ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: DESARROLLO DE UN SISTEMA DE ESCÁNER 3D DE ALTA VELOCIDAD MEDIANTE LA PROYECCIÓN DIGITAL DE LUZ ESTRUCTURADA.

AUTOR: SAMUEL VELOSO LÓPEZ.

TITULACIÓN: GRADO EN INGENIERÍA TELEMÁTICA.

TUTOR: BRYAN HENNELLY.

UNIVERSIDAD: MAYNOOTH UNIVERSITY.
CENTRO: COMPUTER SCIENCE DEPARTMENT.
PAÍS: IRLANDA.

Fecha de lectura: 10/07/2017

Calificación:

El Coordinador de Movilidad,
Development of a high-speed 3D scanner system using Digital Fringe Projection techniques

Samuel Veloso López
16186737

Final Year Project – 2017
B.Sc. in Computer Science and Software Engineering

Department of Computer Science
Maynooth University
Maynooth, Co. Kildare
Ireland

A thesis submitted in partial fulfilment of the requirements for the B.Sc. in Computer Science and Software Engineering

Supervisor
Dr. Bryan Hennelly
# Table of contents

**Declaration** ................................................................................................................................. 1  
**Acknowledgments** ...................................................................................................................... 2  
**Abstract** .......................................................................................................................................... 3  

## 1. Introduction .................................................................................................................................. 4  
1.1. Motivation ................................................................................................................................... 4  
1.2. Problem statement ....................................................................................................................... 4  
1.3. Approach ..................................................................................................................................... 4  
1.4. Development ............................................................................................................................... 4  

## 2. Technical Background .............................................................................................................. 5  
2.1. DFP: Digital Fringe Projection .................................................................................................... 5  
2.2. DLP: Digital Light Processing .................................................................................................... 5  
2.3. Linux framebuffer ....................................................................................................................... 5  
2.4. GCC compilation in Linux .......................................................................................................... 6  
2.5. SSIM image quality index ........................................................................................................... 6  

## 3. The Problem ................................................................................................................................ 7  
3.1. Digital fringe projection approach ............................................................................................. 7  
3.2. Hilbert Transform method .......................................................................................................... 7  
3.3. Song Zhang phase-shifting technique ........................................................................................ 8  
3.4. Unwrapping phase map ............................................................................................................ 8  
3.5. 3D Shape measurement ............................................................................................................ 9  

## 4. The Solution .................................................................................................................................. 10  
4.1. Generate fringe patterns ............................................................................................................ 11  
4.2. High-speed mode ....................................................................................................................... 12  
4.3. Record video ............................................................................................................................. 13  
4.4. Extract fringe patterns .............................................................................................................. 14  
4.5. DFP Algorithm ......................................................................................................................... 15  
  4.5.1. Detect object and remove background .................................................................................. 16  
  4.5.2. Get wrapped phase .............................................................................................................. 17  
  4.5.3. Get unwrapped phase ......................................................................................................... 18  
  4.5.4. Calibrate 3D shape ............................................................................................................. 19  

## 5. Evaluation .................................................................................................................................... 21  
5.1. Parameters analysis .................................................................................................................... 22  
5.2. Time performance analysis ....................................................................................................... 22  

## 6. Budget ........................................................................................................................................... 23  

## 7. Conclusion .................................................................................................................................... 23  

**References** ...................................................................................................................................... 24
Declaration

I hereby certify that this material, which I now submit for assessment on the program of study as part of B.Sc. in Computer Science and Software Engineering qualification, is entirely my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

I hereby acknowledge and accept that this thesis may be distributed to future final year students, as an example of the standard expected of final year projects.

Signed: Samuel Veloso López  Date: 4 March 2017
To my parents, Lourdes López Espinazo and Gerardo Veloso Alonso, who have been always there and showed me that there is nothing better than a job well done.
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 are best conditions to reconstruct the shape 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, this is 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]. This project will cover structured light to get 3D shape measurement from a video, recorded with the smartphone camera, projecting a moving fringe pattern onto the object with a DLP projector.

1.1. Motivation

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.

I will contribute to this field using a DLP LightCrafter 4500 projector module to project a sequence of moving fringes onto an object, record these images using a smartphone, and then applying image processing algorithms to recover the 3D shape of the object using DFP (Digital Fringe Projection) techniques.

1.2. Problem statement

The problem could be divided into three different parts: 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.

