Pontifícia Universidade Católica
do Rio de Janeiro

Matheus Esteves Ferreira

Study and implementation of a single pixel camera by compressive sampling

Dissertação de Mestrado

Thesis presented to the Programa de Pós-graduação em Engenharia Elétrica da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica.

Advisor: Prof. Guilherme Penello Temporão

Rio de Janeiro
March 2021

Pontifícia Universidade Católica
do Rio de Janeiro

Matheus Esteves Ferreira

Study and implementation of a single pixel camera by compressive sampling

Thesis presented to the Programa de Pós-graduação em Engenharia Elétrica da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica. Approved by the Examination Committee.

Prof. Guilherme Penello Temporão
Advisor
Departamento de Engenharia Elétrica – PUC-Rio

Prof. Stephen Patrick Walborn
Universidad de Concepción – UDEC

Prof. Raul Queiroz Feitosa
Departamento de Engenharia Elétrica – PUC-Rio

Rio de Janeiro, March 30th, 2021

Matheus Esteves Ferreira

Graduated in Nanotechnology by the Federal University of Rio de Janeiro.

Bibliographic data

Ferreira,Matheus Esteves

Study and implementation of a single pixel camera by compressive sampling / Matheus Esteves Ferreira; advisor: Guilherme Penello Temporão. – Rio de janeiro: PUC-Rio , Departamento de Engenharia Elétrica, 2021.

v., 88 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica.

Inclui bibliografia

1. Engenharia Elétrica – Teses. 2. Câmeras de Píxel Único;. 3. Sensoriamento Compressivo;. 4. Ghost Imaging;. 5. Imageamento Computacional.. I. Temporão, Guilherme. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. III. Título.

CDD: 621.3

To my friends and family, for their support
and encouragement.

Acknowledgments

This project was only made possible by the people that were always there to support me.

First, I would like to thank Prof. Temporão for his guidance, support, patience and the great work environment he offered me. Temporão gave me the freedom to pursue a research theme that differed from what had been done in the lab before, whilst providing a safety net when I needed the extra support and know how. This freedom gave me room to experiment and grow as a researcher, challenging me to try new things and leave my comfort zone.

Second, I would like to thank my teammates at IBM Research. Without your support, encouragement and understanding, it wouldn't have been possible.

To all my friends and colleagues at the photonics group, thank you for always being ready to help. From helping me understand how to program the instruments, to finding a specific lens I needed, you were always there to help. This was a team effort.

To my friends and family, I'll always be grateful for all your support. Thanks for all the late-night conversations, for understating when I had to work until late to finish all the experiments, for cutting me some slack when I was down. Even during a global pandemic, when the circumstances weren't great, you were there to make sure I had what I needed to carry on.

I would also like to thank Prof. Stephen Walborn and the Quantum Optics laboratory from UFRJ that lent the SLM that we used in this work.

Finally, and definitely not the least, to CNPq, CAPES and PUC-Rio, for the aids granted, without which this work does not could have been accomplished.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001

Abstract

Ferreira,Matheus Esteves; Temporão, Guilherme (Advisor). **Study and implementation of a single pixel camera by compressive sampling**. Rio de Janeiro, 2021. 88p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Single-pixel imaging consists in computationally reconstructing 2-dimensional images from a set of intensity measurements taken by a single-point detector. To derive the spatial information of a scene, a set of modulation patterns are applied to the transmitted/backscattered light from the object and combined with the integral signal on the detector. First, we present an overview of such optical systems and implement a proof of concept that can perform image acquisition using three different modes of operation: Raster scanning, Hadamard basis scanning, and Hadamard compressive sampling. Second, we explore how the different experimental parameters affect image acquisition. Finally, we compare how the three scanning mode perform for acquisition of images of sizes ranging from (8px, 8px) to (128px, 128px).

Keywords

Single Pixel Cameras; Compressive Sampling; Ghost Imaging; Computational Imaging.

Resumo

Ferreira,Matheus Esteves; Temporão, Guilherme. **Estudo e implementação de uma câmera de pixel único por meio de sensoriamento compressivo**. Rio de Janeiro, 2021. 88p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Câmeras de pixel único consistem em reconstruir computacionalmente imagens em duas dimensões a partir de um conjunto de medidas feitas por um detector de único pixel. Para que se obtenha a informação espacial, um conjunto de padrões de modulação são aplicados à luz transmitida/refletida do objeto e essa informação é combinada com o sinal integral do detector. Primeiro, apresentamos uma visão geral desses sistemas e demonstramos a implementação de uma prova de conceito capaz de fazer aquisição de imagem usando três modos de operação: Varredura, escaneamento por base de Hadamard, e escaneamento por base de Hadamard com sensoriamento compreensivo. Segundo, discutimos como os diferentes parâmetros experimentais do sistema ótico afetam a aquisição. Finalmente, comparamos a performance dos três modos de operação quando usados para a aquisição de images com tamanhos entre (8px, 8px) e (128px, 128px).

Palavras-chave

Câmeras de Píxel Único; Sensoriamento Compressivo; Ghost Imaging; Imageamento Computacional.

Table of contents

List of figures

List of codes

List of symbols

DCT – Discrete Cosine Transform

SPI – Single Pixel Imaging

SPC – Single Pixel Camera

QGI – Quantum Ghost Imaging

BBO – β-Barium Borate

PBS – Polarizing Beam Splitter

CGI – Computational Ghost Imaging

SLM – Spatial Light Modulator

LC – Liquid Crystal

TN-LC – Twisted Nematic Liquid Crystal

DMD – Digital Micromirror Devices

PWM – Pulse with modulation

PSNR – Power Signal-to-noise Ratio

SNR – Signal-to-noise Ratio

MSE – Mean Square Error

RMSE – Root Mean Square Error

CS – Compressive Sampling

CNNs – Convolutional Neural Networks

Make Good Art

Neil Gaiman, .

1
Introduction

In past years, advances in camera hardware, especially in consumer electronics, focused on increasing the pixel density in digital cameras. For a long time, the number of megapixels a camera had was an important indicator of its quality, and each year manufacturers would boast how their sensors had more pixels than their competitors.

In conjunction with the increase in resolution came the increase in file size for the aforementioned pictures. To handle these increased file-sizes, ever in more demand, many compression techniques were developed, of which, arguably, JPEG is one of the most widely known.

JPEG is based upon the idea that most images are sparse in the frequency domain, i.e., certain frequencies will be more relevant to the image than others. Using a discrete cosine transform (DCT), JPEG compression will store the coefficients of the most relevant frequencies in the image while discarding the frequencies where there is less or no information. Although JPEG is just one example of a compression method, most compression schemes will use a similar rationale. In that sense, it is possible to say that the current digital photography paradigm is based on:

1. Sampling a scene using a sensor array and quantitating it into a digital representation (bit).

2. Taking the raw image format and compressing it into a file type that yields a lower-sized digital file.

That said, with advances in machine learning and AI applications in mobile devices, the race for the highest number of pixels has been replaced by a competition for the best image processing algorithms in an area known as computational photography.

Computational photography exploits novel image capture and processing techniques intending to improve a camera's capabilities or to allow for functionality that would not be possible in film-based or classical digital photography. Like the High Dynamic Range Image Acquisition, some of these techniques have been around for a while. However, as AI-processors in mobile devices get more powerful, new capabilities have been developed, like improving low-light acquisition, stitching together parts of different images to improve exposure, and reconstructing scenes using deep learning algorithms.

These advances in computer science have not limited themselves to consumer technologies. Much of the advances that allowed for computational photography in mobile phones have now been used to create novel image acquisition techniques like super-resolution (1), non-line of sight (2), and single-photon 3D imaging (3).

With these new advances in signal processing, two main questions arise. First, whether more pixels are necessarily better for image acquisition, and if, considering that the scene is sparse, is it paramount to sample the whole scene, or is it possible to acquire only those frequencies that will be relevant for the compressed data?

With these questions in mind, we come to the field of single-pixel imaging (SPI), where a set of signal processing techniques is used to reconstruct an image from a set of measurements from a single-pixel detector. SPI has been demonstrated to allow for the development of multi-spectral cameras (4), three-dimensional image acquisition (3), and the development of relatively inexpensive cameras in wavelengths where sensors are not widely available (5). This work aims to (i) demonstrate how such single-pixel image acquisition systems can be built; (ii) provide a proof of concept; and (iii) build a python library that could potentially be used to allow for the development of new applications in this area.

Chapter 2 provides the theoretical framework behind SPI and discusses some of the work done in the past few years. Chapter 3 will provide details on the experimental implementation of the SPI system used in this work, and Chapter 4 will examine the results acquired. Finally, Chapter 5 will conclude and provide a vision of possible follow-up works. The python library developed during this study is included in Appendix 6.

2
Theoretical Overview

While typical image acquisition systems rely on sensor arrays for two-dimensional image acquisition, single-pixel cameras spatially correlate the point-sensor data by employing computational reconstruction based on structured illumination shined upon an object. Historically, the first implementations of such imaging systems were based on Quantum Ghost Imaging, which uses entangled photons for the spatial reconstruction of point detector data.

Considering these two closely related techniques, in section 2.1 we will commence with Quantum Ghost Imaging (QGI), introducing the basic concepts underpinning the approach used in single-pixel imaging systems. Following the introduction of QGI, we will spend section 2.2 discussing the essential components of SPI systems and how they work together. In the subsequent sections, we will provide an overview of the current developments behind different aspects of such systems, discussing the different modulation schemes in section 2.2.1, modulation patterns in section 2.2.2, and imaging modes in section 2.2.3. Finally, in section 2.2.4, we will explore some current applications of these systems and trends in this research area.

2.1
Ghost Imaging

To understand the ghost imaging framework, it is helpful to interpret an image as a two-dimensional map of the optical interaction between a light source and a given object.

In the case of a classical camera, the spatial and interaction intensity information is acquired simultaneously as each pixel in the detector array is responsible for measuring the intensity shining upon it, with each one having a defined position. In this scenario, a single measurement is capable of capturing enough information for a map (i.e., the image) to be constructed. On the other hand, Ghost Imaging consists in acquiring the spatial and amplitude information independently, relying on the correlation of different measurements to reconstruct a scene.

Although there are several ways to implement a ghost imaging measurement, its first demonstration relied on entangled photons for image reconstruction (6). This necessity of using a correlated biphoton light source has led this method to be initially interpreted as a quantum phenomenon, being called Quantum Ghost Imaging. Later, experimental and theoretical work demon-

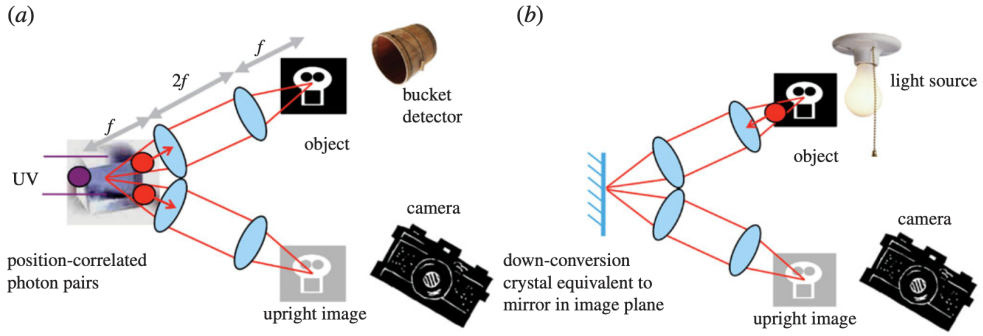

Figure 2.1: While in classical imaging systems (b) the spatial and object information are captured at the same time, in a Quantum Ghost Imaging system (a) this information is split between the two entangled photons. Adapted from (6)

strated that ghost imaging could be performed using non-quantum sources by employing, for example, pseudo-thermal light (7) or structured illumination, which ended up paving the way for Computational Ghost Imaging and Single-pixel cameras (7).

