Development of a high-speed 3D scanner system using Digital Fringe Projection techniques

Samuel Veloso López[1, 2]

[1]UPM, Technical University of Madrid, Spain
[2]NUIM, Maynooth University, Ireland

Abstract

Our brain and eyes make such an extraordinary job that one may have felt inclined to think that reconstructing objects in our 3D world is a trivial task, but for machines it is a huge challenge that researchers have been studying for ages. The purpose of this project is to develop a 3D scanner system to recover the shape of any object recording with the smartphone camera a moving fringe pattern projected onto the object. A DLP LightCrafter 4500 projector will be used to project fringe patterns at high-speed, a Raspberry Pi will be used to control patterns configuration and digital fringe projection techniques will be applied to reconstruct the 3D shape. Finally, it is discussed what best conditions to reconstruct the shape are, along with a time analysis to measure the performance of the algorithm developed and suggesting how to achieve real-time in future projects. The development of the system opens a wide range of possibilities for 3D imaging applications e.g. game industry, medical imaging and diagnosis, online inspection and quality control, recognition, etc.

Keywords: 3D shape measurement; Digital fringe projection; Digital light processing (DLP); Smartphone camera.

1. Introduction

3D reconstruction is an ancient problem in computer vision, or in other words, trying to **recover the 3D shape** of an object from its 2D image representation. There are plenty of solutions for this problem out there, e.g. laser triangulation, structured light, stereo vision, photogrammetry, time of flight, interferometry, etc. [1]

The possibility of reconstructing a 3D shape is **encouraging** and results in a large number of new technological solutions. Traditionally, medical imaging has been the field most interested in 3D reconstruction [2], but in recent years, it has been extended to fields like robotics, industrial systems and virtual reality [3]. Furthermore, it is considered a hot topic in most universities and companies, which are trying to contribute as much as possible to this area.

This project will apply DFF (Digital Fringe Projection) techniques to get 3D shape measurement from a video. A **DLP LightCrafter 4500** projector module controlled by a **Raspberry Pi** will project a sequence of **moving fringes patterns** onto an object, which will be recorded using a **smartphone camera**, and finally applying image processing algorithms the 3D shape of the object will be recovered.

The problem could be divided into **three different stages**: moving fringe patterns projection, video recording and 3D shape measurement.

Moving fringe patterns projection makes use of RGB multiplexing technology thanks to **DMD micromirroring technology** developed by Texas Instrument which allow to project patterns sequences up to 4225 Hz. Video recording is performed with a smartphone camera (30fps usually). 3D shape measurement needs to extract relevant patterns from the video, detect the object, remove the background and retrieve the 3D shape.

2. Related Work

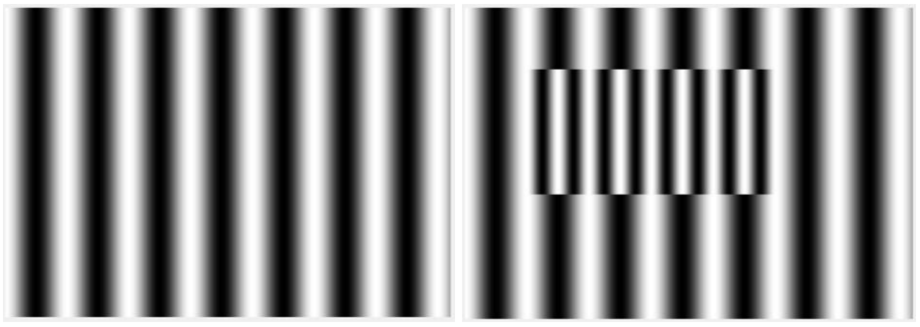
The main problem of the system is how to measure the change that the shape of an object causes in the instantaneous phase of a sinusoidal fringe pattern when the object is projected onto it. This is an ancient problem, how to calculate the **instantaneous phase** of a signal when solely the amplitude is known.

Fourier [4], Wavelet [5] or Hilbert [6] transforms are traditional methods to solve this problem but for 3D shape measurement are **not accurate** enough. Song Zhang [7] proposes a new phase recovery method to retrieve the instantaneous phase known as phase-shifting technique.

3.1. Digital fringe projection approach

Sinusoidal fringe pattern image is given by next equation:

$$I(x,y) = I_0 + I_A \cdot cos\left(\frac{2\pi}{T}x + \varphi_0\right)$$


Sinusoidal fringe pattern / Object overlapped

As we can see, the object modifies the instantaneous phase of the original fringe pattern (henceforth calibration plane). To transform this modification into the shape of the object is needed to get the instantaneous phase of both and subtract them.

First of all we are recovering the instantaneous phase with **Hilbert Transform** method and **Song Zhang phase-shifting** technique [7], to subsequently compare both results.

3.2. Hilbert Transform method

The intensity value of each row is given by next equation:

$$I_{row}(x) = I_A \cdot cos\big(\varphi(x,T,\varphi_0)\big)$$

In order to get the instantaneous phase of this row, we can use the Hilbert Transform as follows:

$$I_{Hilbert} = I_{row} + j \cdot H(I_{row}) = I_A \cdot \big(cos(\varphi) + j \cdot sin(\varphi)\big)$$

With this new signal we can get the relative instantaneous phase:

$$\varphi(x,T,\varphi_0) = tan^{-1}\left[\frac{\mathfrak{I}(I_{Hilbert})}{\mathfrak{R}(I_{Hilbert})}\right] = tan^{-1}\left[\frac{I_A \cdot sin(\varphi)}{I_A \cdot cos(\varphi)}\right] = tan^{-1}\big(tan(\varphi)\big)$$

Applying this formula to all the pixels of both images we get the relative instantaneous phase map for the calibration plane $\varphi_{cplane}(x,y)$ and for the object $\varphi_{object}(x,y)$.

3.3. Song Zhang phase-shifting technique

This technique uses 3 or more fringe patterns, each one with a different phase shift:

$$I_s(x,y,\varphi_s) = I_0 + I \cdot cos(\varphi(x,T,\varphi_0) + \varphi_s)$$

Three is the minimum number of patterns because the instantaneous phase is got building a system of equations with 3 variables $(I_0,\ I,\ \varphi)$ and therefore we need 3 independent equations:

$$\begin{cases} I_1 = I_0 + I \cdot cos\left(\varphi - \frac{2\pi}{3}\right) = I_0 + Icos(\varphi)cos\left(\frac{2\pi}{3}\right) + Isin(\varphi)sin\left(\frac{2\pi}{3}\right) = I_o - \frac{I}{2}cos(\varphi) + \frac{I\sqrt{3}}{2}sin(\varphi) \\ I_2 = I_0 + I \cdot cos(\varphi) \\ I_3 = I_0 + I \cdot cos\left(\varphi + \frac{2\pi}{3}\right) = I_0 + Icos(\varphi)cos\left(\frac{2\pi}{3}\right) - Isin(\varphi)sin\left(\frac{2\pi}{3}\right) = I_o + \frac{I}{2}cos(\varphi) - \frac{I\sqrt{3}}{2}sin(\varphi) \end{cases}$$

To find the instantaneous phase we have to isolate it:

$$I_1 - I_3 = \sqrt{3}Isin(\varphi)$$

$$2I_2 - I_1 - I_3 = 2Icos(\varphi) + Icos(\varphi) = 3I \cdot cos(\varphi)$$

The tangent of the phase relates both equations:

$$\frac{I_1 - I_3}{2I_2 - I_1 - I_3} = \frac{\sqrt{3}Isin(\varphi)}{3I \cdot cos(\varphi)} = \frac{\sqrt{3}}{3}tan(\varphi)$$

Finally:

$$\varphi(x,y) = tan^{-1}\left(\frac{\sqrt{3}(I_1 - I_3)}{2I_2 - I_1 - I_3}\right)$$

In order to improve the accuracy of the system, we could add redundancy using more than 3 patterns. Though, Carré and Hariharan [8] proposes a formula to tolerate large phase shift error using 4 or 5 fringe patterns respectively:

$$\varphi(x,y) = tan^{-1}\left(\frac{\sqrt{(3(I_2 - I_3) + I_4 - I_1) \cdot (I_1 + I_2 - I_3 - I_4)}}{I_2 + I_3 - I_1 - I_4}\right)$$

Carré formula

$$\varphi(x,y) = tan^{-1}\left(\frac{2(I_2 - I_4)}{2I_3 - I_1 - I_5}\right)$$

Hariharan formula

3.4. Unwrapping phase map

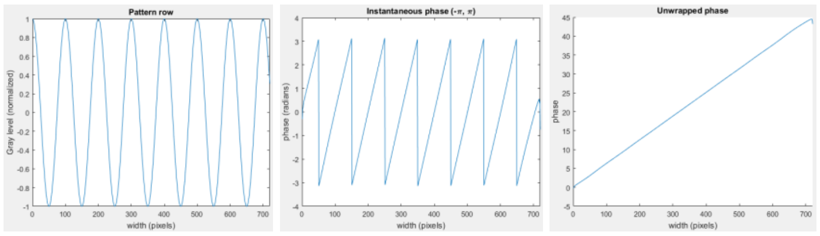
The relative instantaneous phase obtained is wrapped, this means that all points values are constrained to the range $(-\pi, +\pi)$ and an unwrapping algorithm [9] must be applied in order to correct it:

$$\boldsymbol{for}\ i \in \boldsymbol{all\ columns} \begin{cases} \boldsymbol{if}\ \Delta\varphi(i) \geq 2\pi \rightarrow \Phi(i,y) = \mathcal{U}[\varphi(i,y)] = \varphi(i,y) + 2\pi \\ \boldsymbol{if}\ \Delta\varphi(i) < 2\pi \rightarrow \Phi(i,y) = \mathcal{U}[\varphi(i,y)] = \varphi(i,y) - 2\pi \end{cases}$$

$$\boldsymbol{where}\ \Delta\varphi(x) = \varphi(x) - \varphi(x-1)$$

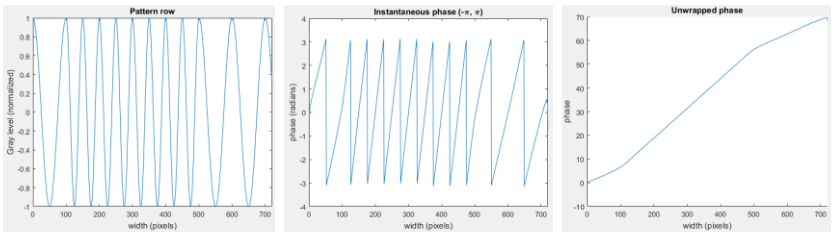
3.5. 3D Shape measurement

Applying all these steps for any row of the calibration plane, we get next result:



Calibration plane row / Instantaneous phase / Unwrapped phase

Following we can observe how the object changes the phase of the pattern, applying same steps for a row with the object overlapped:

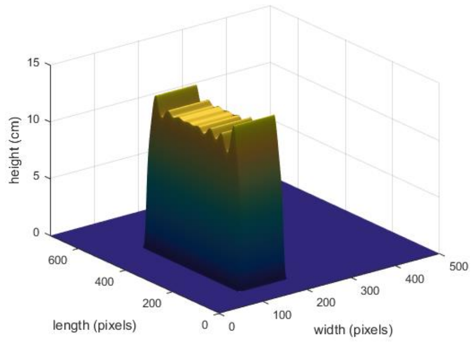


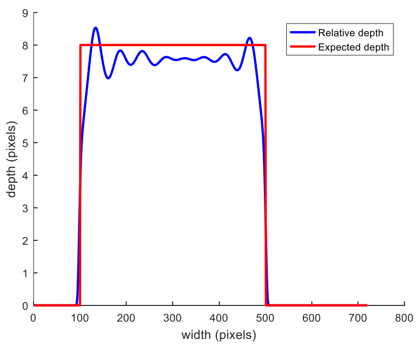
Calibration plane row / Instantaneous phase / Unwrapped phase

The difference between unwrapped phase maps is what we need to derive the shape of the object. If we subtract the unwrapped phase of the object from the unwrapped phase of the calibration plane, we get the relative depth of the object:

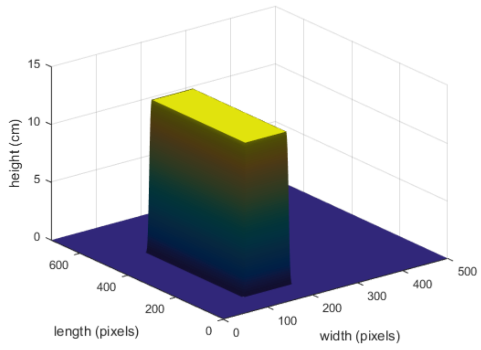
$$\Omega_{depth} = \frac{d\left(\Phi_{object} - \Phi_{plane}\right)}{dx}$$