1.3. Approach

I started with a theoretical approach to understand how digital fringe projection works, then I designed and implemented a real-world system and after many hours of trial and error I was able to reconstruct the 3D shape as detailed in problem statement. For the evaluation I have focused on measuring the quality of the results compared to real object measurement in order to find best system configuration. A time performance analysis has been also carried out to allow real-time for future extension.

1.4. Development

Significant achievements in the project are:

1. Developed C++ application for Raspberry Pi to generate fringe and binary patterns, and to multiplex pattern sequences using DLP 4500 SDK.
2. Extracted each fringe pattern from the video synchronizing camera and projector frequencies.
3. Detected and removed the object from the background.
4. Achieved 3D shape measurement.

My solution uses the smartphone camera where traditionally it has been using a high-speed camera. I have developed my own algorithms for object detection and shape calibration.
2. Technical Background

In this chapter I will outline some of the topics covered in this report, divided into two categories: **topic material** (Digital Fringe Projection) and **technical material** (Digital Light Processing, Linux framebuffer, GCC compilation in Linux, HID protocol and SSIM image quality index).

2.1. DFP: Digital Fringe Projection

XYZT Lab research group ([http://www.xyztlab.com/](http://www.xyztlab.com/)), led by Song Zhang is responsible for most of the major advances in digital fringe projection over the last 10 years. They created **first-ever** real-time 3D imaging system [4] in 2006 and many other 3D shape measurement techniques:

This work will be the base and the starting point for the implementation of my 3D high-speed smartphone camera system and it will be explained deeply in chapter III.

2.2. DLP: Digital Light Processing

Digital Light Processing is a **revolutionary** way to project images, based on the DMD micromirroring device by Texas Instruments [6]. It is revolutionary because its working principle is completely different to existing technologies (LED, LCD and LCoS) and it is able to project images at **much faster** rate than its competitors. The heart of the system is the DMD (Digital Micromirror Device) which has on its surface thousands of microscopic mirrors arranged in a rectangular array, matching the pixels of the image to be projected:

Each micromirror is toggled on and off at incredible speed (~4000 Hz) allowing or denying the pass of light. Sending to these mirrors light from RGB LEDs any full-color image could be displayed [7].

2.3. Linux framebuffer

Framebuffer is the lowest-level driver that Linux provides to work with graphics as applications do not have to worry about the hardware implementation. In Linux all devices are files, this is framebuffer is handled writing and reading from /dev/fb0 as any other file [8].
When RGB, HDMI or any other output video port wants to stream data, it takes data stored in **framebuffer**. This data is stored depending on the video mode configured (8 bit, 16 bit, 24 bit or 32 bit). The block size is 8 bit, therefore the number of channels per each mode is given by:

\[
\text{channels} = \log_8(\text{mode})
\]

Thus, to store an image pixel in framebuffer, it is copied in next framebuffer position:

\[
\text{buffer}[\text{channels} \cdot (x + y \cdot \text{width})] = \text{image}(x, y)
\]

For Raspberry Pi the process is the same than in any other Linux system. Open framebuffer, change video mode to match image mode, map buffer into user memory and copy image to memory map [9].

### 2.4. GCC compilation in Linux

The best and easiest way to compile a C/C++ program with GCC compiler in Linux is using a **Makefile**. Makefile, as the name already indicates, is a script file placed in project root folder to compile all sources files [10]. It may have next information:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Name of the program</td>
</tr>
<tr>
<td>SRC</td>
<td>Source files (.c or .cpp)</td>
</tr>
<tr>
<td>OBJ</td>
<td>Object files (.o)</td>
</tr>
<tr>
<td>CXX</td>
<td>Compiler (gcc or g++)</td>
</tr>
<tr>
<td>CXXFLAGS</td>
<td>Compilation flags (-std, -g, -Wall, etc)</td>
</tr>
<tr>
<td>LDFLAGS</td>
<td>Linking flags (-I, -L, -lm, etc)</td>
</tr>
<tr>
<td>INCLUDE_DIR</td>
<td>Headers installation path (optional)</td>
</tr>
<tr>
<td>LIBS_DIR</td>
<td>Library installation path (optional)</td>
</tr>
</tbody>
</table>

When “make” is typed in command line, next statement is executed to create object (.o) and binary files (.SNAME) as result:

```bash
all: $(NAME)
$(NAME): $(OBJ)
    $(CXX) $(CXXFLAGS) $(LDFLAGS) -o $(NAME) $(OBJ) $(LDLIBS)
```

For installing shared libraries, it is possible to run “make install” to copy library headers to INCLUDE_DIR and library to LIBDS_DIR. For uninstalling a library the command is “make uninstall” and will remove files copied by install command.

