,j) = p(i,j) \sum_{\lambda=\lambda_0}^{\lambda_{\max}} I_{\text{LED},\lambda} \text{QE}_{c,\lambda} = p(i,j) I_{\text{LED}} \cdot \text{QE}_c,$$

where $I_c(i,j)$ is the recorded intensity of pixel at (i,j) on channel c (= R, G, or B), $p(i,j) \in [0,1]$ is the reflectivity of the imaged object that covers the pixel (i,j), $\lambda \in [\lambda_0, \lambda_{\max}]$ is the wavelength, I_{LED} is the intensity profile of the LED, and QE_c is the sensitivity (Quantum Efficiency) of the camera sensor for channel c. For the continuous case the function can be represented by an integral over the wavelength range:

$$I_c(i,j) = p(i,j) \int_{\lambda=\lambda_0}^{\lambda_{\max}} I_{\text{LED}}(\lambda) \text{QE}_c(\lambda) d\lambda.$$

Observe, that for practical (numerical) calculations we must further define the physical units, like what is the power (width) of each wavelength interval in the summing.

Observe also, that the vectors (graphs) for the dot product $I_{\text{LED}} \cdot \text{QE}_c$ should be of equal length. If they are not then we must use e.g. linear interpolation to make them be of equal length.

3.8 RGB sensitivity analysis

Let us look a bit closer the sensitivity curves. Figure (19) shows the sensitivity curves together with the sum of the sensitivities. In the long wavelength region the sum curve seems to be approximately piecewise linear:

$$C(\lambda) \approx \begin{cases} (\lambda - 785)/200 + 2 & \text{if } \lambda \in [730, 815] \\ 3(835 - \lambda)/400 + 2 & \text{if } \lambda \in [815, 980] \\ \text{undefined} & \text{otherwise} \end{cases}$$

i.e.

$$\text{QE}_{\text{NIR}} \propto C(\lambda) \propto (\text{QE}_{\text{R}} + \text{QE}_{\text{G}} + \text{QE}_{\text{B}}).$$

It means that we can measure the incoming power of the light by using a simple relationship between the pixel values read from the camera and $C(\lambda)$. We can e.g. use the camera to measure optical energy in the NIR band (I_{NIR}). Remember e.g. that half of the sun energy is in the near-infrared band ($\lambda \gtrsim 800$nm). However, there are more simple ways to measure solar energy. Our camera is designed for more detailed analysis of objects and their properties, like the condition of diabetic legs and such medical applications like caries. Figure 20 shows an example of simulating the imaging of a leg having an ulcer.