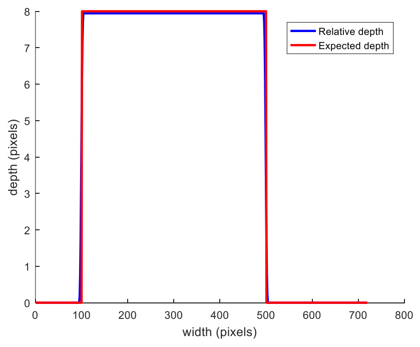
The 3D shape measurement result using **Hilbert transform** method is:





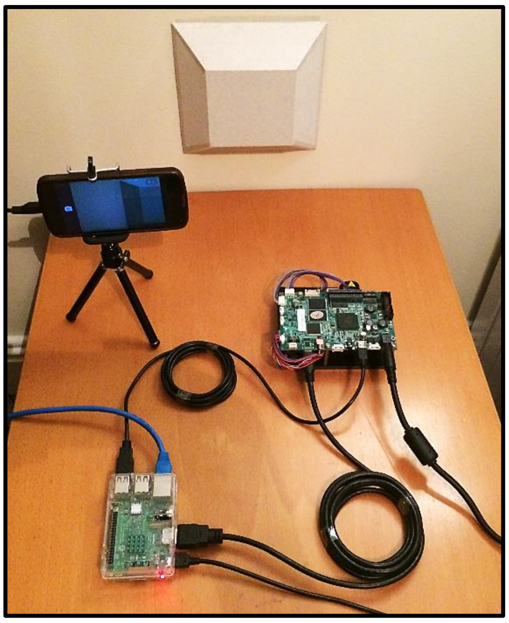
And using **Song Zhang** technique is:



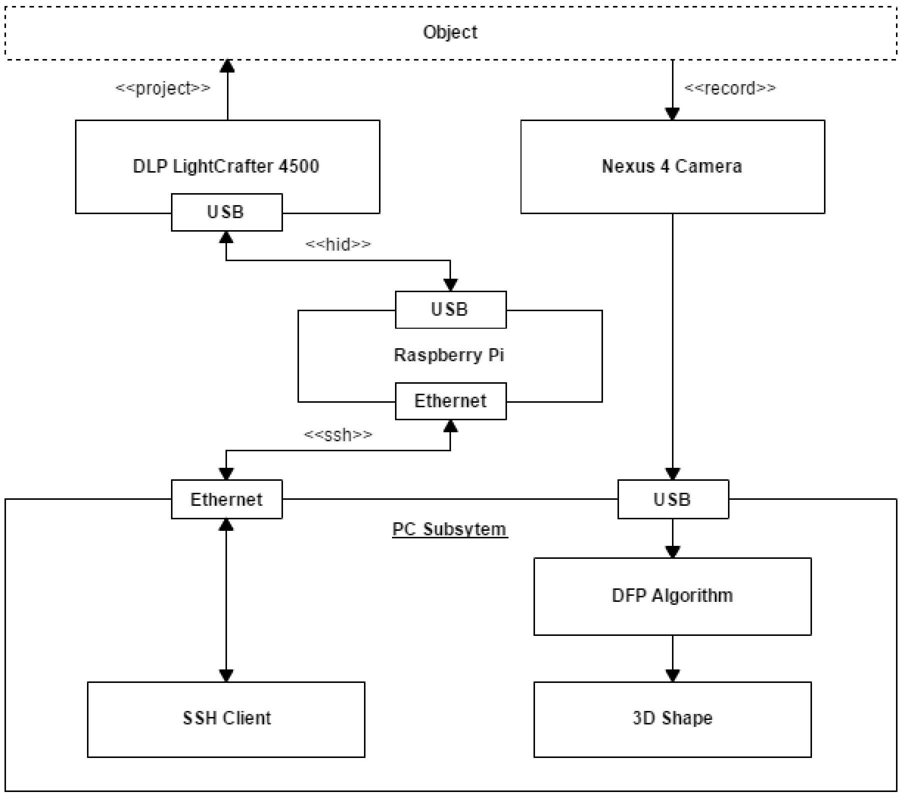


4. The Solution

To recover the 3D shape of an object from a video, it is needed to **project fringe patterns** onto the object, **record** the video, **extract** the relevant patterns from it and finally, **retrieve** the shape using the DFP algorithm. To make this possible I have designed and implemented next system:



The system consists of a **DLP LightCrafter 4500** projector connected to a **Raspberry Pi 3 Model B** which is responsible for creating the fringe patterns to project. At the same time, the RPi is connected to PC via **SSH** to adjust fringe patterns parameters. A Nexus 4 smartphone camera will record the videos that will be sent to the PC in order to retrieve the 3D shape using **MATLAB**.



Subsystems diagram

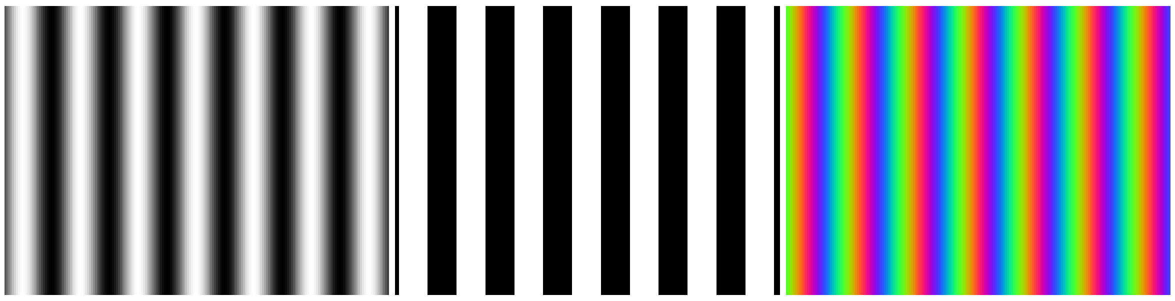
Throughout this chapter I will explain in detail how each subsystem works. Therefore, the first feature to cover is how to project a sinusoidal fringe pattern onto the object using a Raspberry Pi.

4.1. Generate fringe patterns

Texas Instruments provides a **Windows GUI** for the DLP LightCrafter 4500 which allows to project any 24-bit image with a size of 912x1140 stored into de 32MB flash memory [10].To store new images in flash memory, these images must be compressed in a new firmware file and downloaded to the LightCrafter 4500. Hence, each time we want to project a new pattern we have to upgrade the firmware, a risky (flash could corrupt) and very slow (more than 5 minutes) process.

In order to solve this problem a Raspberry Pi 3 Model B will be connected to the LightCrafter 4500 via HDMI to send the patterns though it, which is an instant and safe process. To control the DMD module and generate the patterns, a program in C++ named as "pGenerator" has been developed.

Whit pGenerator it is possible to generate different fringe patterns, choosing best configuration for 3 different working modes: **fringe mode**, **binary mode** (not used in this project) and **high-speed mode**.


Fringe mode / Binary Mode / High-speed mode

All these 3 modes work writing 2D Linux framebuffer (/dev/fb0) configured as 24-bit/RGB video mode, where each channel is given by a different operation on the **reference function δ**:

$$\delta(x, I, T, \phi) = \frac{1}{2}\left(255 + I\cos\left(\frac{2\pi x}{T} + \phi\right)\right)$$

I, T and ϕ are chosen by the user. Where I is the peak intensity value (max $255 \equiv 2^8-1$), T is the fringe period in pixels and ϕ is the initial phase shift in radians.

For **video mode**, each RGB channel is given by:

$$R(x, y) = G(x, y) = B(x, y) = \delta(x, I, T, \phi)$$

For **binary mode**, each RGB channel is given by:

$$R(x, y) = G(x, y) = B(x, y) = \begin{cases} \dfrac{(255 + I)}{2}, & \delta(x, I, T, \phi) > \dfrac{I}{2} \\ \dfrac{(255 - I)}{2}, & \delta(x, I, T, \phi) \leq \dfrac{I}{2} \end{cases}$$

And for **high-speed mode**, each RGB channel is given by:

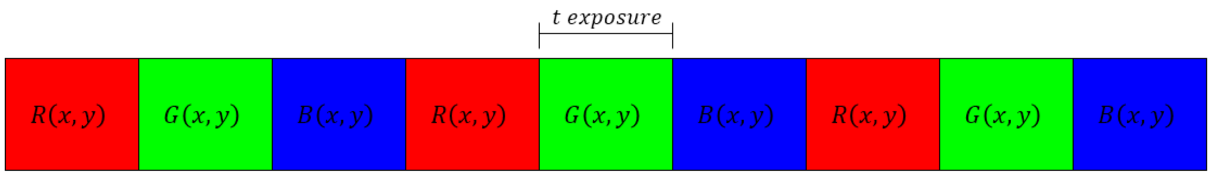
$$R(x, y) = \delta\left(x, I, T, \phi + \frac{2\pi}{3}\right); G(x, y) = \delta(x, I, T, \phi); B(x, y) = \delta\left(x, I, T, \phi - \frac{2\pi}{3}\right)$$

Fringe mode allows to project static patterns and it is enough to recover the 3D shape, but if we want to recover the shape from a video we have to go further and implement high-speed mode.

4.2. High-speed mode

Thanks to the DLP hearth, the digital micromirror device (DMD), it is very easy to generate video patterns with an unbeatable quality compared to other technologies, such as LCD or LCOS [11]. DLP LightCrafter 4500 is able to multiplex up to **4225 Hz/230 µs** (1-bit color depth) and up to **120 Hz/8333 µs** (8-bit color depth) pattern rate.

High-speed mode encodes in each 8-bit RGB channel one sinusoidal fringe pattern with different phase shift (+2π/3, 0, -2π/3). In this way, the C++ program has to talk to the LightCrafter to multiplex a channel each exposure period t (microseconds).



Light multiplexing RGB channels

The way in which the Raspberry Pi communicates with the LightCrafter was a huge challenge. Fortunately the LightCrafter GUI is **open source** as well as its SDK, but it isn't well documented and the GUI is compiled for Windows whereas the Raspberry Pi runs an ARM Linux distro.

This is why, firstly, I had to recompile everything on RPi and then add the multiplexing feature to the C++ program, extracting the relevant code, delving into the original GUI source code.

The SDK uses **USB HID protocol** to communicate with LightCrafter (VID 0x045, PID 0x6401) using 64-bit data blobs. For that, it make use of **HIDAPI** library, which is a multiplatform C library to interconnect any HID device.

To allow the SDK to use HIDAPI, I packed everything necessary (see *supporting/hidapi*) to compile and install it on the Raspberry Pi typing next commands:

```
pi@rpi:~ $ sudo apt-get install libusb-1.0-0-dev
pi@rpi:~ $ cd hidapi
pi@rpi:~/hidapi $ sudo make
pi@rpi:~/hidapi $ sudo make install
pi@rpi:~/hidapi $ sudo ldconfig
```

I did also pack the SDK (see *supporting/sdk*) to compile and install it on the Raspberry Pi typing same commands:

```
pi@rpi:~ $ cd sdk
pi@rpi:~/sdk $ sudo make
pi@rpi:~/sdk $ sudo make install
pi@rpi:~/sdk $ sudo ldconfig
```

As a result, it is possible to use **all SDK functions** (USB_Connect, GetFirmwareVersion, SetLedEnables, etc.) in pGenerator and take control of the LightCrafter, simply adding the library -*ldlp4500sdk* to the Makefile.

Looking deeply through the original GUI source code, I could find the way to implement RGB **multiplexing feature** to pGenerator. Thus, I decided to write a driver that uses SDK functions to handle specifically the multiplexing process.

The driver provides next functions:

Function	Description
DLP4500_WakeUp	Wake up the projector if it is in standby mode.
DLP4500_ChangeToVideoMode	Change current mode to video mode.
DLP4500_ChangeToPatternMode	Change current mode to pattern sequence mode.
DLP4500_LoadFringeSequence	Create a fringe pattern sequence formed by 3 different sub-sequences (one per each RGB channel) and certain user's chosen color (red, green, yellow, blue, magenta, cyan or white). Each sub-sequence lasts exposure period t.
DLP4500_SetSequenceModeConfig	Set pattern exposure period t in milliseconds and HDMI as input video interface.
DLP4500_ValidateSequence	Check if the pattern sequence is well-formed and the exposure period is not too small (at least 8333 µs for 8-bit sub-sequences).
DLP4500_PlaySequence	Start playing the pattern sequence
DLP4500_StopSequence	Stop the pattern sequence.