2.1.1 Quantum Ghost Imaging

Figure 2.1 shows a representation of a ghost imaging system implemented with position-momentum entangled photons, where the correlations shared by such photons allow imaging without the necessity of a spatially resolving detector. In such systems, the entangled photons generated by pumping a non-linear -Barium Borate (BBO) crystal with a laser (25) are orthogonally polarized and can be separated into signal and idler beams by a polarizing beamsplitter (PBS). In one of the beams, we place the object to be imaged, and after it a bucket detector (i.e., A detector that will integrate all the light that shines upon it, providing no spatial information), in the other, we place a spatially resolving detector. If we consider both paths individually, neither contains enough information to recreate the image of the object. By assuming that the entangled photons are correlated in position and momentum, it's possible to reconstruct the image by considering each pathway's coincident measurements. In this case, whilst the spatially resolving detector will provide information about the pixel position where the measurement was done, the bucket detector will give the amplitude information on that point. Considering that the image was formed without directly obtaining any spatially resolved image information from the object itself, the term "Ghost Imaging" was coined.

Considering the necessity of employing entangled photons, the question

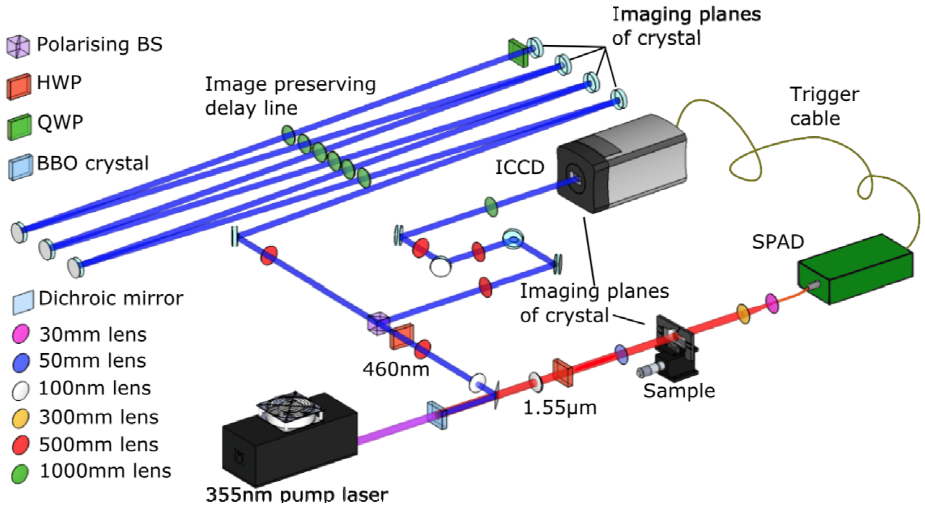

Figure 2.2: One of the uses of quantum ghost imaging is in allowing the use of non-degenerate entangled photons to leverage different wavelengths for image acquisition. In this figure, adapted from (5) , an object is imaged at 1.55μm, while the spatial information is detected at 460nm.

of whether this is a quantum phenomenon arises. Although the parametric down-conversion is a quantum effect, here it is used only as a source of spatial correlation, which could, in principle, be derived from a classical source. As we'll discuss further in the Computational Ghost Imaging section, this spatial correlation can be recreated classically in several ways, such as modulating a laser beam with spatial patterns and using it to extract spatial information of the scene.

While, at first glance, this image acquisition method can be considered less efficient than classical cameras, it allows applications not supported by other methods. In one of these applications, the parametric down-conversion can be tuned to yield correlated idler and signal photons of different wavelengths. Using these entangled photons, it's possible to illuminate the object at one wavelength and record the spatial information at another, perhaps where the imaging detector is more sensitive or less noisy (5)(6). Figure 2.2 shows an example of such ghost imaging system, allowing to "image" a sample in 1.55μm, while acquiring the spatial information of the scene in 460nm. By enabling the separation of the amplitude and spatial information, this imaging method allows the mix and match of detectors to increase the overall system efficiency and lower the costs.

2.1.2

Computational Ghost Imaging

Considering that only the spatial correlation between the photons is needed for image reconstruction, attention was given to identify other ways to implement such imaging systems without needing an entangled photon source. From the different implementations that arose in this area, the most prominent one was developed by Shapiro et al. (8), coined Computational Ghost Imaging (CGI). This classical implementation uses a spatial light modulator (SLM) to create binary intensity patterns (masks) to be projected onto the object. As the modulation patterns are well-defined and known prior to the measurement, the spatial information can be derived from the correlations between the optical beam and projected masks. In this scenario, it's possible to reconstruct an image by employing a single-element detector to measure the intensity of the light that is transmitted or backscattered from the object when the light is modulated with a respective modulation pattern (6). The image mapping will then, in simple terms, be derived by the weighted sum of the measured amplitudes from each projected pattern. Besides being far more straightforward to implement than its "Quantum" counterpart, this computational implementation easily allows the combination of different detectors to improve the wavelength coverage and polarization sensitivity.

2.1.3 Single-Pixel Imaging vs Computational Ghost Imaging

Once it became clear that CGI could lead to different acquisition schemes, other optical implementations were derived, of which single-pixel cameras are one of the most prominent. From an optical perspective, CGI and single-pixel imaging are the same, even though they are commonly treated as separate research fields. In a nutshell, the distinction stems from optical scheme employed for the measurement. While SPI places the SLM after the object (structured detection), computational GI places it before the object (structured illumination). Besides the SLM placement, in SPI systems, the use of a compressive sampling approach is common. Figure 2.3 shows two common representations of CGI and SPI systems, where it can be seen that these two schemes can be demonstrated by interchanging the locations of the light source and the detector.

In this work, the structured detection scheme was chosen and used in tandem with compressive sensing. Thus, we'll refer to the optical system herein described as "single-pixel imaging" even though this distinction can be interpreted as a convention.

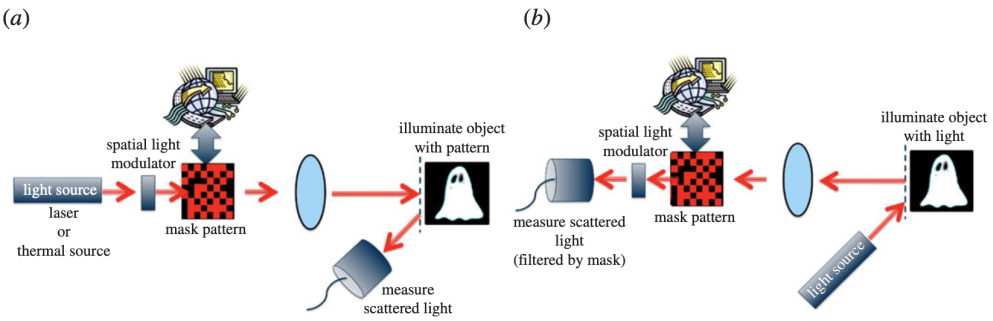

Figure 2.3: Comparison between a computational ghost imaging (a) and a single pixel camera system (b). Although the overall principle is the same, the distinction is made according to the use of structured illumination, or detection, respectively. Adapted from (6)

2.2 Fundamentals of Single-Pixel Imaging

As introduced in the previous section, single-pixel imaging consists of shining the transmitted or backscattered light from an object into a spatial light modulator and then detecting the integral signal using a point-detector. The correlation between the modulation mask and the measured intensity can be used to reconstruct an image computationally. Figure 2.4 shows an example of such a system, where the modulation patterns are multiplied by the corresponding measurement of the point-detector and summed to yield a reconstructed image. Although there are several ways to acquire and reconstruct images in the SPI scheme, this is one of the simplest examples.

Considering this simple setup, the single-pixel camera can be interpreted as an optical computer that sequentially measures the inner products between the incident light field of the scene and a set of two-dimensional test functions. Mathematically, given a $N \times N$ pixel image, a single-pixel system can be understood as:

Let us consider I the image of the object with N^2 pixels. We note P^k as a 2D-pattern loaded on the SLM and the respective m_k measurement collected by the point-sensor, such as:

$$m_k = P^k_{1\times N^2} \cdot I_{N^2\times 1} \tag{2-1}$$

As the N^2 P^k patterns are projected onto the sample, a correspondent measurement m_k will be collected by the point-sensor. Considering this formulation, a single-pixel imager can be understood as the linear system $M = P \cdot I$, where M is the $N^2 \times 1$ measurement vector, P is the $N^2 \times N^2$ set of measurement patterns and I the image vector that needs to be determined. Given this system,

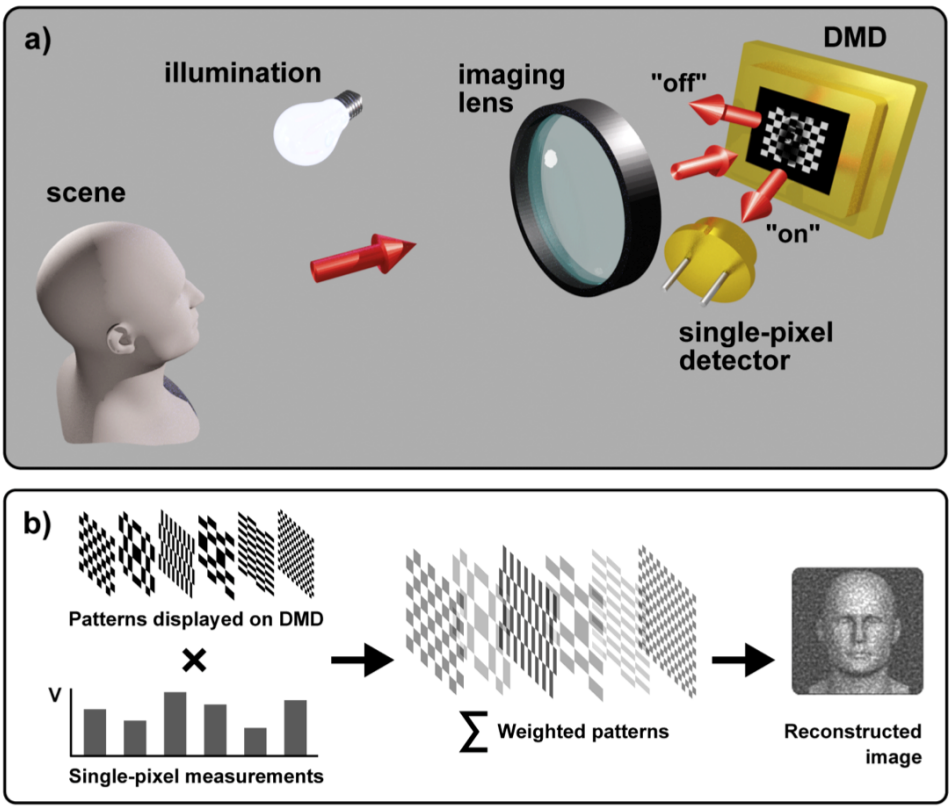

Figure 2.4: A structured detection setup. (a) The transmitted/back-scattered light from the object is shined upon a SLM and modulated before being detected by a single-pixel detector. (b) The image is reconstructed by combining the amplitude measurements from the single pixel detectors with the modulation patterns supplied to the SLM. Adapted from (7)

two main aspects need to be taken into account:

- **Acquisition:** How to choose/design the sequence of P patterns that will be projected?
- **Restoration:** Knowing the set of P patters and M measurements, how to restore the image I?