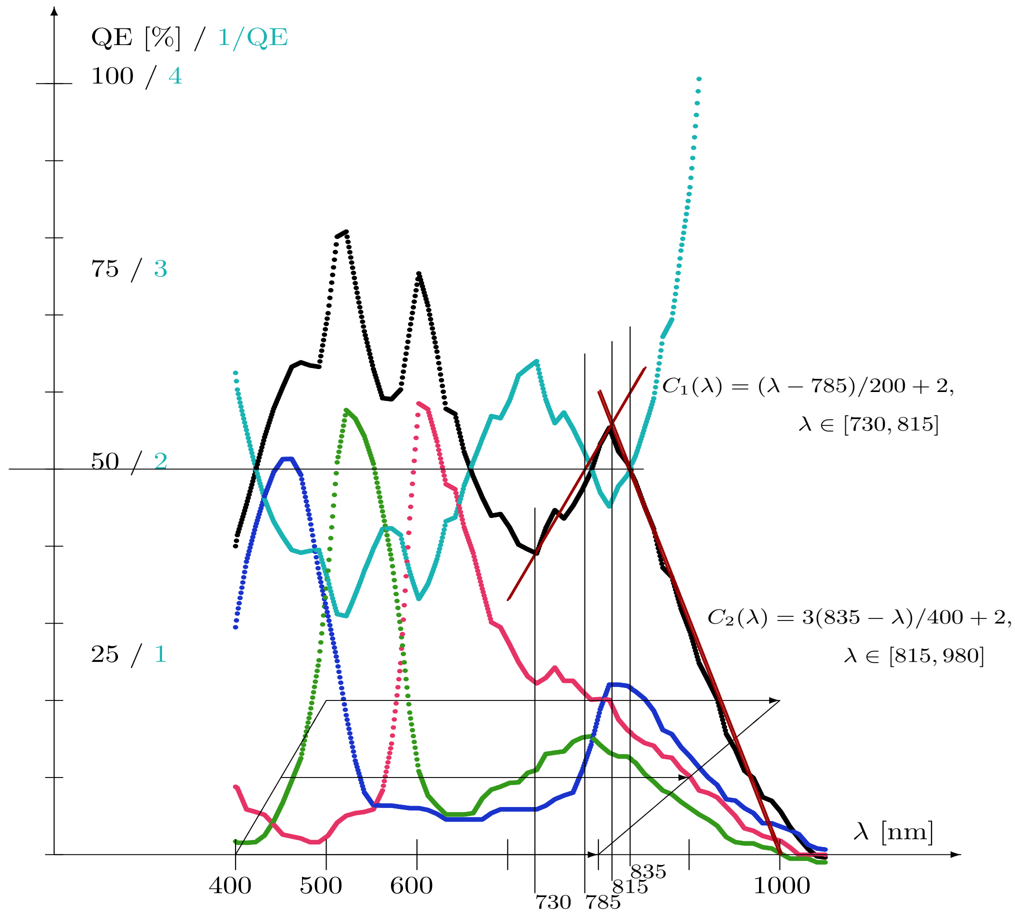


Figure 19: Quantum efficiencies QE_c, $c \in \mathrm{R}, \mathrm{G}, \mathrm{B}$ of color channels (red $= \mathrm{QE}_\mathrm{R}$, green $= \mathrm{QE}_\mathrm{G}$, and blue $= \mathrm{QE}_\mathrm{B}$) of the camera and the sum of channel quantum efficiencies (black curve) and its inverse (cyan curve) as function of wavelength λ. Linear regression lines $C_1(\lambda)$ and $C_2(\lambda)$ in the NIR area shown by dark red lines.

Figure 20: Test image leg (frame 17: 800nm filter setting).

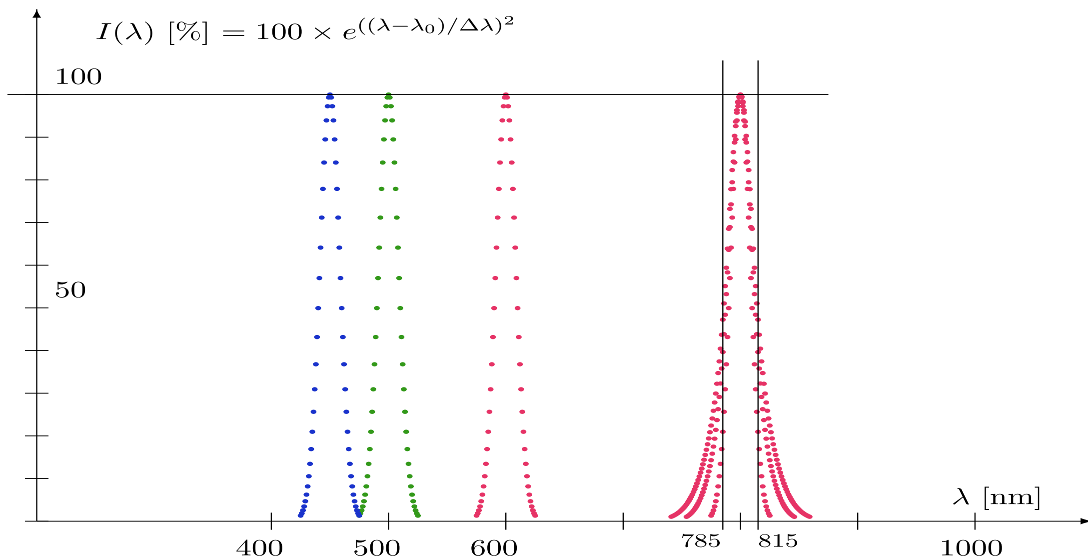


Figure 21: LED spectra simulation for $\lambda_0 = 450$, 500, 600, and 800nm and $\Delta\lambda = 12$nm. The 800nm LED has 1, 2, and 3 Gaussians having common maximum wavelength.