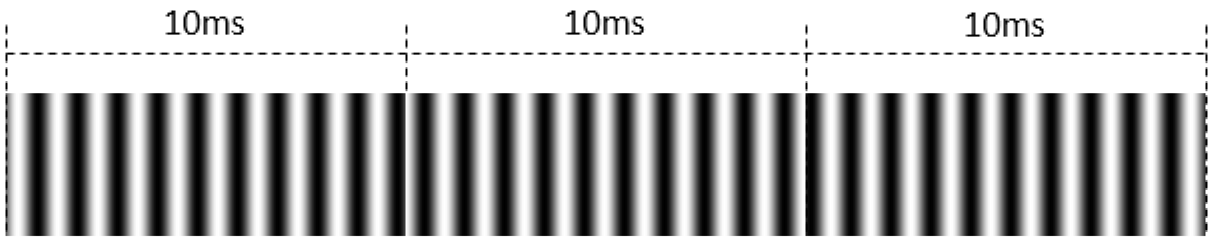
Appendix I details multiplexing process using these driver functions, by means of a UML sequence diagram.

Now, we just have to run pGenerator with desired configuration (peak intensity, fringe pattern period, initial phase shift, pattern exposure period and color) in high-speed mode and record the sequence that will be used to extract fringe patterns.

For example, to run a white fringe pattern sequence with a peak intensity of 150, 10 pixels period, no initial phase shift and 10ms per sub-sequence, we should type:

```
pi@rpi:~$ ./pGenerator -s -I 150 -T 10 -t 10000 -c 7
```

Which would project next fringe sequence, ready to be recorded with any smartphone camera.



RGB multiplexing pattern sequence example

4.3. Record video

Any camera could be used to record the video. In theory, the only **limit** is the framerate:

$$f_{camera} \geq \frac{1}{exposureperiod}$$

However, in practice smartphone camera is not prepared to record fringe patterns at exposure period and a slower projection frequency must be chosen.

To record the video sequence I will be using **Nexus 4 camera** (CMOS, 1920x1080, 30fps). In evaluation chapter there are also results with iPhone 6 camera.

4.4. Extract fringe patterns

On the one hand, we need to record the **calibration-plane** and on the other hand we need to record the **object** to recover the shape from. Once we have both videos, we can proceed to extract the relevant images.

Each video is mathematically defined as:

$$v_c[n], n \in 0, \dots, N_1$$

$$v_o[n], n \in 0, \dots, N_2$$

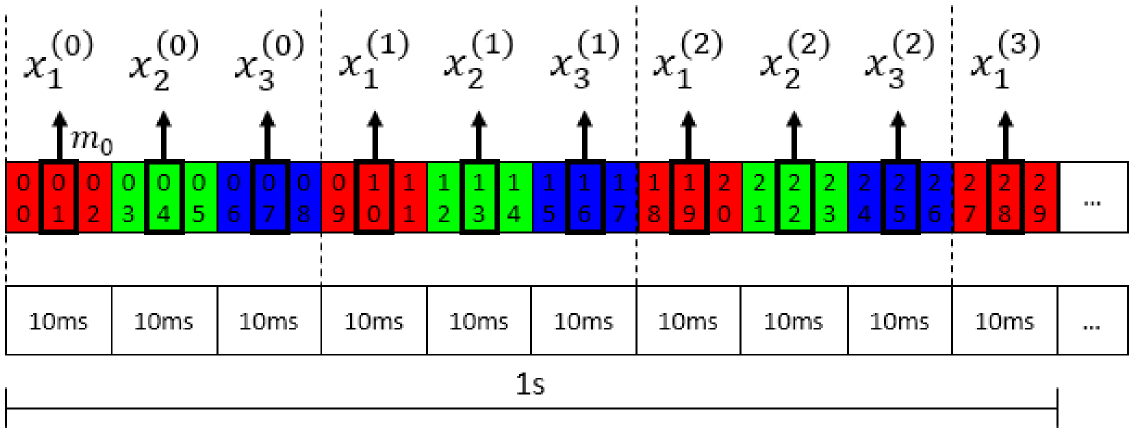
$$Where N_1 \wedge N_2 are the number of frames \in cplane \wedge object video respectively$$

To extract the patterns individually from these videos it is needed to know projecting frequency f_p and camera frequency f_c. Then, for each sequence j we can extract its M patterns as follows:

$$C_i^{(j)} = v_c\left[\frac{f_c}{f_p}(i + j \cdot M - 1) + m_0\right], \qquad i \in 1, \dots, M, \qquad j \in 0, \dots, \lfloor \frac{N_1 \cdot f_p}{M \cdot f_c} - 1 \rfloor$$

$$O_i^{(j)} = v_o\left[\frac{f_c}{f_p}(i + j \cdot M - 1) + m_0\right], \qquad i \in 1, \dots, M, \qquad j \in 0, \dots, \lfloor \frac{N_2 \cdot f_p}{M \cdot f_c} - 1 \rfloor$$

In high-speed mode, due to multiplexing process, M is equals to 3, the number of RGB channels. Even though, it is possible to retrieve the 3D shape with a different number of pattern as we have seen before (Carré or Hariharan algorithm) but other multiplexing technique would have to be used.



$x_1^{(0)}$ $x_2^{(0)}$ $x_3^{(0)}$ $x_1^{(1)}$ $x_2^{(1)}$ $x_3^{(1)}$ $x_1^{(2)}$ $x_2^{(2)}$ $x_3^{(2)}$ $x_1^{(3)}$
m_0
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 ...
10ms 10ms 10ms 10ms 10ms 10ms 10ms 10ms 10ms 10ms ...
1s

Patterns extraction diagram ($f_c = 30Hz \wedge f_p = 10Hz$

Now, we have one-to-one correspondence between calibration-plane patterns and object patterns. If the elements in a pair of patterns have different phase shift we say this pair is non-synchronized and a **synchronization correction** must be applied.

The object pattern corresponding to a certain calibration-plane pattern is the one with least SSIM (Structural Similarity Index) among all object patterns in a sequence j:

$$\cancel{O}_i^{(j)} = argmax\left(SSIM\left(C_i^{(j)}, \left\{O_i^{(j)}, \dots, O_M^{(j)}\right\}\right)\right)$$

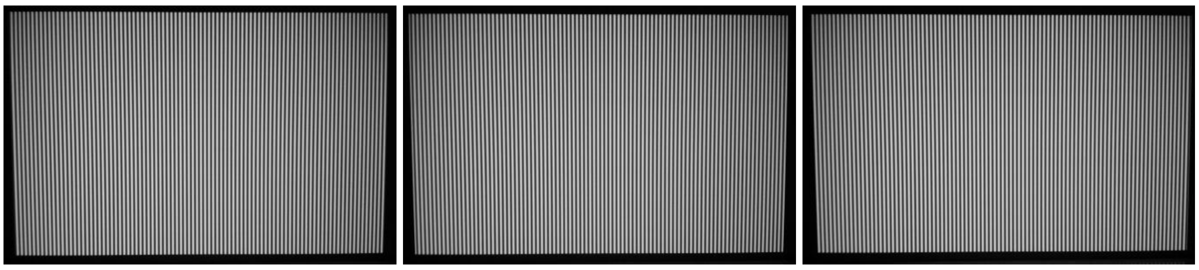
For real-time applications, it is very important to record videos synchronously because the synchronization algorithm could slow down the process significantly.

4.5. DFP Algorithm

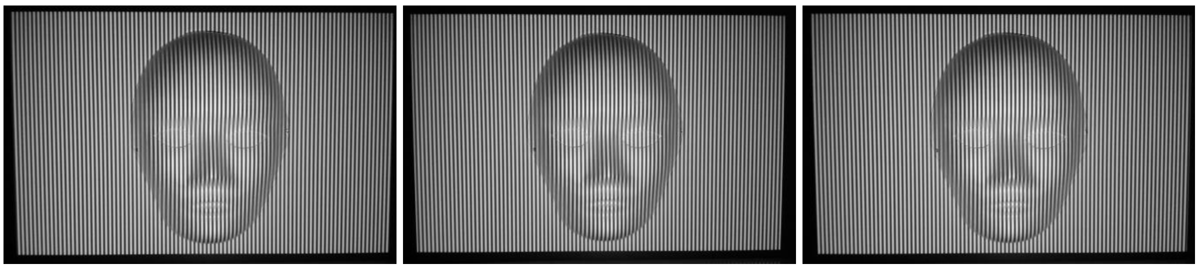
The best way to show how the algorithm works is using a real object to support the explanation of all steps involved in the process. Next **mask**, due to its flat character, is a very good choice to attempt to recover the 3D shape from:

Target object

First of all, as we have seen previously, we need to record **two different videos** (calibration plane and object video). Once the patterns are extracted from a single sequence, we will have a similar result to following depending on the projection configuration we had chosen:



Calibration plane 3 pattern sequence (+2π/3, 0, -2π/3)



Object 3 pattern sequence (+2π/3, 0, -2π/3)

In this case the camera frequency is 30 fps (Nexus 4 smartphone), the pattern exposure is 100 ms and the number of patterns per sequence is 3 due to DLP RGB multiplexing technique has been employed.

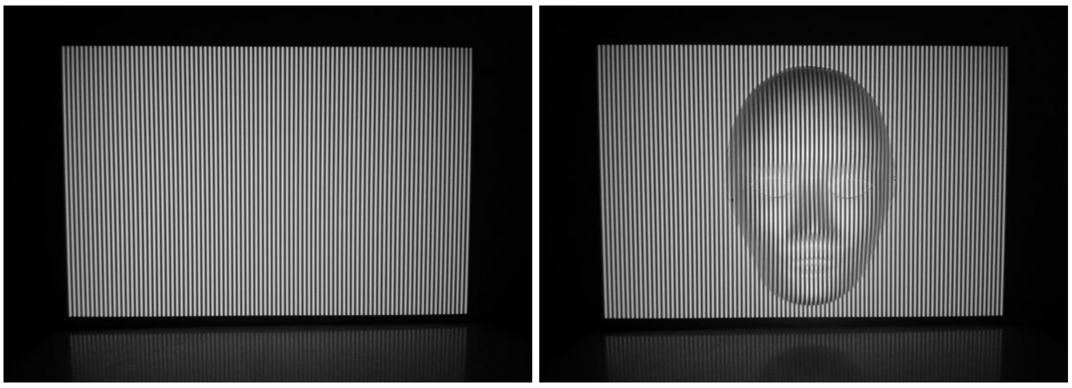
Once we have all patterns for a sequence, we can proceed with the algorithm as detailed in **appendix II flow diagram**.

4.5.1. Detect object and remove background

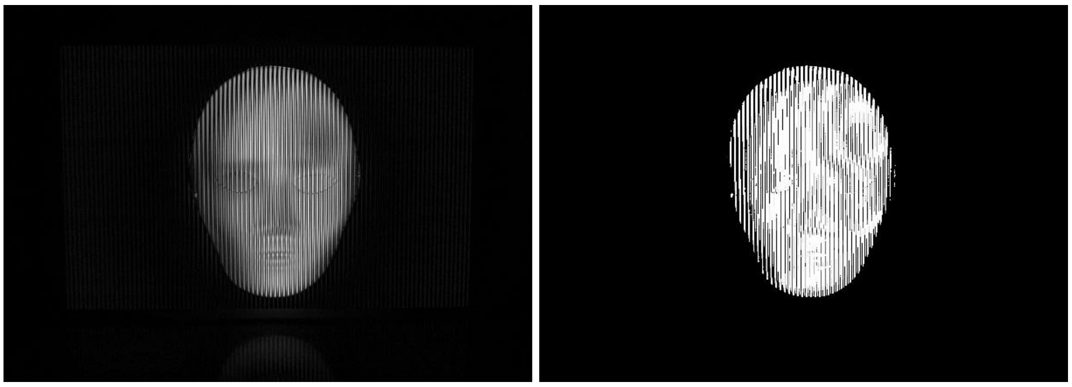
In order to make the algorithm works properly it is needed to **remove the background**, this is, detect the object and separate the background and the foreground.

There are **many techniques** [12] to remove background from an image (contour detection, mixture of Gaussians, K-mean clustering, global image thresholding…). Due to fringe images nature we have to apply a **specific algorithm** using some of these ideas.