In the coming sections, we'll discuss different approaches to deal with both Acquisition and Restoration in single-pixel imaging and outline the choices used in this work.

2.2.1 Modulation Schemes

As discussed in the previous sections, an SPI system depends on establishing spatial correlations between the illumination beam hitting the sample and a set of modulation patterns. The aim of using a spatial light modulator

is to modulate the intensity of the incoming beam as a function of the spatial distribution of the mask, allowing for later reconstruction of the scene.

To modulate the beam intensity, several modulation schemes can be used. In this section, we'll provide a brief overview of the most common modulation techniques for spatial-light modulators and how they function.

2.2.1.1 Pseudothermal

One type of modulation pattern employed by SLMs is a random or pseudo-random modulation pattern. When utilizing such patterns, a simple weighted sum is used to reconstruct the image of the scene.

Although there are several ways to generate those patterns, one is to create a source of pseudothermal light by passing a laser beam through a diffuser. The diffuser can take many forms, as, for example, a rotating ground-glass diffuser (9). As the diffuser is rotated, the intensity pattern generated from the constructive and destructive interference of the light beam will vary with time. By further passing the beam through a turbid medium, the patterns can be further spatially randomised(7). In order to provide the required spatial correlation, the incoming light is passed through an optical beamsplitter to divide it into reference and object beams (7).As the reference beam will reveal the spatial information of the scene, the object will return the interaction information between the light and the object.

Despite this method being highly inefficient, due to being limited to random patterns and the complexity involved in the implementation, it allows modulation of wavelengths where other types of SLMs are not usually present.

Figure 2.5 shows an example of a SPI system for X-rays (9). In this implementation, the speckle pattern is generated by modulating the incoming beam with a rotating diffuser and recording the speckle patterns over time. In a second step, the X-ray is shined on a sample, and the recorded patterns are used to correlate the point-sensor data for image reconstruction spatially. As this kind of method requires only a single point sensor, the overall X-ray dose used for imaging can be decreased, allowing the imaging of fragile samples.

2.2.1.2 Liquid Crystal Spatial Light Modulators

While it's possible to establish a modulation pattern using beam diffusers, it would be preferred to modulate the laser beam's intensity using a computer-controlled SLM that would attune the signal as a function of an arbitrary pattern. This alternative would allow the spatial correlation without the need

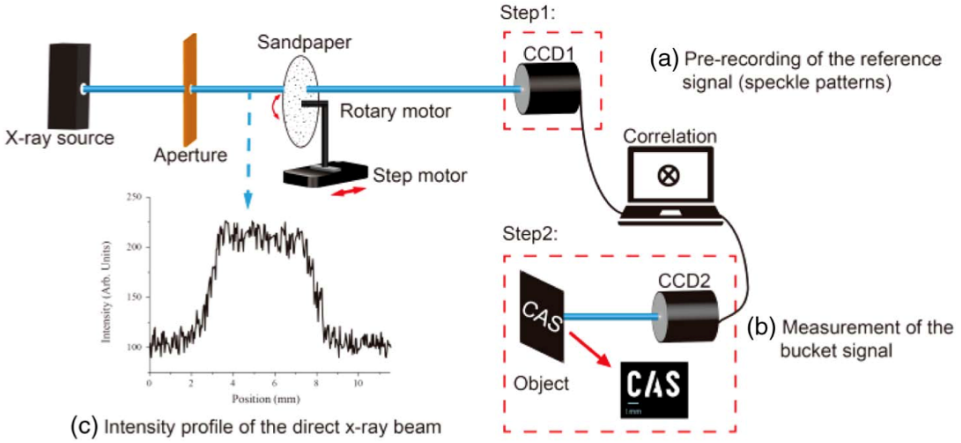

Figure 2.5: In a pseudothermal modulation setup, a reference speckle patterns is pre-recorded and subsequently used to modulate an incoming light source. By referencing the pre-recorded patterns and the measurements from the detector, it's possible to reconstruct an image of the object. Adapted from (9)

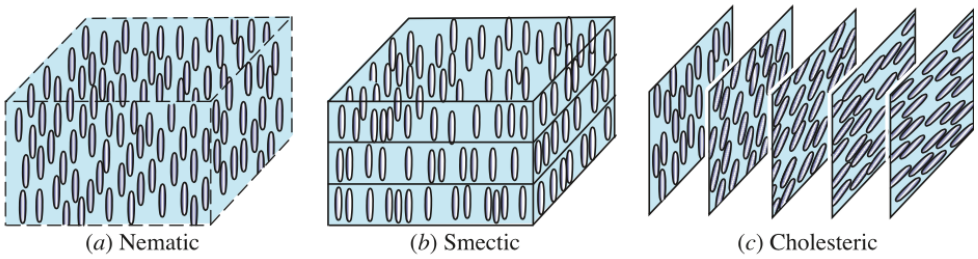

Figure 2.6: Comparison between the different molecular organizations of liquid crystals. Adapted from (17)

for a reference beam (as the computer will store in memory the modulation pattern used) and enable the use of masks different from random patterns, which are inefficient for image reconstruction. Although several techniques would allow this addressable modulation, one of the most common (and the one used in this work) employs liquid crystals to spatially modulate a laser beam.

A liquid crystal (LC) consists of a collection of elliptical organic molecules whose properties lie between those of solids and liquids. While they lack positional order, they maintain orientational order. Usually, these molecules can be classified by how they are organized and fall within three primary forms (Figure 2.6). In Neumatic form, the rod-shaped molecules are all oriented in the same direction, but their position is random. In smectic, the molecules' orientation remains the same, but their centers are stacked in parallel layers

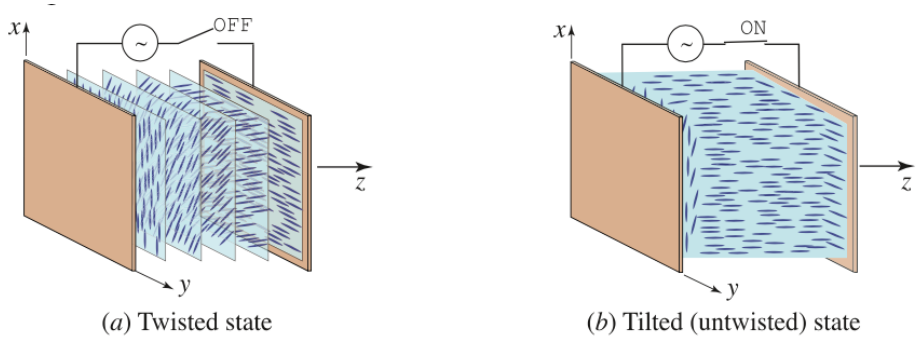

Figure 2.7: By placing a twisted nematic liquid-crystal cell between two conductors, it's possible, by controlling an external electric field, to change the cell's orientation by switching the field off (a) and on (b). Adapted from (17)

within which they have random positions; they, therefore, have positional order only in one dimension. The final form, cholesteric, consists of a distorted form in which the orientations undergo helical rotation about an axis.

Due to its anisotropic nature, LC interacts with light locally as a collection of uniaxial crystals, modifying the incident light's properties in the optical axis parallel to the elongated direction. These materials can be used to modulate the incident light to alter its phase, polarization, or amplitude.

To allow polarization modulation, a Twisted nematic liquid crystal (TN-LC) form is used, which corresponds to nematic liquid crystals on which a twist is externally imposed. At rest, TN-LC show a twist of the optical axis (elongated size) as a function of the depth (z). When an external electric field is imposed, the rods tilt and untwist Figure 2.7. Considering that the LC molecules interact with light in the elongated axis, the twisting and untwisting of these molecules can be used to modulate the incoming beam's polarization, and consequently, its amplitude when the LC cells is placed between polarizers.

In a rest state, the incoming linearly polarized light will interact will each plane of the LC and rotate its polarization by the twist angle. As the light travels through the crystal, the overall polarization rotation will correspond to the twist coefficient (degrees per unit length) and the travel distance. On the other hand, when an external electric field is imposed on the cell, the molecules will align themselves with the elongated axis in the direction of the electric field. In the new orientation, light passing through the cell won't be affected by the molecules, and will maintain the original polarization state. Figure 2.8 shows the amplitude modulation process of TN-LC when placed between two linear polarizers. In this use-case, by switching the external field, it's possible to control it the light will be transmitted or absorbed by the polarizers.

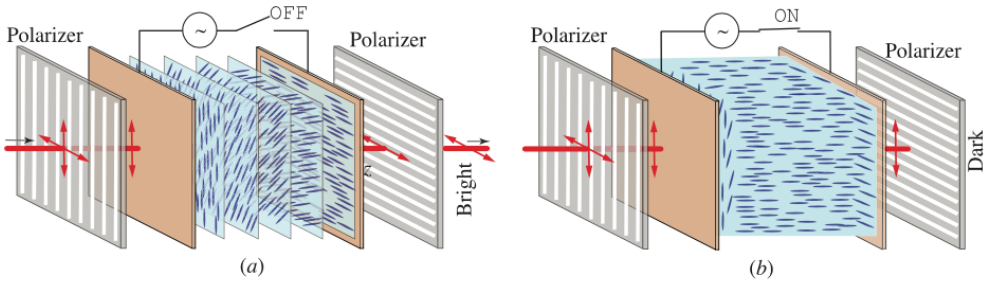

Figure 2.8: The control of the orientation of liquid crystal cells can be combined with linear polarizers to modulate the polarization of a light beam travelling through the cell. In this example, when the electric field is off (a), the polarization is changed due to the interaction with the liquid crystal molecules, whereas when an external electric field is applied (b), there is no modulation. Adapted from (17)

The use of LC in SLMs consists of arraying several independently addressable TN-LC cells that can be modulated with a defined voltage. Supplying the SLM with a greyscale image addresses each cell with a different voltage correspondent to the intensity value. This voltage causes a tilt of the LC molecules, which will modulate the polarization of the light shining on the array. As the SLM is placed between linear polarizers, it's possible to modulate the amplitude of the incoming light independently in each cell.

In a SPI system, the modulation patterns will result in a structured illumination hitting the sample. By storing in the computer memory information about the modulation masks, image reconstruction can be carried out.

2.2.1.3 Digital Micromirror Devices

Besides LC-SLMs, another category of widely used devices for illumination modulation is digital Micromirror devices (DMD). DMDs provide superior modulation range and broader wavelength response, with commercially available systems able to achieve display rates allowing near-video rate image reconstruction on a standard performance computer.

As the name suggests, DMDs consist of an array of microscopic mirrors that can be individually controlled and can be tilted in two positions, in and out of the optical axis (Figures 2.9). By feeding the DMD with binary patterns, it is possible to create structured illumination/detection masks for CGI and SPI, respectively. Besides generating binary patterns, the DMD also offers the possibility to load 8-bit gray-level patterns employing pulse width modulation (PWM).