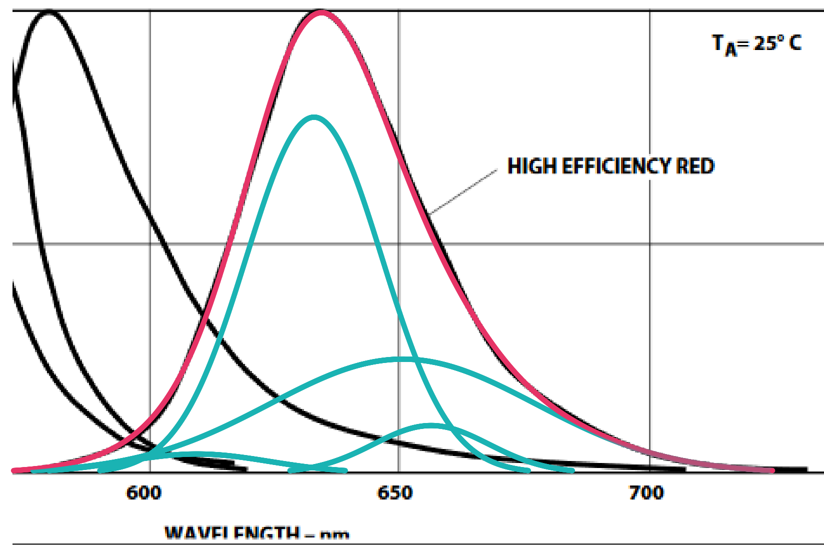


Figure 22: LED spectra simulation for HLMP-3750/-3390/-1340 Series LED Lamps (Mouser / Avago) $\lambda_0 \approx 630$nm (high efficiency red), $\Delta\lambda \approx 45$nm. The model has 3 Gaussians shown by cyan while the total spectra is shown by a red curve that mostly hide the datasheet curve shown in black. Graphs plotted for comparison on the datasheet image[24]. The parameters of the Gaussians are shown in Table 3.

4 Running ARMmovie.c

The current version of the generator is 2, which is shown in the file name, `ARMmovie2.c`. The program `ARMmovie2.c` was run on a SoC development board in which the CPU and the FPGA are implemented on the same chip. The board we used was Genesys ZU-3EG by Digilent Inc.[13] This board also has an external mSATA memory chip (512 GB) attached to it. The test image program is used in a Linux environment which is installed on the Genesys board. Previously this program has been used succesfully with Digilent's Zybo Z7-20 SoC board.[42]

4.1 Running procedure

The testing was done by using the following procedure:

1. The Genesys kit was connected to the laptop (Lenovo Thinkpad T61) and started by using PuTTy terminal application. The ssh connection between the laptop and the board was established.
2. The program and image files were copied to the folder `/home/ubuntu`.
3. From this folder the files were copied to the Genesys kit by using the `ssh` connection.

4. The test image program was compiled by: `gcc -pg -o myRun ARMmovie2.c -lm`.
5. The program was executed by: `./myRun`.
6. The test images created by the program were copied to the folder `/home/ubuntu` by using the `ssh` connection.

Using the test image program on the Genesys kit created successfully all of the required test images. The larger processing capacity and memory allocation of the Genesys kit make it more efficient in using the program compared to the Zybo Z7-20 board. This test indicates that the Genesys kit is suitable in creating test images for image processing pipelines.

5 Conclusions and Future

In this report we have presented a C language program called `ARMmovie.c`, which generates either purely synthetic images or combines a synthetic image with given images (user given `.ppm` files) for various embedded camera testing purposes. It was originally designed to be compact and stand-alone in order to be useful for testing a spectral camera hardware based on heavily memory limited FPGA-SoC technology but it is not limited to that application even if it contains several functions that simulate the special spectral image recording, including illumination by LEDs, of the object system. These features include the selection of wavelength bands recorded by the camera and illumination of the object by simulated narrow band LEDs. The program can be run in a stand-alone mode on an embedded ARM processor on a FPGA-SoC chip or on a PC or similar more conventional computer. The program functions and outputs are controlled by a few parameters that can be input as a string (hard coding) or read from a file when run under an operating system. Some attempts to optimize the processing was also done with the result of considerable speed-up. The output consists of an image pixel stream that can be input to the FPGA or external files that can be further combined e.g. into a gif animation.