The approach starts taking advantage we have already got a **correspondence** between two different frames, one with the object and the other one without it:


First pattern background-object correspondence

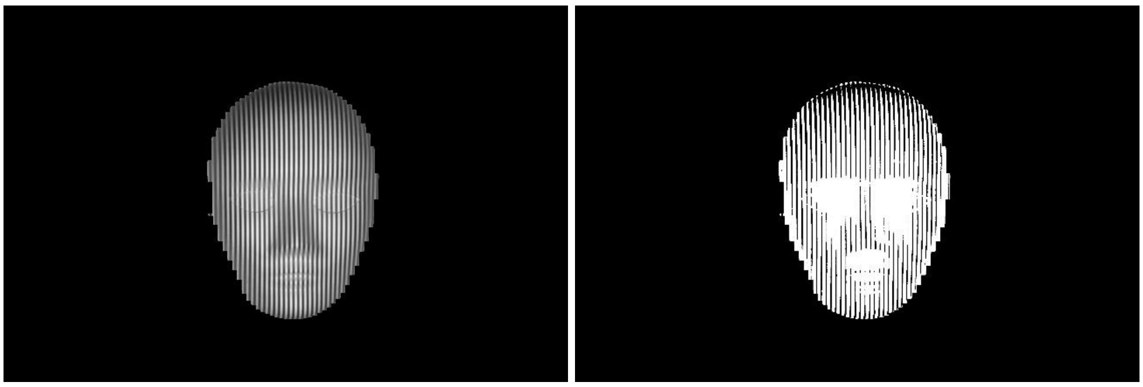
If we get the image of the difference between both images and convert it to a binary image using a **threshold** of 0.2 we can isolate the object:


Object-background difference / Difference thresholded mask

The fringe nature causes a *barcode effect* in the mask that must be fixed. As a result, the mask is **dilated** (removing holes) and **eroded** with same 64 pixels square factor:

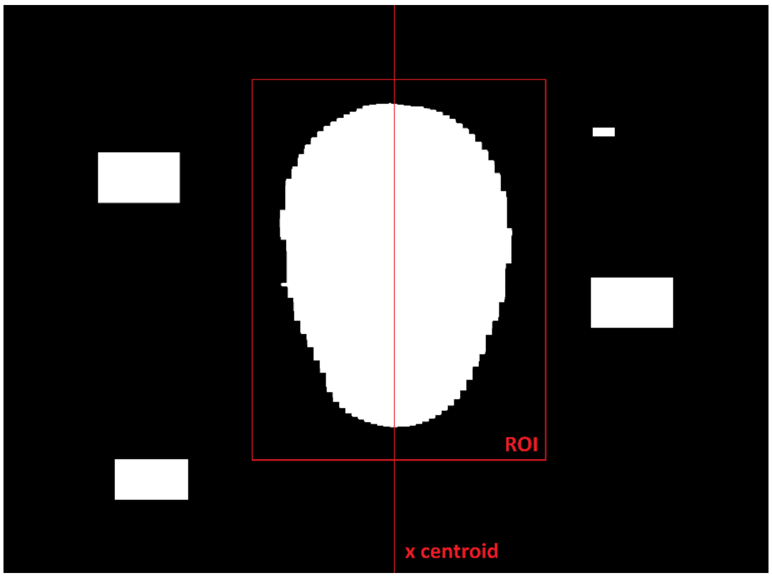
Mask dilated / Mask eroded

If the object casts a **shadow** on the background it must be removed because it will look like noise in the 3D final shape. The original object pattern is masked and a threshold is applied to remove the shadow, generating the new mask:


Object pattern masked / New mask without shadow

Same dilating and eroding step than before is applied again with a 64 pixels square factor in order to remove *barcode effect*.

Finally, all image objects that are not part of the main object are removed. The mask is labelled and the main object is the labelled object with largest area. Now, the main object is used to define the region of interest to crop patterns, the **final mask** and the x centroid that will be used later on unwrapping phase step.



Final mask result

4.5.2. Get wrapped phase

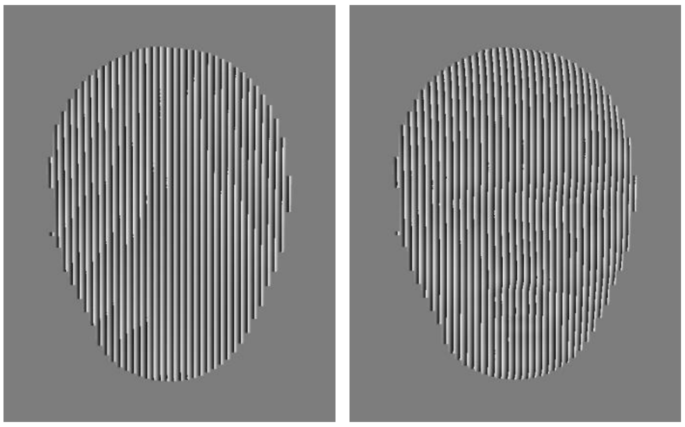
The next step is to get the wrapped phase for calibration plane patterns and object patterns, previously cropped using the mask.

As we are working with 3-patterns algorithm the formula applied is **Song Zhang** formula:

$$\varphi(x,y) = tan^{-1}\left(\frac{\sqrt{3}(I_1 - I_3)}{2I_2 - I_1 - I_3}\right)$$

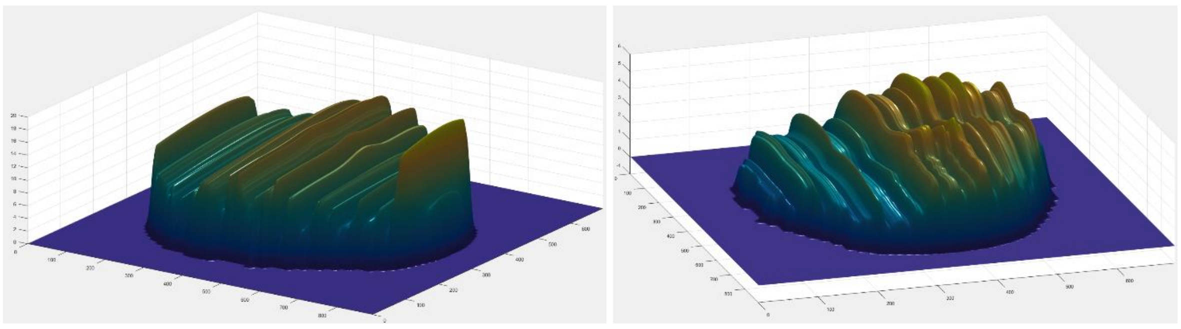
Song Zhang formula

And the result for calibration phase and object after applying the formula is:


Calibration plane wrapped phase / Object wrapped phase

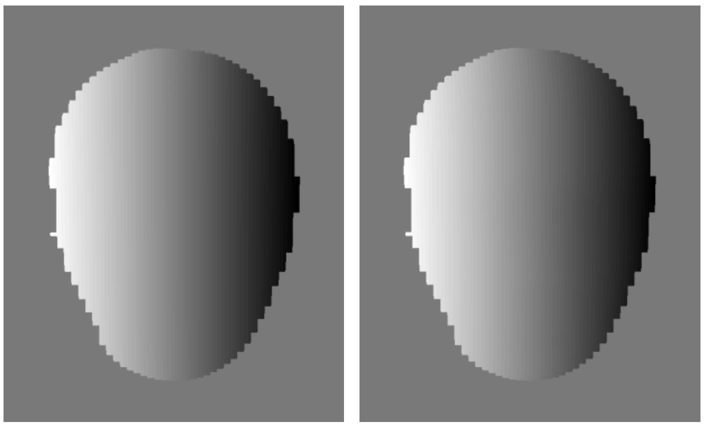
4.5.3. Get unwrapped phase

We are one step away from retrieving the 3D shape, since it is the difference between calibration plane and object unwrapped phase. MATLAB provides **2D unwrapping function** (horizontal and vertical axis) but it is row/column **neighbor independent**:


Horizontal unwrapping / Vertical unwrapping (exaggeration)

All rows should have same height reference, this is corrected using a column (x centroid coordinate as default) as column reference and all rows will be modified to match the height of this column following next algorithm:

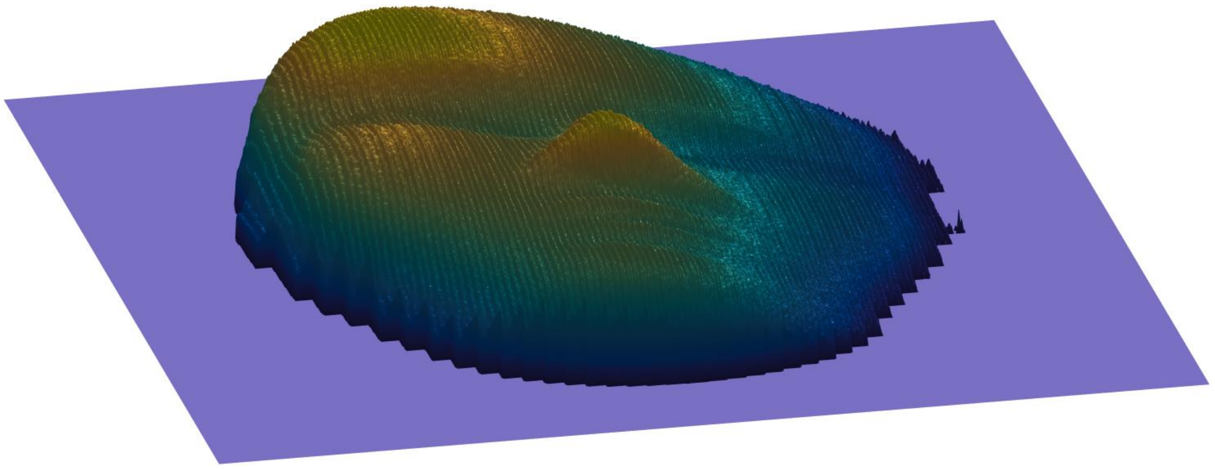
$$\varphi_{reference}(y) = \boldsymbol{unwrap}\big(\varphi(xcentroid, y)\big)$$
$$\boldsymbol{for}\ j \in \boldsymbol{all\,rows} \rightarrow \Phi(x,j) := \Phi(x,j) - \Phi(k,j) + \varphi_{reference}(j)$$


Calibration plane unwrapped phase map / Object unwrapped phase map

4.5.4. Calibrate 3D shape

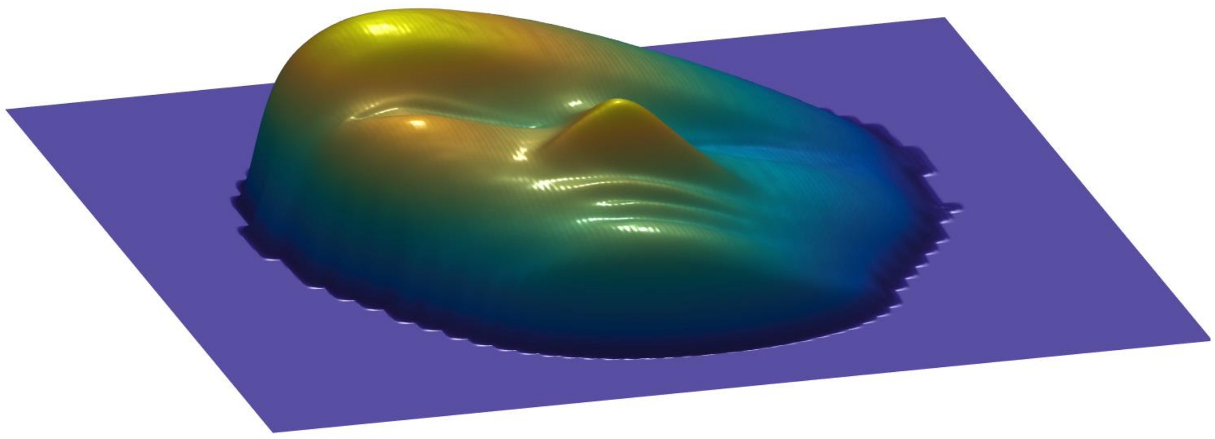
In the end we can recover the 3D shape **subtracting** the object unwrapped phase map from the calibration plane unwrapped phase map:

$$z(x,y) = \Phi_{cplane}(x,y) - \Phi_{object}(x,y)$$



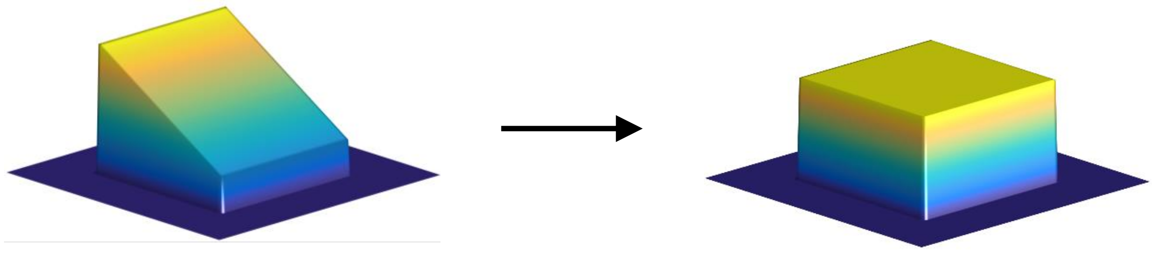
3D shape before calibration

It is plain to see that result is not as good as expected, this is why some adjustments must be applied in order to improve quality. First of all, we will decimate the shape to certain **fixed width** (720 px) and filter with a **5x5 Gaussian filter** to reduce the most significant noise:



3D shape after filtering

The result looks much better, but as it is observed there is still a linear distortion that affects equally all shape rows, caused by the unwrapping algorithm (other unwrapping algorithm could be applied in order to avoid this). It is thanks to its linear character, the distortion could be easily fixed applying next **transformation**:



Linear distortion transformation

The depth of each column is transformed using next equation:

$$\tilde{z}(x,y) = \frac{z(x,y)}{c(x)}$$

Where the linear coefficient c is:

$$c(x) = \frac{z(x,y)}{\tilde{z}(x,y)} = \alpha x + \beta$$

To get $\alpha$and β parameters, 2 points (x_1, y_1, z_1) and (x_2, y_2, z_2) have to be picked with same known depth (millimeters) from the original object and then apply the transformation to each column:

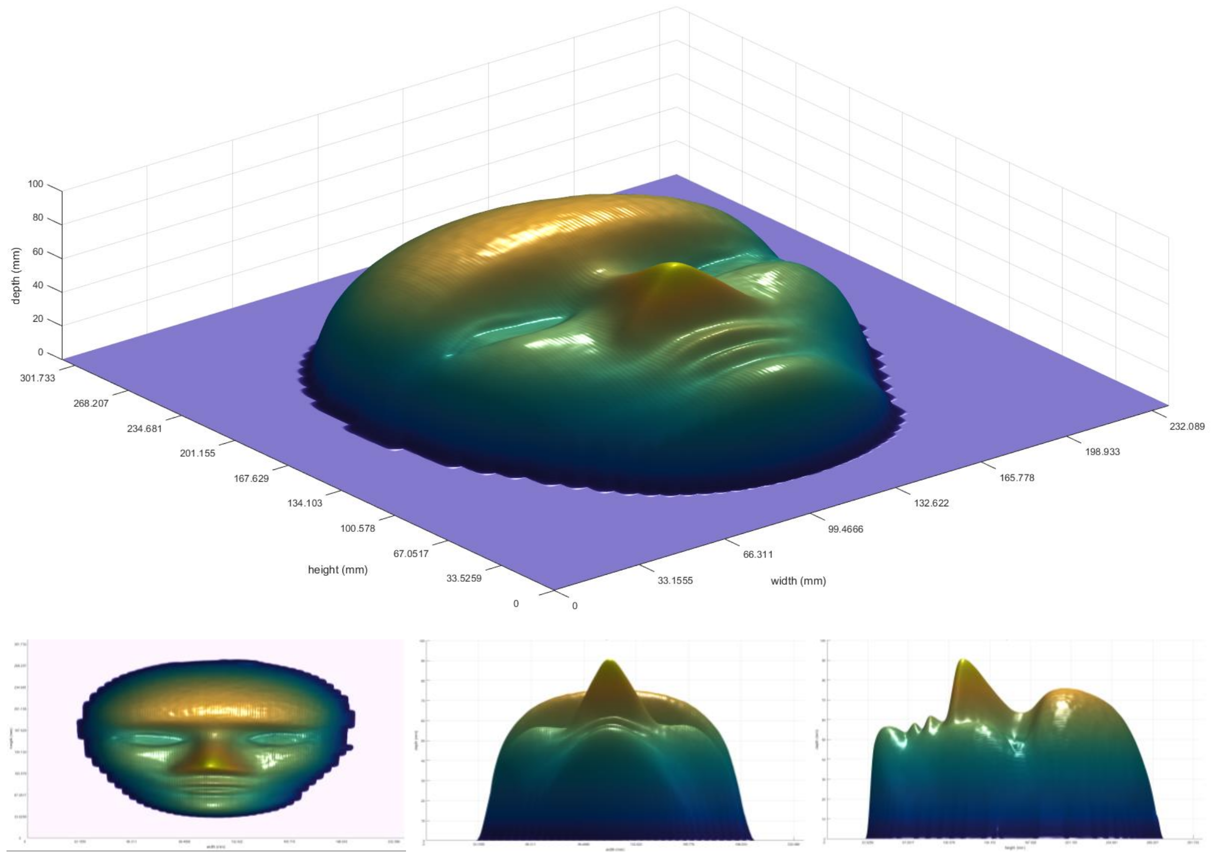
$$\alpha = \frac{1}{depth}\left(\frac{z_2 - z_1}{x_2 - x_1}\right) \qquad \beta = \frac{1}{depth}\left(\frac{z_1(x_2 - x_1) - x_1(z_2 - z_1)}{x_2 - x_1}\right)$$

Finally, a new different linear transformation must be applied to width and height axis in order to convert pixels to millimeters, which will allow to take actual measurements:

$$\tilde{x} = x \cdot \frac{actualwidth}{objectwidth} \qquad \tilde{y} = y \cdot \frac{actualheight}{objectheight}$$

In our case, I have chosen the **mask eyes** as linear transformation points p1 (232, 457, 3.379) and p2 (483, 462, 1.490) to calibrate the result. Moreover, I have measured the width, height and depth of the original mask getting 170 mm, 220 mm and 50 mm respectively.

Using these calibration points along with original dimensions we get the **3D final shape measurement**:



3D final shape measurement

5. Evaluation

In solution chapter has been shown the algorithm works properly under certain specific configuration. In this chapter it will be studied how much **different changes** in fringe pattern parameters affect to measurement quality.

Two approaches are proposed, on the on hand a **known-depth trapezoidal prism** will be used to compare the result generated by the algorithm to the expected result, and on the other hand the other hand, **LEGO pieces** will be useful to find the least depth accuracy and how well the algorithm works with different color objects.

Trapezoidal prism object dimensions are 165 mm bottom base, 100 mm top base and 40 mm depth. The better the result is the more similar the recovered object and the original object are. To measure this **similarity** between the original object O and the recovered object $\tilde{O}$, an **error function is defined** on a reference row j:

$$\varepsilon = \frac{1}{MAX_{ERROR} \cdot N_{SAMPLES}} \sum_{i=0}^{N-1} \left| O(i,j) - \tilde{O}(i,j) \right|$$

Appendix III collects all tests performed to measure the quality of the solution and error associated to each case. Test cases have been chosen using EP (equivalence partitioning) technique:

Test	Object	Intensity	Period	N patterns	Exposure	Color	Camera	Error
T01	Prism	255	10	3	Static	White	Nexus 4	3.13%
T02	Prism	150	10	3	Static	White	Nexus 4	2.62%
T03	Prism	50	10	3	Static	White	Nexus 4	3.15%
T04	Prism	10	10	3	Static	White	Nexus 4	5.51%
T05	Prism	150	5	3	Static	White	Nexus 4	3.02%
T06	Prism	150	20	3	Static	White	Nexus 4	3.59%
T07	Prism	150	30	3	Static	White	Nexus 4	5.14%
T08	Prism	150	80	3	Static	White	Nexus 4	9.64%
T09	Prism	150	10	4	Static	White	Nexus 4	1.78%
T10	Prism	150	10	5	Static	White	Nexus 4	1.47%
T11	Prism	150	10	RGB	1000 ms	White	Nexus 4	1.95%
T12	Prism	150	10	RGB	500 ms	White	Nexus 4	2.63%
T13	Prism	150	10	RGB	200 ms	White	Nexus 4	-
T14	Prism	150	10	RGB	10 ms	White	Nexus 4	-
T15	Prism	150	20	RGB	300 ms	Red	Nexus 4	5.59%
T16	Prism	150	20	RGB	300 ms	Green	Nexus 4	3.17%
T17	Prism	150	20	RGB	300 ms	Blue	Nexus 4	6.48%
T18	Prism	150	20	3	Static	White	iPhone 6	2.08%
T19	LEGO 1	150	5	3	Static	White	Nexus 4	
T20	LEGO 2	150	5	3	Static	White	Nexus 4	
T21	LEGO 3	150	5	3	Static	White	Nexus 4	

Furthermore, a **time performance analysis** will be carried out to measure the execution speed of the algorithm and detect any possible bottleneck.

5.1. Parameters analysis

Tests expose how parameters adjustment is a crucial decision and how the quality of the 3D shape measurement is directly related to aforementioned configuration.

It is noted the **optimal configuratio**n for fringe pattern is: intensity (150 ±50 unit), period (15 ±10 pixel) and white color pattern. Any other value out of these ranges causes the quality to go down to the extent that it is impossible to recover the 3D shape. Moreover, the improvement got using more than 3 patterns does not justify using a different multiplexing light technique. Besides, the quality of the smartphone camera is important but it is not crucial, it has been shown how results using Nexus 4 camera and iPhone 6 camera are very similar.

Object properties is another critical point, **best objects** to recover are any color, **matt** (no shine) and **flat** (no sharp edges) objects. The system is very accurate and it can even measure LEGO *studs* (< 5mm) with some difficulties due to LEGO pieces nature.

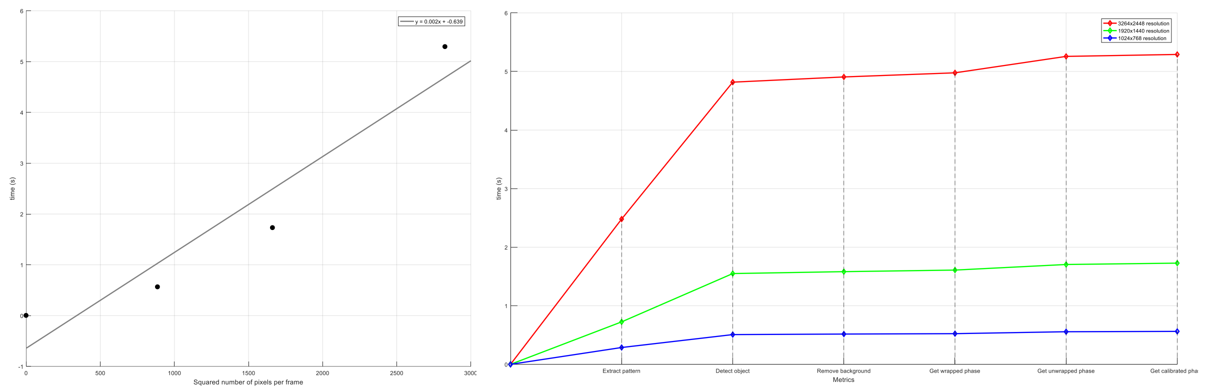
Finally, smartphone camera framerate (30 Hz for Nexus 4) restricts the maximum projecting frequency for 8 bit pattern sequence (120 Hz) in spite of the algorithm is prepared to extract patterns from videos recorded at any frequency. This could be fixed using a high-speed camera.

5.2. Time performance analysis

The time the algorithm takes to recover the 3D shape is meaningful for future real-time applications.

The total time could be separated in **different phases** (extract patterns, detect object, remove background, get wrapped phase, get unwrapped phase and get calibrated shape) and measure the time performance of each one.

For a **PC with 2.60 GHz and 8.00 GB RAM** and different resolution videos (3264x2448, 1920x1440, and 1024x768) the algorithm has taken:



Number of pixels VS time performance regression / Phases time performance

As we can see in first graph, the total amount of time needed is **directly proportional** to the number of pixels in each video frame. Under these conditions, to achieve a total time less than 1 second, the video resolution should be smaller than equivalent resolution 880x880 pixels, for example 1024x768 (4:3 aspect ratio) which would slightly reduce the quality of the measurement.

Second graph shows phases that are consuming more resources are "extract patterns phase (1)" and "detect object phase (2)", a figure that represents around 90% of the total time. Otherwise, the rest of the phases have **very good performance**.

It would be interesting to use **GPU acceleration** in order to speed up the algorithm time performance significantly, using MATLAB Parallel Computing Toolbox for example.

6. Conclusion

In this project, I have successfully designed and implemented a **high-speed 3D scanner system** using the smartphone camera and I have deeply tested its proper performance in real-world cases. The system implementation is perfectly prepared to solve some real-world problem, e.g. in the field of virtual reality, quality control, medical diagnosis, etc.

I have verified the **effectiveness** of Digital Fringe Projection to measure 3D objects and I have contributed developing new algorithm approaches for background removing and 3D shape calibration.

However, there are still some aspects for **improvement**. Synchronize sequences much faster, remove calibration plane requirement would reduce the time for patterns extraction process by half, speed up removing background step, develop better unwrapping algorithm for noisy images, and/or get full 3D shape representation and not only one object perspective.