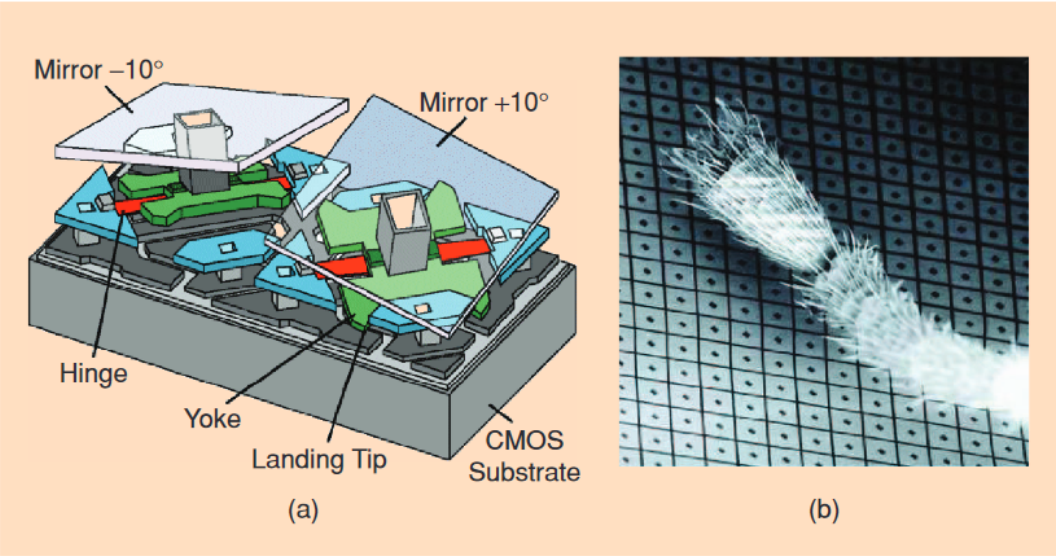

Figure 2.9: A close-up of a DMD's active region. Adapted from (10)

2.2.1.4 LED Arrays

Although all examples described until this point consisted of an external light source modulation, it isn't the only option. Considering the architecture of a structured illumination system (CGI), any form of illumination that allows for 2D modulation of the light source whilst recording the modulation patterns for later use could be used in principle.

Considering these boundary conditions, other systems have been proposed to circumvent the limited frame rate of many SPC and CGI systems. One of such systems consists of using 2D LED arrays for high-speed structured illumination. By levereging the fast switching time of the LEDs and the use of a orthonormal basis set like the Hadamard, there have been demonstrations of systems with acquisition rates of to 1000fps (7).

2.2.2 Pattern Choice

The use of modulation patterns to generate spatial information is nothing new. In the early days of mechanical televisions, a rotating Nipkow disk has been used to extract spatial information of a scene. In this example, the signal was measured as each of the holes rotated past the scene and shined upon a detector, and line-by-line, this would construct an image. As discussed in the previous section, the modern version of this approach uses a SLM to generate and display a set of carefully chosen masks.

The most straightforward approach would be to mimic the concept of

mechanical televisions and measure a single area per-pixel, raster scanning a single pixel over the scene; Although this per-pixel measurement method works well, it is highly inefficient and usually depends on high light levels. Another approach is to use an orthogonal basis set to systematically sample a scene by breaking it down its component spatial frequencies.

In this section, we'll explore the most common sets of modulation masks used in SPI systems.

2.2.2.1 Random Binary

One of the simplest ways to implement an SPC, other than raster scanning, is to use random binary patterns. In this method, a SLM is used to display masks of randomly generated binary patterns. As the modulated laser light hits the detector, the overall signal intensity will be given by the integrated signal coming from all-white pixels displayed by the SLM.

In this scenario, image reconstruction will be given by the sum of the $N^2 \times 1$ binary masks weighted by the measurement signal:

$$I_{N^2\times 1} = \sum_{1}^{N^2} m_k \cdot P_{N^2\times 1} \tag{2-2}$$

Where, m_k represents the photon count detected and P the $N^2 \times N^2$ binary masks.

Although this method can be easily implemented, even without externally controlled SLMs (see pseudothermal modulation), it can take many iterations to reconstruct a low noise image (7). One way to improve image acquisition is to use a differential measurement, which also captures the signal corresponding to the negative modulation mask for each iteration. In this case, the SNR is expected to improve and be less affected by fluctuations in the light source. A differential signal acquisition can be described by:

$$I(x,y) = \langle(\delta m_k - \langle \delta m_k\rangle)\rangle\langle(P_k(x,y) - \langle P_k(x,y)\rangle)\rangle \tag{2-3}$$

Where m_k represents the measured differential signal, P_k represents the k-th pattern, and the angle brackets denote an ensemble average for k iterations, $\frac{1}{M}\sum m_k$, where M is the number of measurements.

2.2.2.2 Hadamard Transform

While random binary masks can be easily implemented, the number of M measurements required to reconstruct an image tends to be $M >> N^2$, where N^2 is the number of pixels in the image. This inefficiency is a result of

the number of repeated measurements that exist when randomly sampling a scene.

One way to circumvent this issue is to use a method analogous to JPEG compression, where the image is treated as sparse and is decomposed into a set of coefficients of an orthonormal basis set. By sampling with binary masks from a basis set, the image compression occurs simultaneously as the sampling. As we'll discuss in the next section, this approach enables the use of compressive sensing techniques that may allow $M<N^2$ measurements to reconstruct a scene.

One of such orthonormal basis sets is the Walsh-Hadamard basis. Hadamard patterns are a generalized example of Fourier transforms characterized by being orthogonal with binary entries made up of either +1 or 1. This transformation decomposes a signal into a set of orthogonal, rectangular waveforms called Wash functions. A Hadamard basis set can be constructed by:

$$H(2^1) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2-4}$$

$$H(2^2) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix} \tag{2-5}$$

$$H(2^k) = \begin{bmatrix} H(2^{k-1}) & H(2^{k-1}) \\ H(2^{k-1}) & -H(2^{k-1}) \end{bmatrix} \tag{2-6}$$

One of the advantages of Hadamard transforms is that they are closely related to their inverse ($H \cdot H^T = nInd$, with Ind being the identify matrix), which enables image reconstruction without the need of matrix inversion, lowering the computational demand.

For a Hadamard acquisition of a N^2 pixel image ($N \times N$), a N^2xN^2 Hadamard matrix is created (encoder). For each iteration, a row of the encoder matrix is reshaped into the image size ($N \times N$), generating a binary pattern representing a set of frequencies (Figure 2.10). For each binary pattern, a signal m_k is captured from the single-pixel detector. A final image can be reconstructed by multiplying the N^2xN^2 encoder by the N^2x1 measurement vector M to produce a one-dimensional vector of the output image, I, that

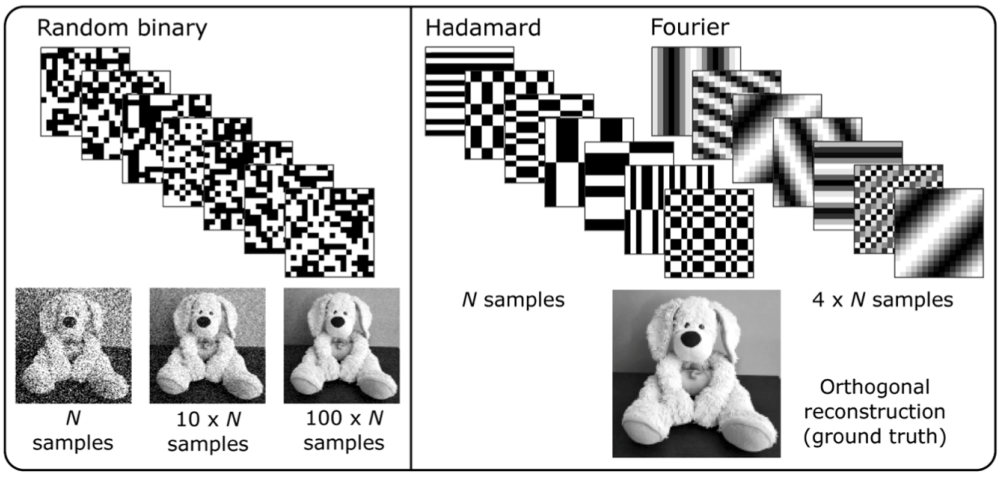

Figure 2.10: A comparison of different modulation patterns used for single-pixel imaging. In general random binary patterns will require a greater number of measurements to reconstruct the image when compared to Hadamard and Fourier basis sets. Adapted from (7)

can be then reshaped to the image size $(N \times N)$:

$$I = H \cdot M$$

As with random sampling, a differential measurement can be applied to improve the SNR. In a differential approach, for each iteration, a measurement is taken for both the Hadamard binary mask and its negative $(-H)$. Although this method requires twice the number of patterns, it can account for offsets in the image caused by variations in the background or source illumination (7).

By sampling with an orthonormal basis, the number of displayed patterns can be reduced and still enable recovering of an image. In Figure 2.11 we see the effects of reducing the number of patterns used in the image reconstruction when compared with ground truth by calculating the Power signal-to-noise ratio (**PSNR**), defined as (7):

$$PSNR = 10 \times log_{10} \frac{MAX_{I_{GT}}^2}{MSE} \tag{2-7}$$

Where the mean squared error (**MSE**), is given by (7):

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (I_{GT}(i,j) - I(i,j))^2 \tag{2-8}$$

In the Hadamard spectrum, the zero-frequency component in the top left, and the maximum frequency in the lower right. As we see in Figure 2.11, as the number of sampling patterns decreases, the image quality also decreases, but it's still possible to distinguish the object. This sampling effect can be

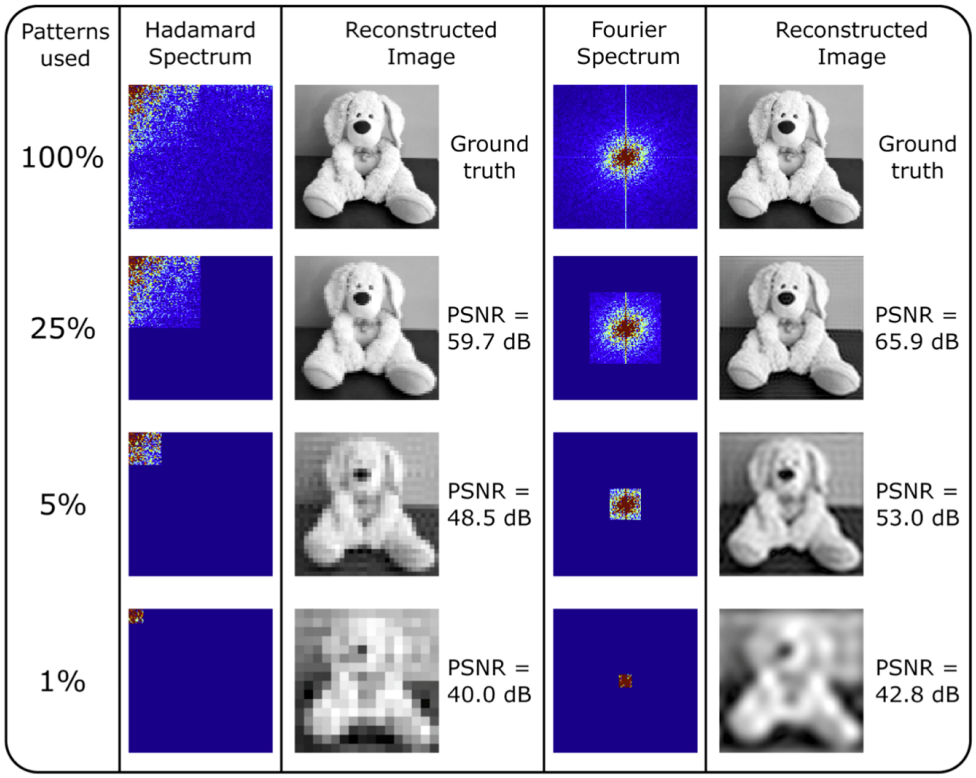

Figure 2.11: The effect of the reduction of the number of patterns in the image reconstruction done with Hadamard and Fourier basis sets. Adapted from (7)

interpreted as reducing the number of pixels in the image as the number of Hadamard patterns is decreased.