5.1 Future

Originally the generator was designed to be run under a Java language program with Java JNI. The Java program could also handle the graphical user interface (GUI) of the device. However, the current GUI is done using Python. The generated patterns could be optimised based on the object systems functions using e.g. evolutionary optimisation[21]. For selftesting purposes it could be possible to implement the generator at least partly on the FPGA. Especially the synthetic pattern is such that it could be implemented on a HDL, while the real image functions are much better suited for a processor. The generator could also compare the generated and processed images, which would greatly support the realisation of selftesting.

References

[1] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. *The AWK Programming Language*. Addison-Wesley Publishing Company, Reading, MA, 1988.

[2] Top 10 AI Image Generators Review. `https://topten.ai/image-generator-review/`.

[3] Jarmo T. Alander, Maarit Harju, and Petri O. Välisuo. Optimal testing of embedded software using genetic algorithm — a prestudy. In Jarmo T. Alander, editor, *Proceedings of the Third Nordic Workshop on Genetic Algorithms and their Applications (3NWGA)*, pages 299–304, Helsinki (Finland), 18.-22. August. Finnish Artificial Intelligence Society (FAIS).

[4] Jarmo T. Alander and Timo Mantere. Ohjelmistojen testausta geneettisten algoritmien avulla [Testing programs using genetic algorithms]. In Matti Linna, editor, *Tekniikan koulutusta 10 vuotta Vaasan yliopistossa [10 Years of Technology in the University of Vaasa]*, pages 35–38. 1998. (in Finnish).

[5] Jarmo T. Alander and Timo Mantere. Automatic software testing by genetic algorithm optimization, a case study. In Conor Ryan and J. Buckley, editors, *Proceedings of the 1st International Workshop on Soft Computing Applied to Software Engineering*, pages 1–9, Limerick, Ireland, 12.-14. April 1999. Limerick University Press.

[6] Jarmo T. Alander, Timo Mantere, and Pekka Turunen. Genetic algorithm based software testing. In George D. Smith, Nigel C. Steele, and Rudolf F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms, Proceedings of International Conference (ICANNGA97)*, pages 325–328, Norwich (UK), April 1997. Springer-Verlag, Wien.

[7] Jarmo T. Alander, Timo Mantere, Pekka Turunen, and Jari Virolainen. GA in program testing. In Jarmo T. Alander, editor, *Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications (2NWGA)*, Proceedings of the University of Vaasa, Nro. 11, pages 205–210, Vaasa (Finland), 19.-23. August. University of Vaasa.

[8] Anders S. G. Andrae and Tomas Edler. On global electricity usage of communication technology: Trends to 2030. *Challenges*, 6(1), 2015.

[9] L. Annala, N. Neittaanmäki, J. Paoli, O. Zaar, and I. Pölönen. Generating hyperspectral skin cancer imagery using generative adversarial neural network. In *42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 1600–1603, Montreal, QC, Canada, 2020.

[10] Kent Beck. *Test-Driven Development by Example*. Vaseem: Addison-Wesley, 2002.

[11] Saman Biookaghazadeh, Ming Zhao, and Fengbo Ren. Are FPGAs suitable for edge computing? In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, Boston, MA, July 2018. USENIX Association.

[12] D. S. Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Memorandum 4148, Royal Signal and Radar Establishment, 1988.

[13] Digilent's Genesys ZU-3EG SoC development board. `https://www.xilinx.com/products/boards-and-kits/1-18dv0nz.html`, (Accessed 13.5.2022).

[14] Stanislav Frolov, Tobias Hinz, Federico Raue, Jörn Hees, and Andreas Dengel. Adversarial text-to-image synthesis: A review. *Neural Networks*, 144:187–209, 2021.

[15] Nicola Jones. How to stop data centres from gobbling up the world's electricity. *Nature*, 561:163–166, 2018.

[16] James W. Kalat. *Biological Psychology.* Wadsworth Thomson Learning, Belmont, CA, 2001.

[17] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language.* Addison-Wesley Publishing Company, Reading, MA, 1 edition, 1978.

[18] Woo-Tae Kim, Cheonwi Park, Hyunkeun Lee, Ilseop Lee, and Byung-Geun Lee. A high full well capacity CMOS image sensor for space applications. *Sensors*, 19:–, 2019.