In the future, with all these improvements the system could be applied to solve any 3D shape measurement problem in **real-time**.

References

[1] Giovanna Sansoni, Marco Trebeschi and Franco Docchio. “State-of-The-Art and Applications of 3D Imaging Sensors in Industry, Cultural Heritage, Medicine, and Criminal Investigation”, Laboratory of Optoelectronics, University of Brescia, Brescia, Italy, 2009.

[2] Thomas M. Deserno et al. “Viewpoints on Medical Image Processing: From Science to Application”, Department of Medical Informatics, Uniklinik RWTH Aachen, Germany, 2013.

[3] François Blais. “Review of 20 Years of Range Sensor Development” National Research Council Canada, Institute for Information Technology, Ottawa, Ontario, Canda.

[4] Mitsuo Takeda, Hideki Ina and Seiji Kobayashi. “Fourier-transform method of fringe-pattern analysis for computer-based topography and interferometry”, University of Electrocommunications, Tokyo, Japan, 1981.

[5] Maria I. Todorovska. “Estimation of instantaneous frequency of signals using the continuous wavelet transform”, University of Southern California, Los Angeles, California, United States, 2004.

[6] Picinbono, Bernard. “On Instantaneous Amplitude and Phase of Signals”, IEEE Transaction on signal processing, Vol. 45, No. 3, IEEE, 1997.

[7] Song Zhang. “High-resolution, Real-time 3-D Shape Measurement”. Stony Brook University, New York, United States, 2005.

[8] Song Zhang. “High-Speed 3D Imaging with Digital Fringe Projection Techniques”, ed. CRC Press, p. 25-26, 2016.

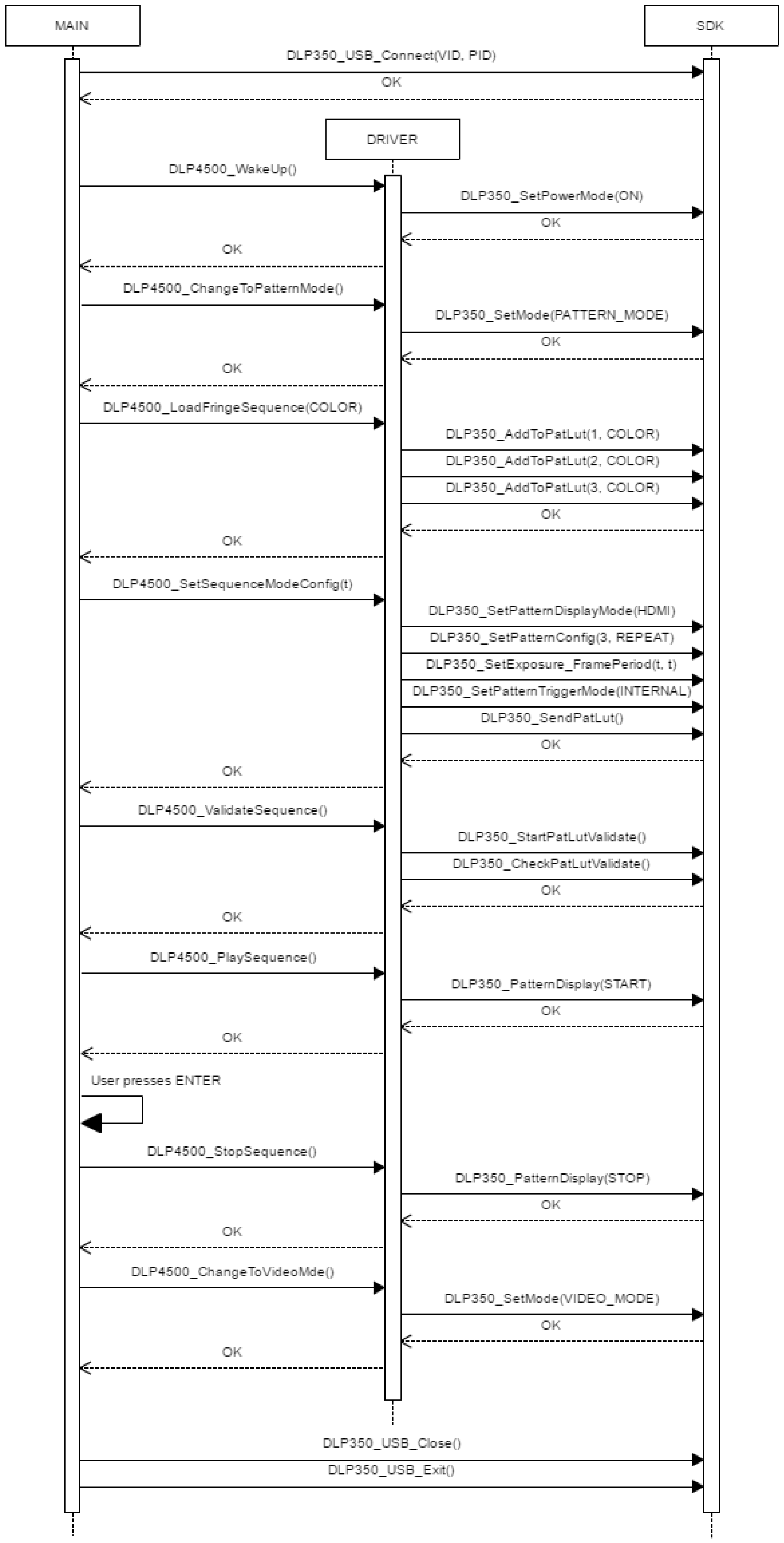
[9] Dennis C. Ghiglia, and Mark D. Pritt. “Two-Dimensional Phase Unwrapping: Theory, Algorithms, and Software”, John Wiley & Sons, 1998

[10] *DLP® LightCrafter 4500™ Evaluation Module User's Guide*, 2nd ed., Texas Instruments Co., Texas, 2015, pp. 40-43.

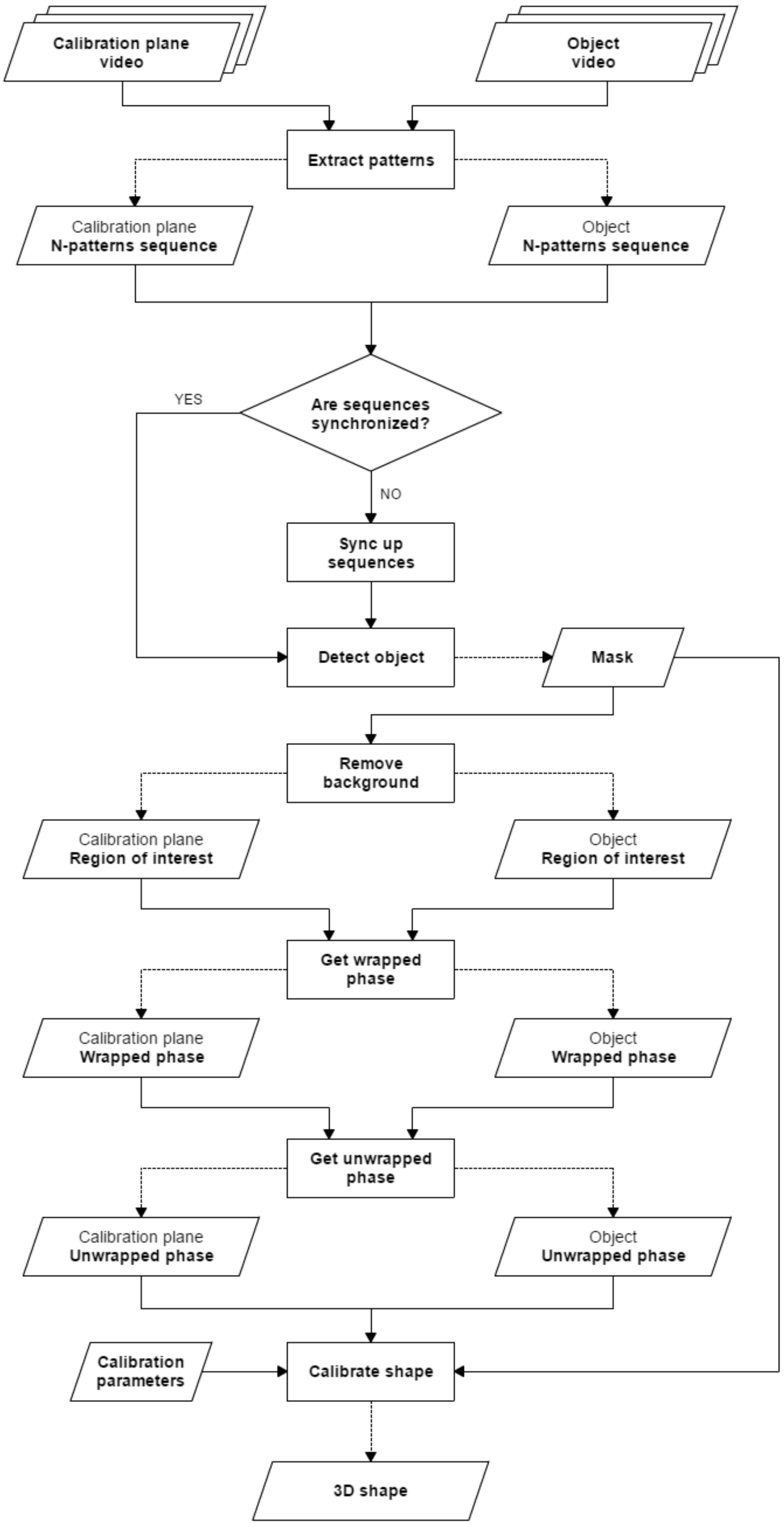
[11] Frankowski, G. et al. “DLP-Based 3D Metrology by Structured Light or Projected Fringe Technology for Life Sciences and Industrial Metrology”, GFMesstechnik GmbH Co., 2015, pp. 3.

[12] Alvaro Bayona, Juan Carlos San Miguel and José M. Martínez. “Comparative evaluation of stationary foreground object detection algorithms based on background subtraction techniques”, Universidad Autonoma de Madrid, Madrid, Spain, 2009.

Appendix 1 – DLP 4500 driver UML sequence diagram

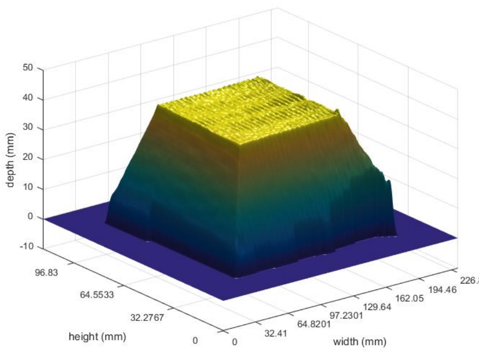

MAIN
SDK
DLP350_USB_Connect(VID, PID)
OK
DRIVER
DLP4500_WakeUp()
DLP350_SetPowerMode(ON)
OK
OK
DLP4500_ChangeToPatternMode()
DLP350_SetMode(PATTERN_MODE)
OK
OK
DLP4500_LoadFringeSequence(COLOR)
DLP350_AddToPatLut(1, COLOR)
DLP350_AddToPatLut(2, COLOR)
DLP350_AddToPatLut(3, COLOR)
OK
OK
DLP4500_SetSequenceModeConfig(t)
DLP350_SetPatternDisplayMode(HDMI)
DLP350_SetPatternConfig(3, REPEAT)
DLP350_SetExposure_FramePeriod(t, t)
DLP350_SetPatternTriggerMode(INTERNAL)
DLP350_SendPatLut()
OK
OK
DLP4500_ValidateSequence()
DLP350_StartPatLutValidate()
DLP350_CheckPatLutValidate()
OK
OK
DLP4500_PlaySequence()
DLP350_PatternDisplay(START)
OK
OK
User presses ENTER
DLP4500_StopSequence()
DLP350_PatternDisplay(STOP)
OK
OK
DLP4500_ChangeToVideoMde()
DLP350_SetMode(VIDEO_MODE)
OK
OK
DLP350_USB_Close()
DLP350_USB_Exit()