2.2.2.3 Fourier Basis

Besides the Hadamard basis, other sampling schemes have been proposed. In Fourier encoding, the pattern sets are constructed from greyscale masks. In this case, for a square image consisting of N^2 pixels, the masks are created for spatial frequencies 0 to $(N-1)$ in both x and y dimensions and frequencies u and v. Each pattern $P(u,v)$ is given by (7):

$$P(u,v) = cos(2\pi(\frac{ux}{N} + \frac{vy}{N}) + \phi) \tag{2-9}$$

Given that the intensity signal is insufficient to reconstruct an image, a phase measurement is performed by changing the phase term between four different values spaced between 0 and 2π. The Fourier spectrum component $F(u,v)$ for the spatial frequencies u and v are is defined as (7):

$$F(u,v) = (D_\pi - D_0) - i(D_{\frac{3\pi}{2}} - D_{\frac{\pi}{2}}) \tag{2-10}$$

Where D_ϕ represents the intensity measurement of each pattern P with a phase ϕ.

In this method, final image reconstruction is given by applying an inverse Fourier transform. As with the Hadamard encoding, the Fourier basis allows filtering of the number of patterns needed to reconstruct the image. Figure 2.11 compares the effect of sampling reduction between the two modes.

2.2.3 Image Acquisition and Reconstruction

As introduced in section 2.2, a SPI measurement can be summarized as:

1. Illuminating a scene and shining the light over a SLM;
2. Generating patterns in the SLM to encode spatial information of the scene;
3. Collecting the encoded light from the SLM into a detector and measuring the correspondent signal;
4. Using the set of mask patterns and signals to decode the scene's spatial information and reconstruct an image.

In the previous sections, we've discussed each of these steps and how they are related. In this section, we'll provide an in-depth overview of three scanning techniques used in the SPI system implemented in this work, discussing each method from both the acquisition and reconstruction perspectives.

2.2.3.1 Raster Scan

Raster scanning consists in measuring each pixel (or set of) individually and recording the corresponding signal from the detector.

Acquisition:

Raster scan image acquisition is given by generating 2D binary masks that contain a single white pixel in the position (i, j) of the $N \times N$ mask. For each iteration, a different pixel is set to 'on' while the rest of the mask is kept 'off', and the correspondent signal is recorded by the detector.

Reconstruction:

The final reconstruction will be given by the sum of the unitary patterns multiplied by their respective signal:

$$I_{N^2\times 1} = \sum_{1}^{N^2} m_k \cdot P^k_{N^2\times 1} \tag{2-11}$$

Where P^k represents the k-th unitary mask and m_k the corresponding signal.

For a $N \times N$ image, a set of N^2 measurements will be necessary for image reconstruction, one for each pixel position.

2.2.3.2 Differential Hadamard Basis Scan

As discussed in section 2.2.2, an orthonormal basis set can provide a more efficient way to encode and reconstruct an image in a SPI system by sampling specific frequencies of a given basis and measuring the correspondent coefficients.

From the various basis sets available for encoding, the Walsh-Hadamard transform is one of the most widely used due to its simplicity and properties that allow for image reconstruction without matrix inversion. A Hadamard encoder can be operated in a differential measurement to decrease the noise introduced by fluctuations in the scene illumination.

Acquisition

For a given $N \times N$ square image, a $N^2 \times N^2$ Hadamard matrix is generated as encoder. For each iteration, a $1 \times N^2$ row of the encoder is selected and reshaped to $N \times N$ to generate the mask pattern P_k. As the mask pattern is displayed in the SLM, the correspondent measurement m_k is recorded.

To achieve a differential measurement, the negative $-P_k$ of each mask pattern will be computed, and the correspondent measurement $-m_k$ will be stored as well. The difference between both measurements will be calculated, and a differential measurement δm_k will be stored.

Finally, the averages of the differential measurements and mask patterns will be subtracted from each measurement δm_k and its corresponding pattern P_k. The final differential signal can be described by:

$$I(x,y) = \langle (\delta m_k - \langle \delta m_k \rangle) \rangle \langle P_k(x,y) - \langle P_k(x,y) \rangle) \rangle \tag{2-12}$$

Where δm_k represents the measured differential signal, P_k represents the k-th pattern, and the angle brackets denote an ensemble average for k iterations, $\frac{1}{M} \sum m_k$, where M is the number of measurements.

Reconstruction

For an image I, a set of M measurements, and P patterns, we can describe a basis scan measurement as:

$$M_{N^2 \times 1} = P_{N^2 \times N^2} \cdot I_{N^2 \times 1} \tag{2-13}$$

Considering the properties of a Hadamard matrix that:

$$HH^T = nInd \tag{2-14}$$

with Ind being the identity matrix,

And

$$H^T = H \tag{2-15}$$

We can write,

$$HH^T = n(HH^{-1}) \tag{2-16}$$

$$H = nH^{-1} \tag{2-17}$$

Thus, given a Hadamard matrix H and the system:

$$M_{N^2\times 1} = P_{N^2\times N^2} \cdot I_{N^2\times 1} \tag{2-18}$$

The reconstruction

$$H^{-1}_{N^2\times N^2} M_{N^2\times 1} = I_{N^2\times 1} \tag{2-19}$$

Can be given simply by

$$\frac{1}{n} H_{N^2\times N^2} M_{N^2\times 1} = I_{N^2\times 1} \tag{2-20}$$

Thus, to reconstruct the image, a simple dot product must be calculated between the differential Hadamard encoder and the correspondent differential signals. For a $N \times N$ image, a set of N^2 measurements will be necessary to reconstruct the scene.

2.2.3.3 Differential Hadamard Compressive Scan

All acquisition modes described until now share the requirement of $M \geq N^2$ measurements to allow image reconstruction, being N^2 the number of pixels in a $N \times N$ image. Considering that the number of measurements required for image acquisition is one of the main limitations of SPI systems, it would be beneficial if it was possible to approximate the scene with $M < N^2$ measurements. One possible approach to allow subsampled images to be correctly estimated is to employ a compressive sampling (CS) approach.

In CS, we consider that the image to be captured is sparse in the frequency domain (i.e., that most frequencies will have zero elements, while some dominant frequencies will carry out most of the information), similar to the approach used in techniques like JPEG compression. In this case, instead of compressing the image after acquisition, CS aims to compress the signal during the sampling of a scene and store only the critical information required for signal reconstruction.

In a traditional image compression, a camera acquires a picture of a scene using N^2 pixel array sensor. After sampling, the N^2-pixel image is translated

into a $N^2 \times 1$ image vector I and decomposed as coefficients of an orthonormal basis (10):

$$I = \psi\alpha \tag{2-21}$$

Where α represents the $N^2 \times 1$ set of coefficients of the $N^2 \times N^2$ basis vectors ψ.

The compression aims to find a basis set ψ such as α is sparse, that is, only $M < N^2$ coefficients are nonzero. After the basis set is defined, the compression algorithm will store the information about the coefficients and their locations and discard the rest.

The main disadvantage of this compression approach is that although we strive to find a small number of M coefficients, N^2 samples are collected and its respective coefficients calculated. This means that although the full scene information is stored, most of it will end up being discarded.

To circumvent these issues, CS aims to use a set of test functions to sample a scene and directly acquire $M < N^2$, such as a sparse image signal can be recovered and used for image reconstruction.

In this case, we consider a set of M $1 \times N^2$ test functions ϕ_m and their respective measurements y_m by the single pixel sensor, such as (10):

$$y_m = \langle \phi_m, I \rangle \tag{2-22}$$

We can represent the CS process by the linear system (10):

$$y_{M\times 1} = \phi_{M\times N^2} I_{N^2\times 1} = \phi\psi\alpha \tag{2-23}$$

Where, ϕ represents the $M \times N^2$ matrix of test functions (encoder matrix), where the $1 \times N^2$ test functions ϕ_m test functions are stacked as rows. As we can see above, this transformation constitutes a dimensionality reduction from N^2 to M. CS aims to utilize test functions ϕ such as the sparse vector α can be recovered (10).

Acquisition

In the CS framework used in this work, Hadamard matrices were used as test functions. The measurement approach was the same discussed in section 2.2.3.3 where a differential measurement was used. The acquisition approach used in this scan mode differs only in the number of measurements taken. While in the Differential Hadamard Basis scan N^2 measurements were taken (equal to the number of pixels in the image), we acquire only M measurements, where $M \leq N^2$.

Reconstruction

As we are trying to estimate the image with $M \leq N^2$ measurements, the $N^2 \times 1$ coefficient vector α is longer than the $M \times 1$ measurement vector M,

which means that there will be an infinite amount of solutions y^*that solve the equation. To resolve this problem, convex optimization algorithms can be used to minimize the linear problem and approximate the sparsest solution, i.e. the solution where α has the lest amount of non-zero elements. In this work, we used the CVXPY convex optimization library (11)(12).

Although to properly resolve this minimization problem a L0 norm would be theoretically be required, it has been shown that the use of the L1 norm can be used to approximate the image precisely or very closely in a computationally scalable way (10). In this case, the minimization problem is given by:

$$min||\alpha||_1 \tag{2-24}$$

Such that,

$$||\phi\psi\alpha - y|| = 0 \tag{2-25}$$

with the L_n norm defined as:

$$||x||_n := (\sum_{i=1}^{n} |x_i|^p)^{1/p} \tag{2-26}$$

2.2.4 Applications and Current Trends

The versatility of single-pixel cameras comes from its system requirements. As long as there are point-based sensors and some way to spatially modulate the incoming light, it should be possible to implement a SPI setup. This flexibility has led these imaging systems to be applied to several use cases, from multispectral imaging (4) to secure communications (13). In general, it's possible to bundle the many applications and advances of SPCs into one of two categories: Acquisition and Reconstruction.

In the acquisition approach, the main concern is to enable imaging in domains where traditional 2D arrays cameras are not widely available, such as multispectral imaging, 3D imaging, etc. In reconstruction, the aim generally revolves around improving the computational methods that recover the image information.

In this section, we'll provide some examples of SPC applications and discuss how they can be used to enable new possibilities in image acquisition.