### 2.5. SSIM image quality index

To measure how similar 2 images are, best metric is SSIM (structural similarity) over PSNR (peak signal-to-noise ratio) and MSE (mean squares error) [11].

The similarity between image \(x\) and \(y\) is given by next formula:

\[
\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{\left(\mu_x^2 + \mu_y^2 + c_1\right)\left(\sigma_x^2 + \sigma_y^2 + c_2\right)}
\]

Where \(\mu\) is the average of each image, \(\sigma\) is the variance of each image, \(\sigma_{xy}\) is the covariance between images and \(c_1\) and \(c_2\) are constants to stabilize division with weak dominator. The output is in range [-1, +1] where +1 means images are identical and -1 means they are completely different.
3. The Problem

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 [12], Wavelet [13] or Hilbert [14] transforms are traditional methods to solve this problem but for 3D shape measurement are not accurate enough. Song Zhang [15] 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:

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

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 [15], to subsequently compare both results.

3.2. Hilbert Transform method

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

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

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

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

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

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

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:

\[ I_s(x, y, \phi_s) = I_0 + I \cdot \cos(\phi(x, T, \phi_0) + \phi_s) \]

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

\[
\begin{align*}
I_1 &= I_0 + I \cdot \cos\left(\phi - \frac{2\pi}{3}\right) = I_0 + I \cos(\phi) \cos\left(\frac{2\pi}{3}\right) + I \sin(\phi) \sin\left(\frac{2\pi}{3}\right) = I_0 - \frac{I}{2} \cos(\phi) + \frac{I\sqrt{3}}{2} \sin(\phi) \\
I_2 &= I_0 + I \cdot \cos(\phi) \\
I_3 &= I_0 + I \cdot \cos\left(\phi + \frac{2\pi}{3}\right) = I_0 + I \cos(\phi) \cos\left(\frac{2\pi}{3}\right) - I \sin(\phi) \sin\left(\frac{2\pi}{3}\right) = I_0 + \frac{I}{2} \cos(\phi) - \frac{I\sqrt{3}}{2} \sin(\phi)
\end{align*}
\]

To find the instantaneous phase we have to isolate it:

\[ I_1 - I_3 = \sqrt{3}I\sin(\phi) \]

\[ 2I_2 - I_1 - I_3 = 2I\cos(\phi) + I\cos(\phi) = 3I \cdot \cos(\phi) \]

The tangent of the phase relates both equations:

\[ \frac{I_1 - I_3}{2I_2 - I_1 - I_3} = \frac{\sqrt{3}I\sin(\phi)}{3I \cdot \cos(\phi)} = \frac{\sqrt{3}}{3} \tan(\phi) \]

Finally:

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

In order to improve the accuracy of the system, we could add redundancy using more than 3 patterns. Though, Carré and Hariharan [16] proposes a formula to tolerate large phase shift error using 4 or 5 fringe patterns respectively:

\[ \phi(x, y) = \tan^{-1}\left(\frac{\sqrt{3}(I_2 - I_3) + I_4 - I_1}{I_2 + I_3 - I_1 - I_4}\right) \]

\[ \text{Carré formula} \]

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

\[ \text{Hariharan formula} \]

### 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 [17] must be applied in order to correct it:

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

\[ \text{where } \Delta\phi(x) = \phi(x) - \phi(x - 1) \]
3.5. 3D Shape measurement

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

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

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:

\[ \Omega_{depth} = \frac{d(\Phi_{object} - \Phi_{plane})}{dx} \]

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:

![Subsystem diagram]

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**.
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 [18]. 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.

With **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](image)

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 $\delta$:

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

$I$, $T$ and $\phi$ 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 $\phi$ 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:

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

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 [19]. DLP LightCrafter 4500 is able to multiplex up to \textbf{4225 Hz/230 \( \mu \text{s} \)} (1-bit color depth) and up to \textbf{120 Hz/8333 \( \mu \text{s} \)} (8-bit color depth) pattern rate.

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

The way in which the Raspberry Pi communicates with the LightCrafter was a huge challenge. Fortunately the LightCrafter GUI is \textbf{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 \textbf{USB HID protocol} to communicate with LightCrafter (VID 0x045, PID 0x6401) using 64-bit data blobs. For that, it make use of \textbf{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 \texttt{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 \texttt{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 \textbf{all SDK functions} (USB_Connect, GetFirmwareVersion, SetLedEnables, etc.) in pGenerator and take control of the LightCrafter, simply adding the library -\texttt{ldlp4500sdk} to the Makefile.