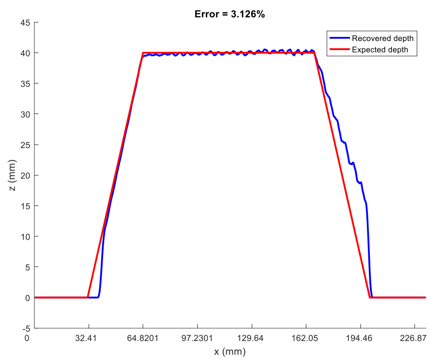
Appendix 2 – DFP Algorithm flow diagram



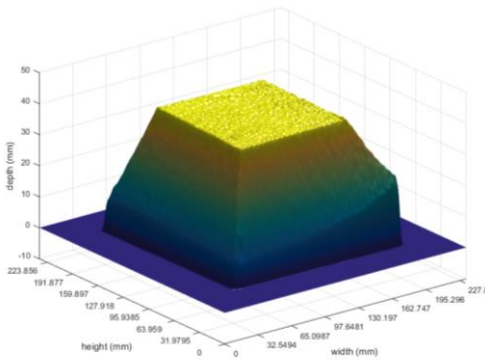
Appendix 3 – Evaluation tests

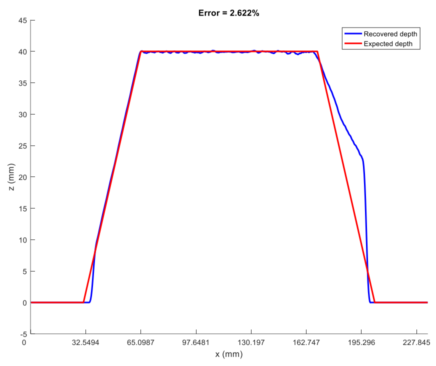
1. Test 01 (255 intensity, 10 px period, 3 patterns, static, white, Nexus 4)



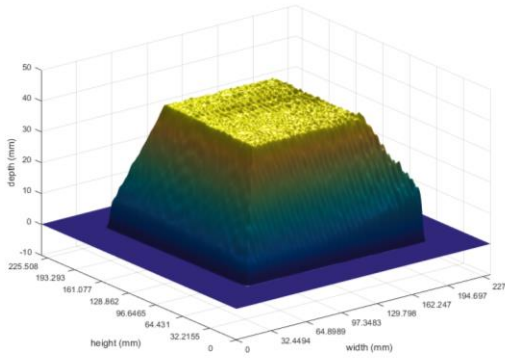


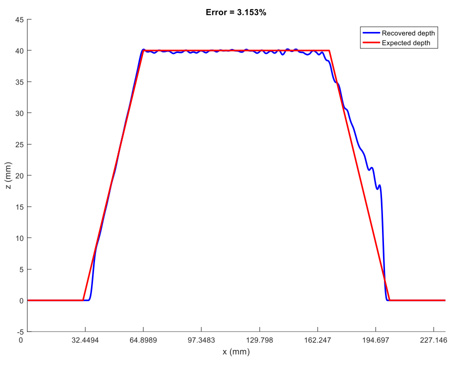
2. Test 02 (150 intensity, 10 px period, 3 patterns, static, white, Nexus 4)



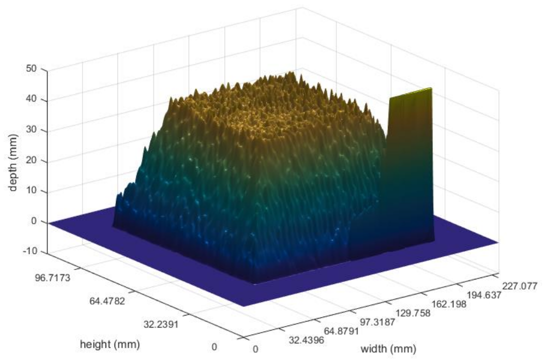


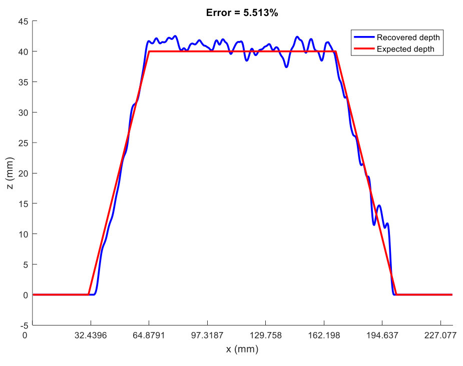
3. Test 03 (50 intensity, 10 px period, 3 patterns, static, white, Nexus 4)



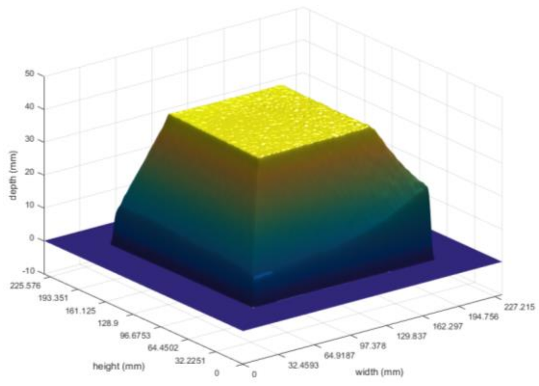


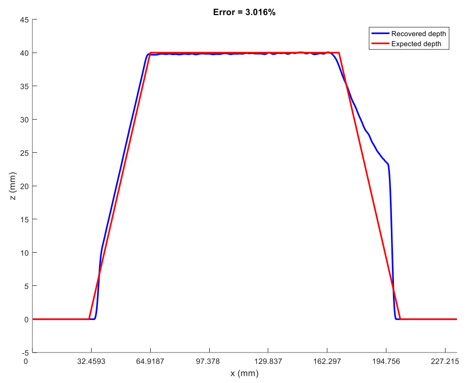
4. Test 04 (10 intensity, 10 px period, 3 patterns, static, white, Nexus 4)



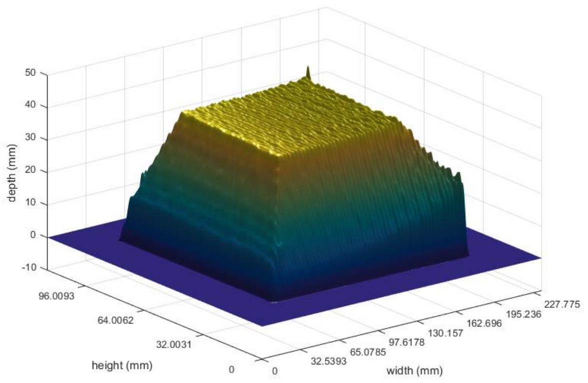


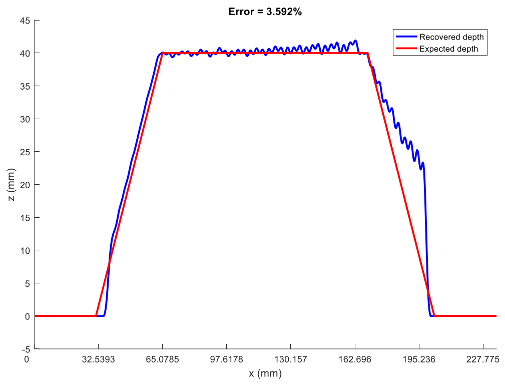
5. Test 05 (150 intensity, 5 px period, 3 patterns, static, white, Nexus 4)



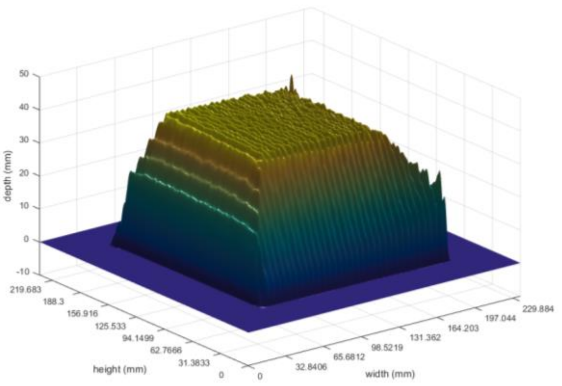


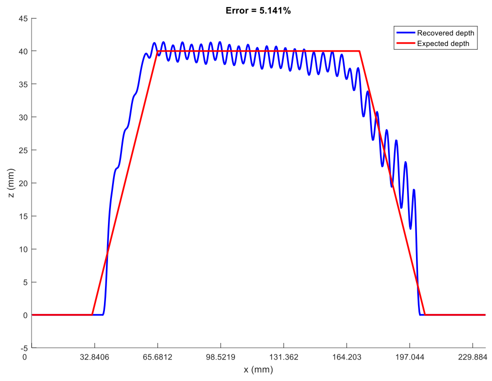
6. Test 06 (150 intensity, 20 px period, 3 patterns, static, white, Nexus 4)



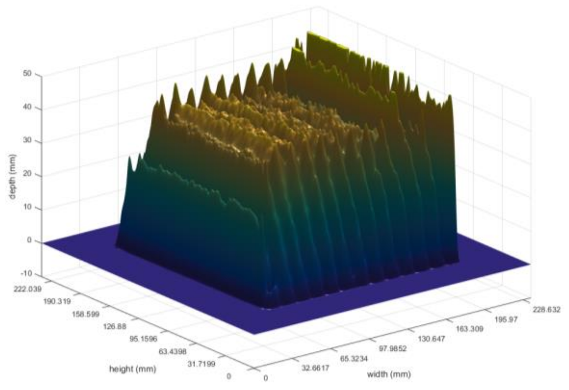


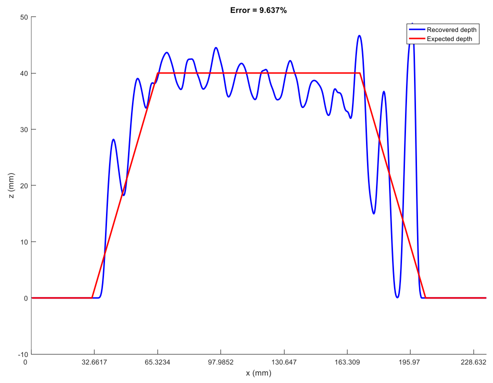
7. Test 07 (150 intensity, 30 px period, 3 patterns, static, white, Nexus 4)



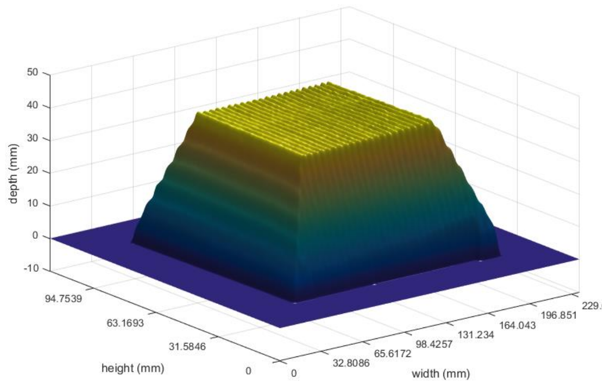


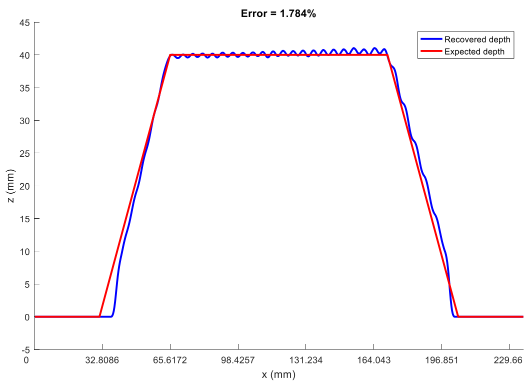
8. Test 08 (150 intensity, 80 px period, 3 patterns, static, white, Nexus 4)



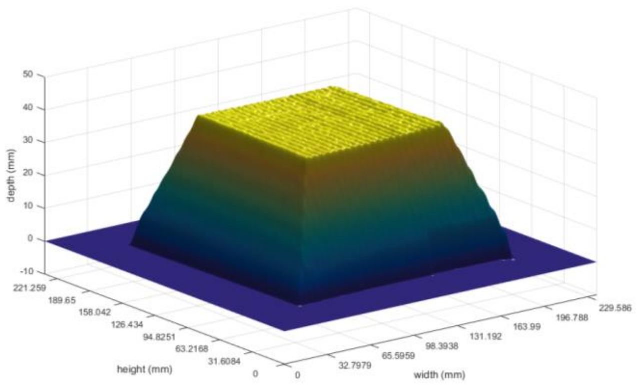


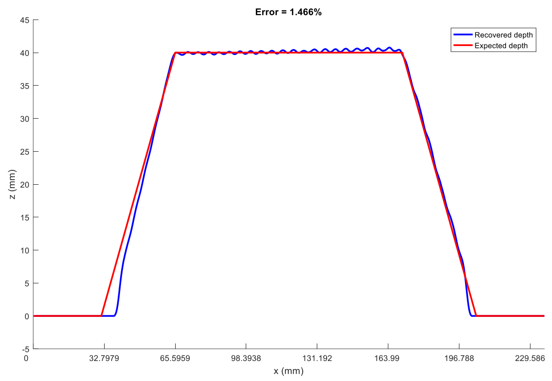
9. Test 09 (150 intensity, 10 px period, 4 patterns, static, white, Nexus 4)