2.2.4.1 Multispectral Imaging

Multispectral imaging refers to image acquisition of multiple wavelength ranges across the electromagnetic spectrum. Traditional multispectral cameras require one or more sensor arrays capable of independently resolving or sep-

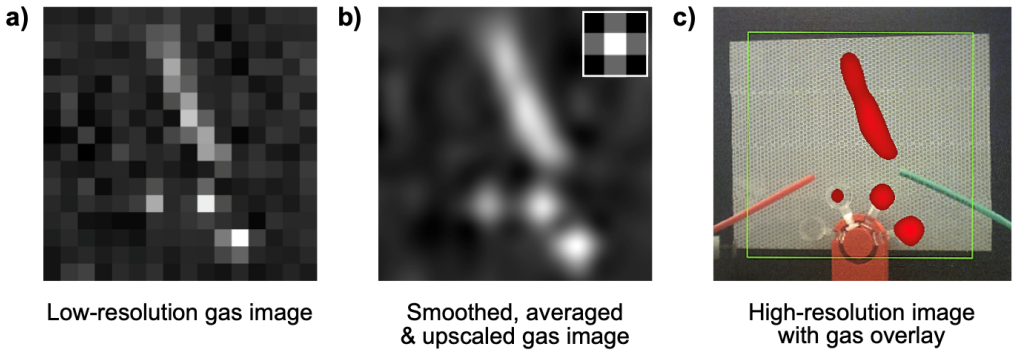

Figure 2.12: An example of a gas leak image taken by a single pixel camera tuned to the gas' absorption frequency (a), the post processed and smoothed image (b) and its superposition with a classical camera image to provide leak localisation (c). Adapted from (14)

arating the channels corresponding to the different imaging bands. Although these cameras have been available for quite some time, there are two main disadvantages with these systems. Firstly, broadband sensors can be costly and have limited availability. Secondly, when compared to their narrowband counterparts, these sensors usually suffer from lower resolution and SNR.

On the other hand, by employing SPC, it's possible to build multispectral systems by operating multiple narrow single-pixel sensors in tandem. This approach can use high-resolution sensors that are usually less expensive than their 2D array analogues in exchange for computational complexity, which is generally easier to scale. Furthermore, it's possible to combine SPC with traditional digital cameras to provide a lower resolution image overlay of specific wavelengths of interest.

Figure 2.12 demonstrates a SPI system tuned to image in the absorption spectra of methane gas. In this work, the low-resolution image coming from a single pixel camera is combined with a digital camera to provide an overlay capable of recognizing the position of gas leaks. With the high-resolution camera proving accurate depiction of the scene, the SPC can be operated in low resolution, allowing real-time operation (14).

Figure 2.13 demonstrates another setup where the light hitting a broadband detector is split into different wavelengths. Applying a compressive sampling algorithm makes it possible to separate the information coming from the different bands and reconstruct independent 2D images for multiple wavelengths (4).

2.2.4.2

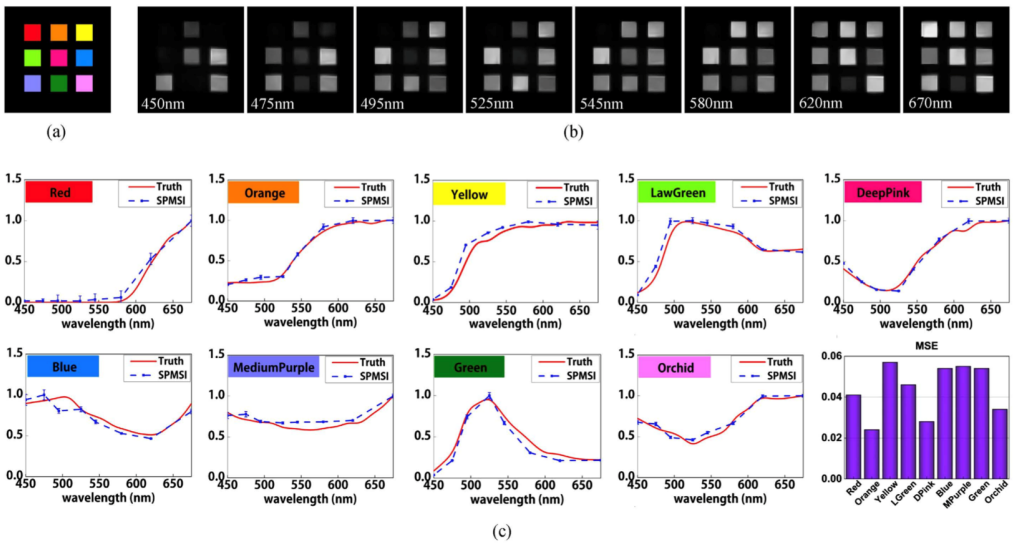

Figure 2.13: A multispectral single pixel imaging system capable of reconstructing images of a color checker film in multiple frequencies. Adapted from (4)

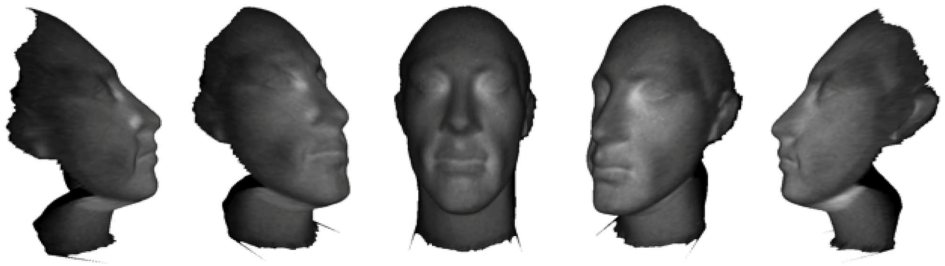

Figure 2.14: 3D reconstruction of an object using a single pixel camera. Adapted from (3)

3D Imaging

Another area where single-pixel cameras can be applied is in depth estimation and 3D imaging. In this scenario, multiple SPC are used to derive 2D images of a scene. From the differences in shading between the images, it's possible to derive the scene's surface gradients and reconstruct a 3D image of the object.

Figure 2.14 shows one example where multiple single-pixel detectors were used together with a light projector to create 3D scans of an object (3). As the different spatially separated sensors acquire the same structure patterns, it's possible to achieve pixel registration and compare the different images for reconstruction.

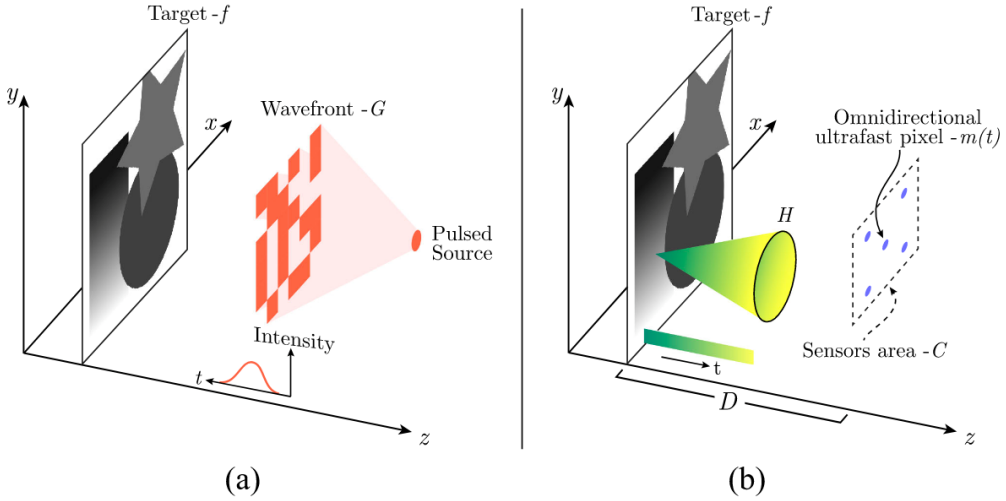

Figure 2.15: In a lensless image acquisition setup, a pulsed modulated light source is used to illuminate a scene (a) and a ominidirection ultrafast sensor(s) used to reconstruct the image given its relative position to the object (b). Adapted from (15)

2.2.4.3 Lensless Imaging

SPC can also be employed in lensless imaging, where a single-pixel detector is used to reconstruct a scene without the need for optical lenses. In one implementation, the scene is illuminated with a pulsed light source modulated with a mask. As the light hits an object and is reflected into the detector, the time delay is used to estimate the reflected signal's spatial position. In this method, other SPC can be placed to improve the spatial determination of the scene. Overall, this method allows lensless imaging with fewer modulation patterns than regular non-time-tagged SPCs. Figure 2.15 shows a schematic of such imaging system (15).

2.2.4.4 Machine Learning and Image Reconstruction

One approach explored to improve SPC image acquisition is to use machine learning approaches for image reconstruction. These methods aim to use deep learning in the form of convolutional neural networks (CNNs) to perform image reconstruction with fewer measurements than traditional compressive sampling.

In deep-learning SPI, CNNs are trained to produce both the sampling pattern to be displayed in the SLM and the reconstruction algorithm. Figure

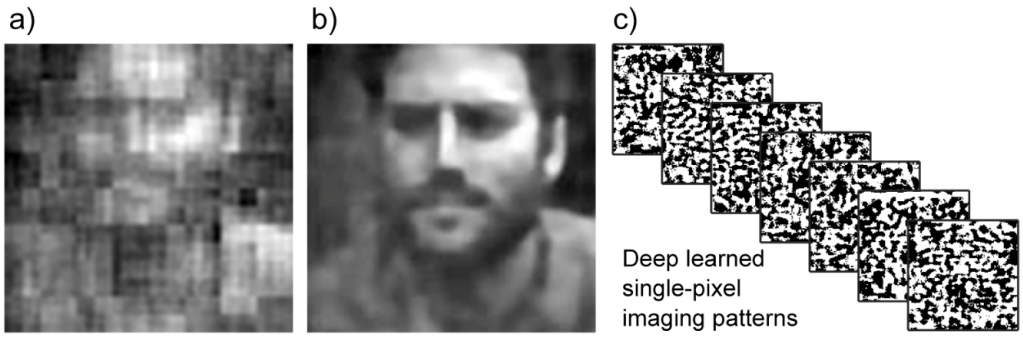

Figure 2.16: An example of a single-pixel imaging reconstruction using deep-learning. (a) shows the reconstruction calculated using 4% sampling of a set of Hadamard patterns, while (b) shows the reconstruction using a trained neural network using deep-learned pattern sets. (c) shows examples of the deep-learned patterns. Adapted from (7)

2.16 shown an example where this approach was used to reconstruct an image using only 4% of the measurements, allowing for video-rate image acquisition of higher resolution images.

As new reconstruction methods are explored and optimized, SPC becomes more widespread and can be applied to other use-cases.

3
Experimental Implementation

In the last section, we introduced the theoretical framework behind single-pixel cameras, their operation and provided details on the different image acquisition methods used in this work. In this section, we'll give an in-depth look into the experimental implementation used in this study and provide the rationale behind the choices taken, paving the way for the discussion of the experimental results in Chapter 4.

3.1
Optical Setup

As introduced in chapter 2, single-pixel cameras rely on modulating the illumination being shined upon a sample, storing information about the modulating pattern, measuring the light-sample interaction, and finally processing the arrays of patterns and their corresponding measurements to reconstruct an image computationally. Figure 3.1 shows a diagram of the system employed in this work and its components. In general terms, the system can be described by :

1. Expanding the laser beam to cover the sampling area;
2. Cleaning the laser mode and collimating the beam;
3. Projecting the beam upon the sample and into the SLM active area;
4. Focusing the beam onto a single photon detector;
5. Converting and processing the signal from the single-photon detector and the SLM modulation pattern to reconstruct the image.

This section will expand on each of these steps, providing further information about how and why they were implemented in this work.

3.1.1
System Enclosure

In a system that relies on photon counts for image reconstruction, the control of the stray photons hitting the detector becomes a significant concern. To decrease the effect of illumination changes and improve the signal-to-noise ratio (SNR) of the system, an enclosure was built around the main optical components (Figure 3.2).