Looking deeply through the original GUI source code, I could find the way to implement RGB \textbf{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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLP4500_WakeUp</td>
<td>Wake up the projector if it is in standby mode.</td>
</tr>
<tr>
<td>DLP4500_ChangeToVideoMode</td>
<td>Change current mode to video mode.</td>
</tr>
<tr>
<td>DLP4500_ChangeToPatternMode</td>
<td>Change current mode to pattern sequence mode.</td>
</tr>
<tr>
<td>DLP4500_LoadFringeSequence</td>
<td>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.</td>
</tr>
<tr>
<td>DLP4500_SetSequenceModeConfig</td>
<td>Set pattern exposure period t in milliseconds and HDMI as input video interface.</td>
</tr>
<tr>
<td>DLP4500.ValidateSequence</td>
<td>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).</td>
</tr>
<tr>
<td>DLP4500_PlaySequence</td>
<td>Start playing the pattern sequence</td>
</tr>
<tr>
<td>DLP4500_StopSequence</td>
<td>Stop the pattern sequence</td>
</tr>
</tbody>
</table>

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](image)

### 4.3. Record video

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

\[
f_{\text{camera}} \geq \frac{1}{\text{exposure} \text{period}}
\]

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:

\[
\begin{align*}
v_c[n], n \in 0, ..., N_1 \\
v_o[n], n \in 0, ..., N_2
\end{align*}
\]

Where \(N_1 \land N_2\) are the number of frames in calibration-plane \& 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:

\[
\begin{align*}
C_i^{(j)} &= v_c \left[ \frac{f_c}{f_p} (i + j \cdot M - 1) + m_0 \right], \quad i \in 1, ..., M, \quad j \in 0, ..., \left\lceil \frac{N_1 \cdot f_p}{M \cdot f_c} - 1 \right\rceil \\
O_i^{(j)} &= v_o \left[ \frac{f_c}{f_p} (i + j \cdot M - 1) + m_0 \right], \quad i \in 1, ..., M, \quad j \in 0, ..., \left\lceil \frac{N_2 \cdot f_p}{M \cdot f_c} - 1 \right\rceil
\end{align*}
\]

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.

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

Patterns extraction diagram \((f_c = 30Hz \land f_p = 10Hz)\)

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 \(j\):

\[
O_i^{(j)} = \text{argmax} \left( \text{SSIM} \left( C_i^{(j)}, \{ O_i^{(j)}, ..., O_M^{(j)} \} \right) \right)
\]

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:

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:

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 [20] to remove the 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](image1)

If we get the image as a result 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](image2)

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](image3)
If the object casts a **shadow** on the background it must be removed because it will look like some 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](image)

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](image)

### 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:

\[
\varphi(x, y) = \tan^{-1} \left( \frac{\sqrt{3}(l_1 - l_3)}{2l_2 - l_1 - l_3} \right)
\]

**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 the calibration plane and the 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:

$$\varphi_{\text{reference}}(y) = \text{unwrap}(\varphi(x_{\text{centroid}}, y))$$

for \( j \in \text{all rows} \rightarrow \Phi(x,j) := \Phi(x,j) - \Phi(k,j) + \varphi_{\text{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:

\[ z(x, y) = \Phi_{\text{cplane}}(x, y) - \Phi_{\text{object}}(x, y) \]

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:

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:
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 $\beta$ 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:

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

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 real measurements:

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

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**:
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 one 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 $j$:

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

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:

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

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 configuration 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:

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. Budget

The total cost for setting up this project is **3550 euro**:

<table>
<thead>
<tr>
<th>Item</th>
<th>Units</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLP LightCrafter 4500</td>
<td>1</td>
<td>1300 €</td>
</tr>
<tr>
<td>Raspberry Pi Model 3 B</td>
<td>1</td>
<td>50 €</td>
</tr>
<tr>
<td>MATLAB R2017a License</td>
<td>1</td>
<td>2000 €</td>
</tr>
<tr>
<td>Smartphone Camera</td>
<td>1</td>
<td>200 €</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td></td>
<td><strong>3550 €</strong></td>
</tr>
</tbody>
</table>

7. 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


Appendix I – DLP 4500 driver UML sequence diagram
Appendix III – 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.