[19] Janne Tapio Koljonen, Jarmo Tapani Alander, Vladimir Bochko, and Sami Juhani Lauronen. Fast fixed-point bicubic interpolation algorithm on FPGA. In Jari Nurmi, Peeter Ellervee, Kari Halonen, and Juha Röning, editors, *Proceedings of The 2019 IEEE Nordic Circuits ans Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, pages 1–7, Helsinki (Finland), 29.-30. October 2019. IEEE.

[20] Timo Mantere. *Automatic Software Testing by Genetic Algorithms.* PhD thesis, University of Vaasa, Department of Electrical Engineering and Production Economics, 2003.

[21] Timo Mantere and Jarmo T. Alander. Automatic test image generation by genetic algorithms for testing halftoning methods. In David P. Casasent, editor, *Intelligent Systems and Advanced Manufacturing: Intelligent Robots and Computer Vision XIX: Algorithms, Techniques, and Active Vision*, volume SPIE-4197, pages 297–308, Boston, MA, 5. -8. November 2000. The International Society for Optical Engineering, Bellingham, WA.

[22] Timo Mantere and Jarmo T. Alander. Automatic software testing by optimization with genetic algorithms, introduction to the method and considerations of the possible pitfalls. In Radek Matoušek and Pavel Ošmera, editors, *Proceedings of the 7th International Mendel Conference on Soft Computing (MENDEL 2001)*, pages 19–23, Brno (Czech Republic), 6.-8. June 2001. Technical University of Brno.

[23] Timo Mantere and Jarmo T. Alander. Testing a structural light vision software by genetic algorithms—estimating the worst case behavior of volume measurement. In David P. Casasent and Ernest L. Hall, editors, *Intelligent Robots and Computer*

Vision XX: Algorithms, Techniques, and Active Vision, volume SPIE-4572, pages 466–475, Boston, MA, 29. -31. October 2001. The International Society for Optical Engineering, Bellingham, WA.

[24] HLMP-3707, HLMP-3907, HLMP-3750, HLMP-3850, HLMP-3950, HLMP-3960, HLMP-3390, HLMP-3490, HLMP-3590, HLMP-1340, HLMP-1440, HLMP-1540, HLMP-K640 T-13/4 (5 mm), T-1 (3 mm), Ultra-Bright LED Lamps Data Sheet. https://eu.mouser.com/datasheet/2/678/AVGOS05392_1-2524844.pdf.

[25] Antti Näsilä, Christer Holmlund, Endija Briede, Rami Mannila, Roberts Trops, Martti Blomberg, Ingmar Stuns, Bin Guo, Kai Viherkanto, Kari Rainio, Heikki Saari, and Anna Rissanen. Cubic-inch MOEMS spectral imager. In *Proc. SPIE 10931, MOEMS and Miniaturized Systems XVIII*, page 109310F, 4 March 2019. https://www.spiedigitallibrary.org/conference-proceedings-of-spie/10931/2508420/Cubic-inch-MOEMS-spectral-imager/10.1117/12.2508420.short?SSO=1.

[26] Antti Näsilä, Heikki Saari, H. Ojanen, I. Stuns, K. Viherkanto, Christer Holmlund, R. Mannila, Roberts Trops, and E. Briede. Miniaturized spectral imaging technologies at VTT. In *OIE 2019 The 13th Japan-Finland Joint Symposium on Optics in Engineering*, Espoo and Tallinn, 26-30 August 2019. https://docplayer.fi/151101301-Proceedings-of-oie-2019.html.

[27] The economic impacts of inadequate infrastructure for software testing. Planning Report 02-3, National Institute of Standards and Technology, 2002. https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf.

[28] 1/2.5-Inch 5 Mp CMOS Digital Image Sensor MT9P031. https://www.onsemi.com/pdf/datasheet/mt9p031-d.pdf.

[29] Pradeep Parthiban. 7 reasons why software testing is important. https://www.indiumsoftware.com/blog/why-software-testing/.

[30] Ilkka Pölönen, S. Rahkonen, L. Annala, and N. Neittaanmäki. Convolutional neural networks in skin cancer detection using spatial and spectral domain. In *Photonics in Dermatology and Plastic Surgery 2019*, volume SPIE-10851, page 108510B. International Society for Optics and Photonics, February 2019.