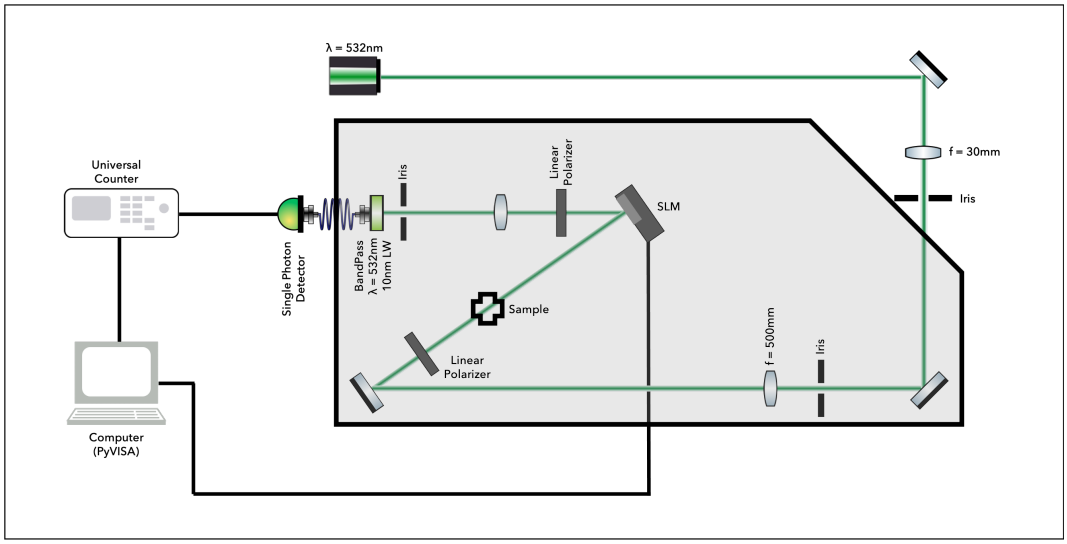

Figure 3.1: A visual schematic of the optical implementation used in this work

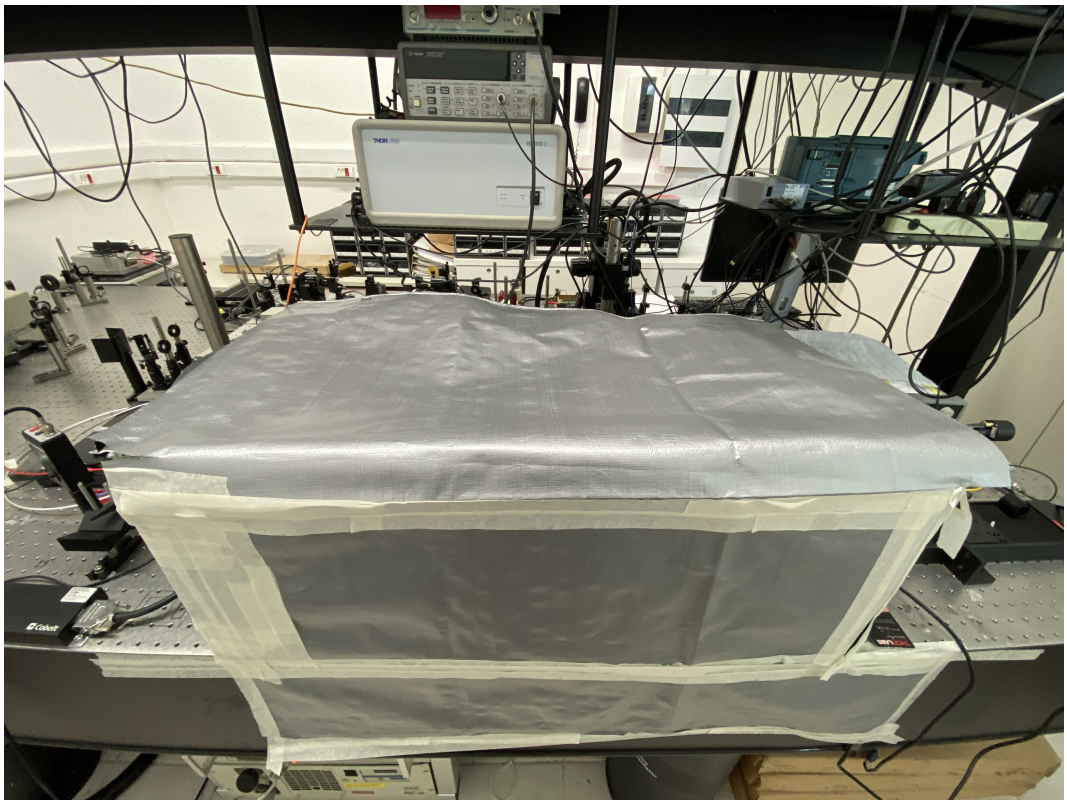

Figure 3.2: To decrease the effect of stray light into the system, an enclosure was constructed

The enclosure's structure was build from cardboard around the grey area shown in Figure 3.2 and a tarp was placed on top of it to prevent outside light from getting into the system. Due to size constraints, the optical path between the laser and the first iris was left open, but tests didn't seem to show it as a problem, as most of the stray photons that get into the optical path will be filtered out by the bandpass filter in the detector.

3.1.2 Beam Cleaning and Collimation

The first step towards implementing the SPI system is to make sure that the incident illumination hitting the sample is big enough to cover the sampling area and small enough to be contained inside the active region of the spatial light modulator. To achieve those constraints, the laser beam was first collimated through a Gaussian spatial filter composed of two confocal lenses (30 and 500mm focus length respectively) with an iris placed in the focal point Figure 3.3. The beam path was aligned so the laser light (Cobolt Samba 150 532nm, Cobolt) would hit the iris' center, cutting away part of the noise and acting as a Gaussian filter. The ratio between the lenses' focal points was chosen to magnify and expand the beam size by a factor of 16.7 approximately. An optical chopper (SR540, Stanford Research Systems) was added to modulate the laser light and gate the universal counter (53131A, Agilent). After benchmarking the acquisition modes, the optical chopper didn't seem to improve the SNR much when compared to setting a constant gate time. For simplicity, the synchronization between the counter and chopper was tested and implemented in the automation library but ultimately was not used in this work.

After passing through the spatial filter and being expanded, the laser beam hits a secondary iris placed on the optical path (Figure 3.4). The use of an auxiliary iris gives the user control of the beam diameter, which may be increased or decreased as a function of the sample size to ensure that the size constraints are respected.

3.1.3 Sample and Beam Modulation

The sample used in this work consisted of a 3D printed (Objet 30 Pro, Stratasys) optical target with a 1.5x1.5cm cross-shaped hole. The beam diameter was set to cover the whole sample region. Before hitting the sample, the incoming beam passes through a linear polarizer (Figure 3.5) to make sure

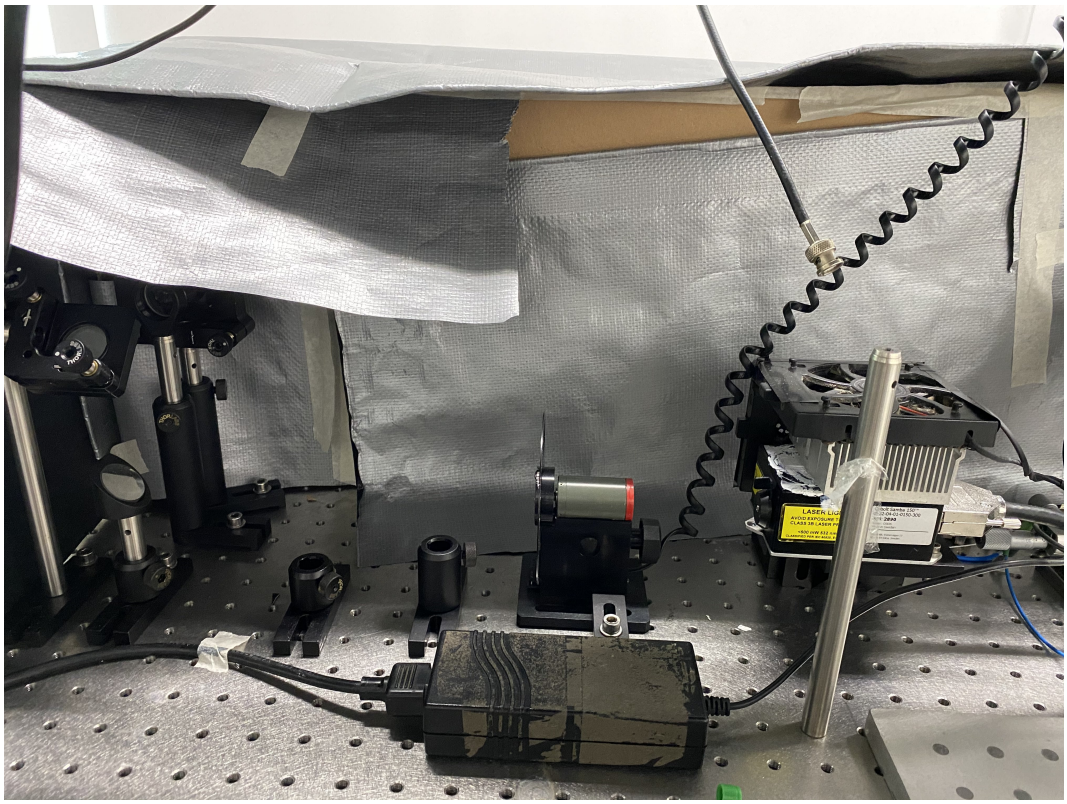

Figure 3.3: The laser first goes through a light chopper (not used in this work) and is then coupled into a Gaussian spatial filter to clean up the laser's mode.

that the incoming line is linearly polarized. The polarizer's angle was set by maximizing the photon count hitting the detector.

After hitting the target, the beam hits the SLM's active region (SDE1024, Cambridge Correlators). Once hitting the SLM, the beam's polarization will be modulated as a function of the binary pattern displayed. Another linear polarizer Figure 3.6 is set after the SLM to allow for amplitude modulation, and its angle was determined to maximize and minimize the photon count when white and black patterns were displayed in the SLM, respectively.

The SLM modulation is done via a VGA port connected to an external computer. The computer recognizes the SLM as an external monitor and modulates the liquid cristal active region as a function of the greyscale values being displayed. The binary masks, and consequently the spatial modulation, are constructed by displaying white (RGB: 255,255,255) or Black (RGB: 0,0,0) pixels in different regions of the SLM's digital monitor.