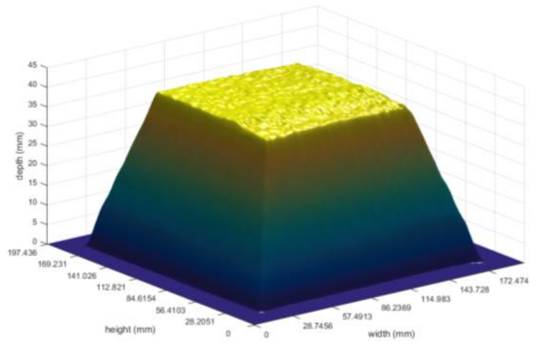


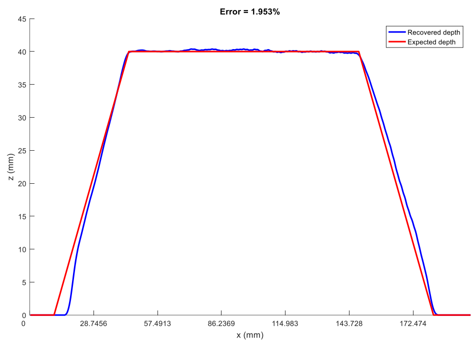
10. Test 10 (150 intensity, 10 px period, 5 patterns, static, white, Nexus 4)



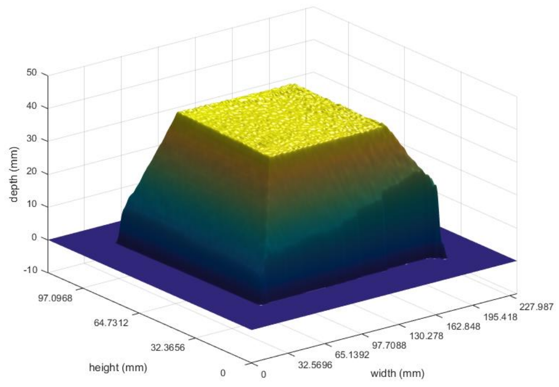


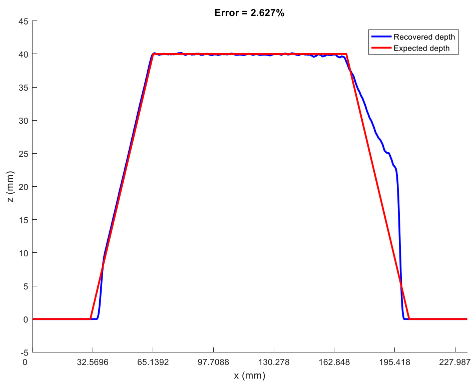
11. Test 11 (150 intensity, 10 px period, RGB, 1000 ms, white, Nexus 4)





12. Test 11 (150 intensity, 10 px period, RGB, 500 ms, white, Nexus 4)





13. Test 13 (150 intensity, 10 px period, RGB, 200 ms, white, Nexus 4)

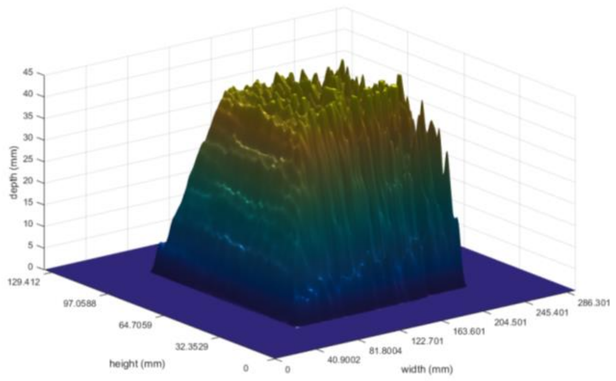
Even though the camera frequency (30 Hz) is higher than the projection frequency (5 Hz) and it is possible to extract patterns individually, aliasing makes impossible to recover the shape from the video sequence.

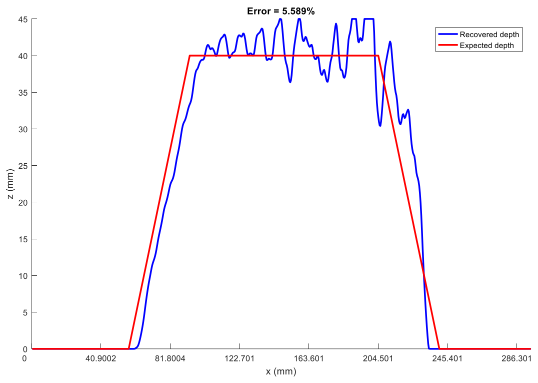
An optical low-pass filter could be added to the camera to reduce aliasing.

14. Test 14 (150 intensity, 10 px period, RGB, 10 ms, white, Nexus 4)

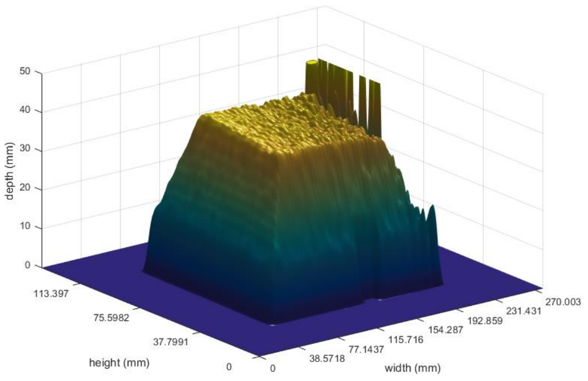
Owing to camera frequency (30 Hz) is smaller than projection frequency (100 Hz) it is impossible to extract the patterns from the video sequence and therefore, the shape cannot be recovered.

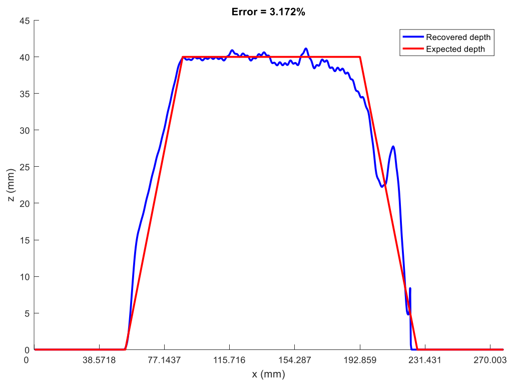
15. Test 15 (150 intensity, 20 px period, RGB, 10 ms, red, Nexus 4)



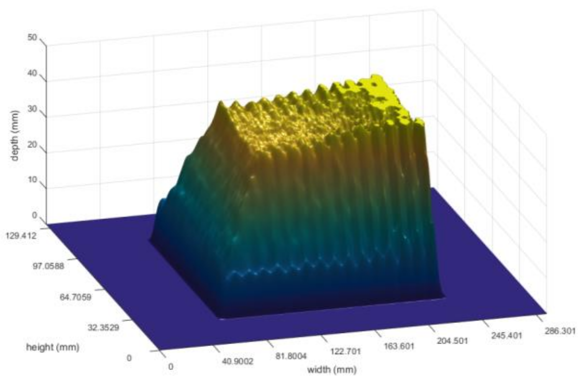


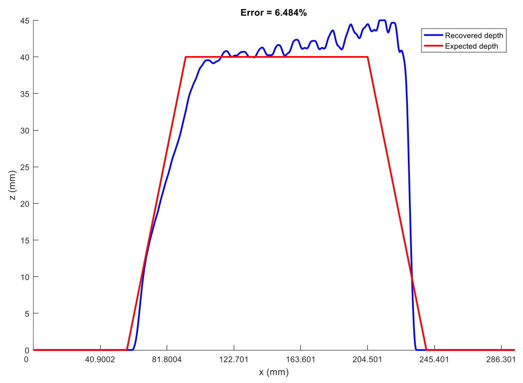
16. Test 16 (150 intensity, 20 px period, RGB, 10 ms, green, Nexus 4)



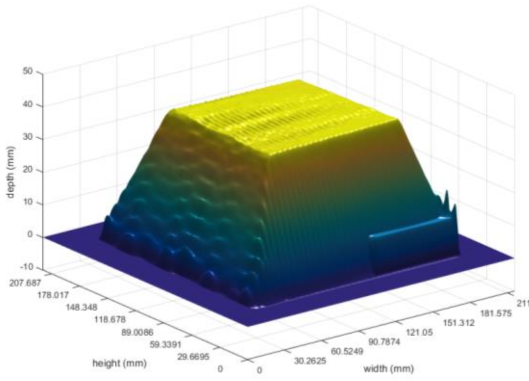


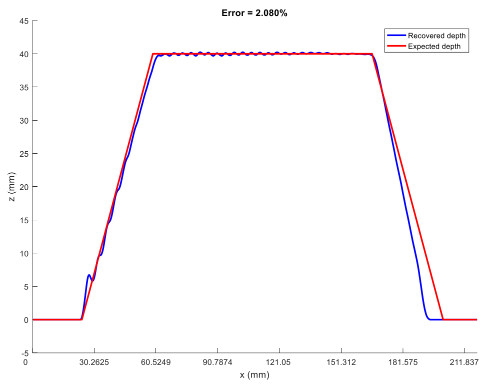
17. Test 17 (150 intensity, 20 px period, RGB, 10 ms, blue, Nexus 4)



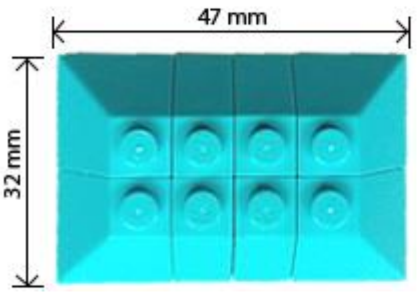


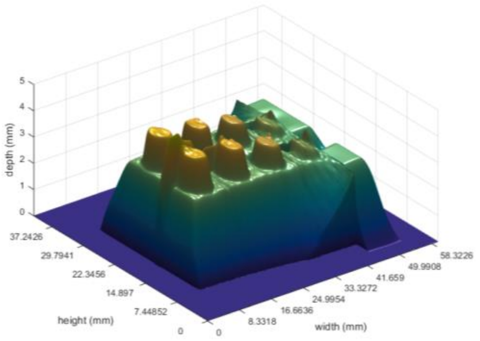
18. Test 18 (150 intensity, 20 px period, 3 patterns, static, white, iPhone 6)



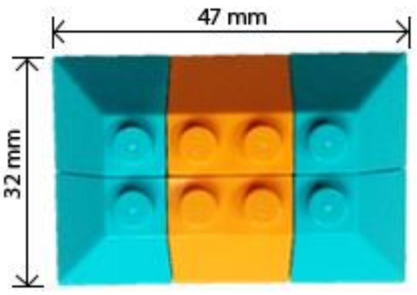


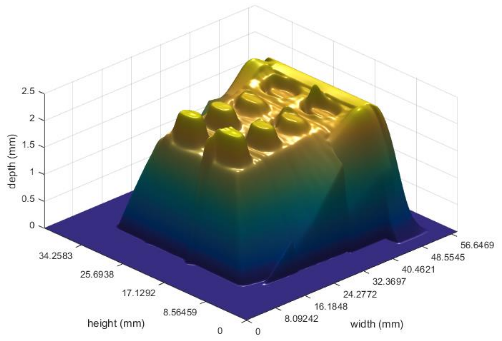
19. Test 19 (LEGO piece, same color, 10 mm height, 5 px period)



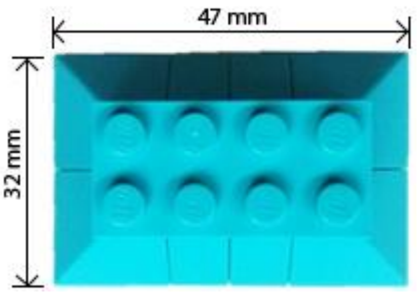


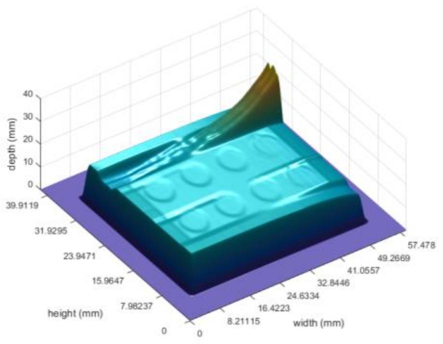
20. Test 20 (LEGO piece, different color, 10 mm height, 5 px period)





21. Test 21 (LEGO piece, same color, 20 mm height, 5 px period)





The shadow generated over the base makes impossible for the algorithm to recover the shape. It is **essential** to project light in a way no shadows appear.