[31] S. Rahkonen, E. Koskinen, Ilkka Pölönen, T. Heinonen, T. Ylikomi, S. Äyrämö, and M. A. Eskelinen. Multilabel segmentation of cancer cell culture on vascular structures with deep neural networks. *Journal of Medical Imaging*, 7(2):024001, 2020.

[32] Anna-Maria Raita-Hakola, Leevi Annala, Vivian Lindholm, Roberts Trops, Antti Näsilä, Heikki Saari, Annamari Ranki, and Ilkka Pölönen. FPI based hyperspectral imager for the complex surfaces—calibration, illumination and applications. *Sensors*, 22:3420, 2022.

[33] Heikki Saari, H. Ojanen, and Ingmar Stuns. Novel hyperspectral imager based on angle-tuned multi pass band filter, LEDs and RGB image sen-

sor. In *OIE 2019 The 13th Japan-Finland Joint Symposium on Optics in Engineering*, Espoo and Tallinn, 26-30 August 2019. https://docplayer.fi/151101301-Proceedings-of-oie-2019.html.

[34] Heikki Saari, Ilkka Pölönen, Heikki Salo, Eija Honkavaara, Teemu Hakala, Christer Holmlund, Jussi Mäkynen, Rami Mannila, Tapani Antila, and Altti Akujärvi. Miniaturized hyperspectral imager calibration and UAV flight campaigns. In *Sensors, Systems, and Next-Generation Satellites XVii*, volume SPIE-8889. International Society for Optics and Photonics, August 2013.

[35] Joseph S. T. Smalley, Xuexin Ren, Jeong Yub Lee, Woong Ko, Won-Jae Joo, Hongkyu Park, Sui Yang, Yuan Wang, Chang Seung Lee, Hyuck Choo, Sungwoo Hwang, and Xiang Zhang. Subwavelength pixelated CMOS color sensors based on anti-Hermitian metasurface. *Nature Communications*, 11(3916):–, March 2020.

[36] Fajar Suryawan. FPGA-based geometric test pattern generator. In *Proceedings of the 3rd International Conference on Electrical, Communication and Computer Engineering (ICECCE)*, pages –, Kuala Lumpur (Malaysia), 12.-13. June 2021. IEEE, Piscataway, NJ.

[37] Roberts Trops, Anna-Maria Hakola, Severi Jääskeläinen, Antti Näsilä, Leevi Annala, Matti A. Eskelinen, Heikki saari, Ilkka Pölönen, and Anna Rissanen. Miniature MOEMS hyperspectral imager with versatile analysis tools. In *Proc. SPIE 10931, MOEMS and Miniaturized Systems XVIII*, page 109310W, 4 March 2019. https://www.spiedigitallibrary.org/conference-proceedings-of-spie/10931/2506366/Miniature-MOEMS-hyperspectral-imager-with-versatile-analysis-tools/10.1117/12.2506366.short.

[38] Roberts Trops, Antti Näsilä, Christer Holmlund, and Heikki Saari et.al. Cubic-inch hyperspectral imager for space exploration. In *Finnish Satellite Workshop & Remote Sensing Days 2020*, Aalto University, Dipoli, Finland, 20-22 January 2020. https://drive.google.com/file/d/1iglT16hqzKl1G0C5IaMCHgXPb5RLmFEh/view.

[39] F. Vermolen and Ilkka Pölönen. Uncertainty quantification on a spatial Markov-chain model for the progression of skin cancer. *Journal of Mathematical Biology*, 80(3):545–574, 2020.

[40] Test pattern generator (IP). https://www.xilinx.com/products/intellectual-property/tpg.html.

[41] Video test pattern generator. Datasheet, 2022. https://www.zipcores.com/datasheets/tpg.pdf.

[42] Digilent's Zybo Z7-20 SoC board. https://digilent.com/reference/programmable-logic/zybo-z7/reference-manual (Accessed 13.5.2022).

A Program files

The following files are available at `https://lipas.uwasa.fi/~TAU/reports/report22-1/`

	Program	*Size (lines)*	*Comment*
	`ARMmovie2.c`	~2,000	the main program
	`ARMmovie2.h`	~400	definition
	`QEtable.h`	~650	QE curves
	`LEDs.h`	~250	LED models
	`ARMmovie.txt`	~20	parameters
	`*.ppm`	á ~2...14Mbytes	ext. input image files
Total	~5+ files	~3,300	+ext. .ppm images