3.1.4 Detection and Information Processing

After being modulated, the beam is focused (f=250mm) onto an iris (Figure 3.7) to remove unintended diffracted light coming from the SLM,

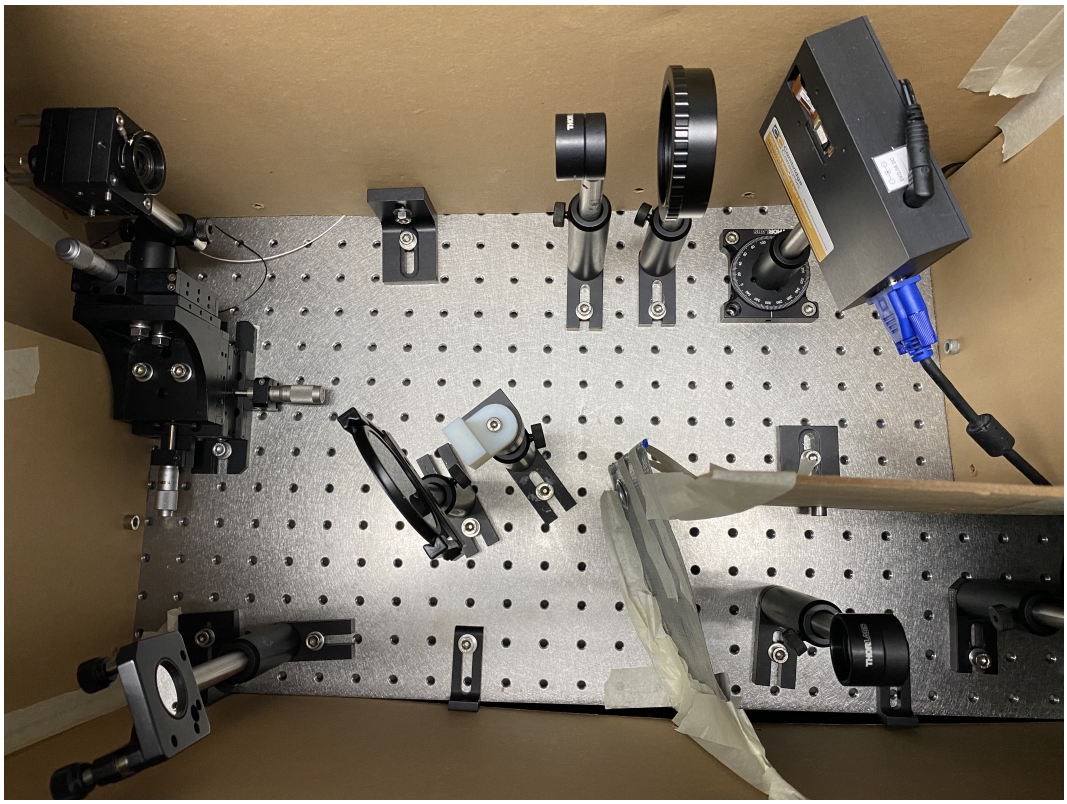

Figure 3.4: An image showing the disposition of the optical elements inside the enclosure.

capturing only the light traveling through the optical path. To improve the SNR and decrease the number of stray photons getting into the detector, a bandpass filter (λ=532nm, Line Width = 10nm) was added after the iris. After hitting the iris and the bandpass filter, the incoming light is coupled into an optical fibre and taken to a single photon detector (ID100, ID Quantique).

To count the number of photons being detected as a function of time, the detector was connected to a universal counter (53131A, Agilent). When a photon hits the detector, it outputs a square-wave signal inputted into the counter and represents one count. The counter will then totalize the number of counts inside a given gate time and average then along with a given measurement time window.

To store the number of photon counts for a given measurement and allow for the correlation with the binary masks displayed in the SLM, a GPIB-USB instrument control device is connected to the counter (GPIB-USB-HS, National Instruments), allowing for control through a computer. To interface with the GPIB-USB controller, the pyvisa (16) library was used.

3.2

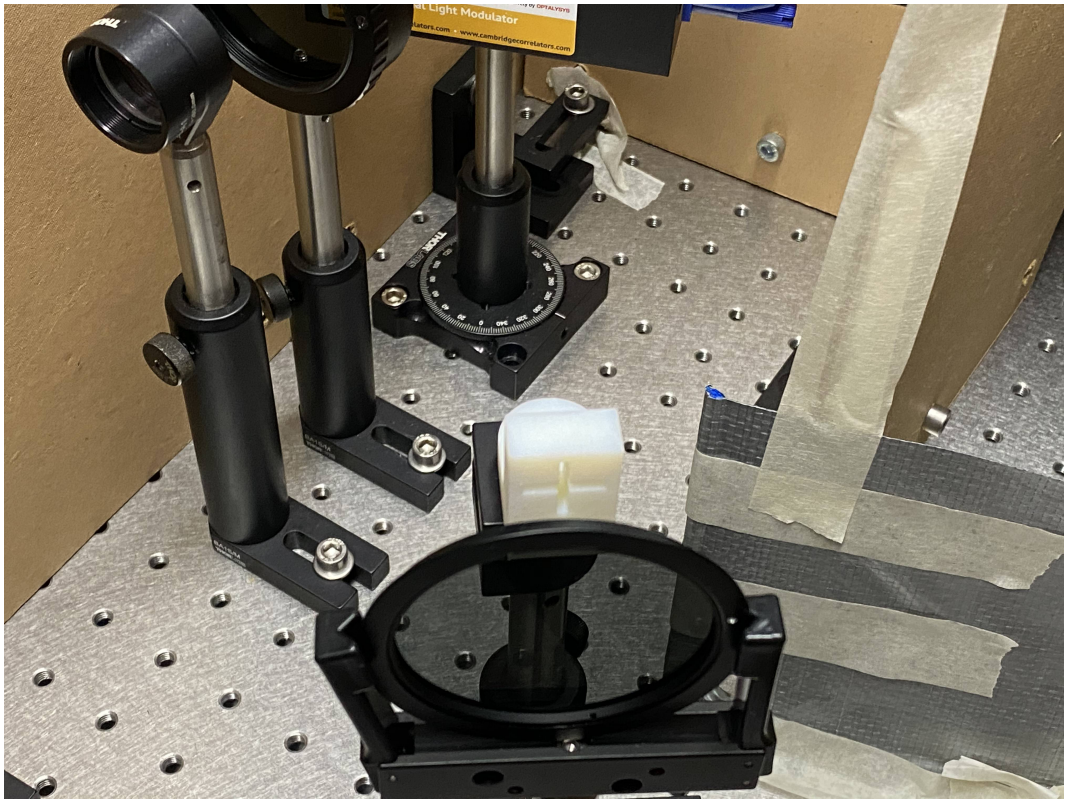

Figure 3.5: The object consists in a 3D printed target placed in the optical axis.

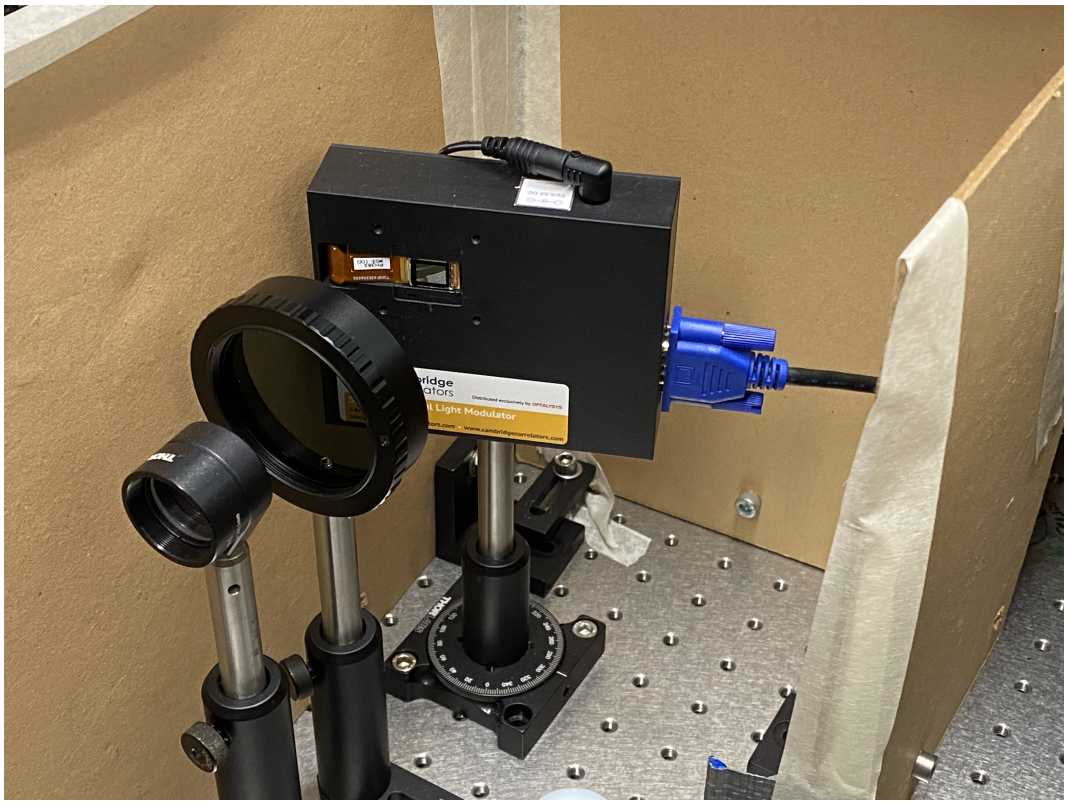

Figure 3.6: The spatial light modulator used in this work

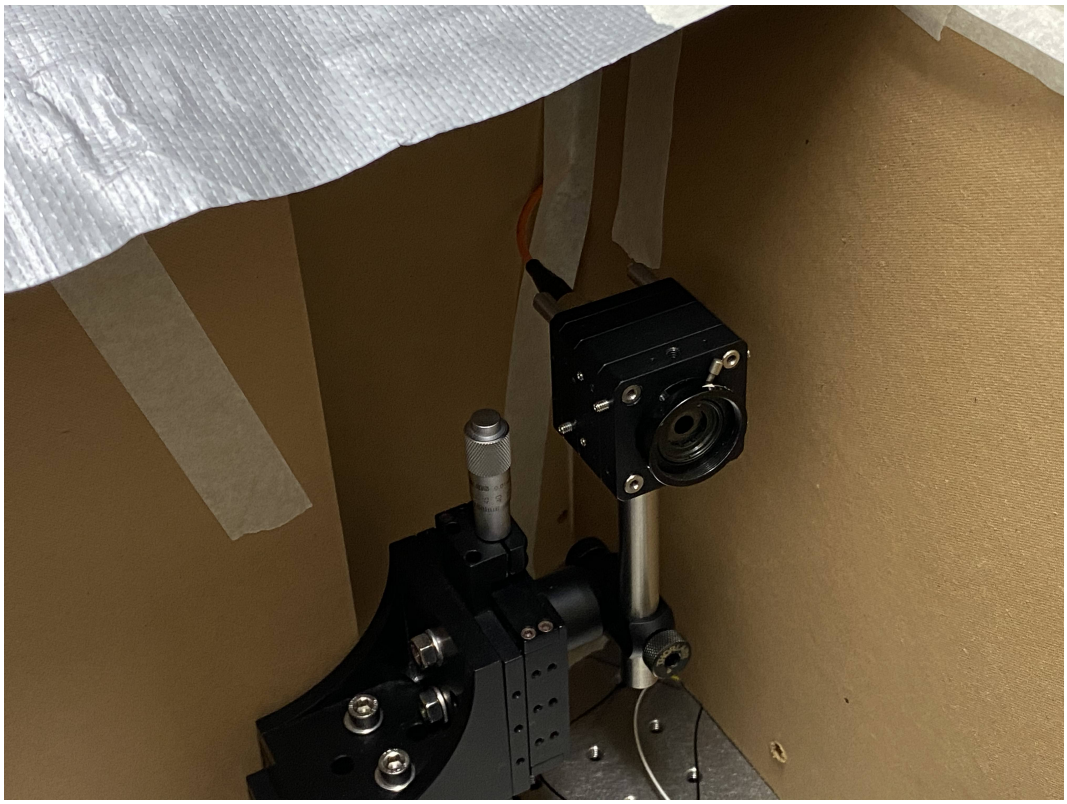

Figure 3.7: The detector assembly consisted of an iris, bandpass filter and fibre coupler taking the incoming light to a single photon detector.

Python Single Pixel Imaging Library

As described in the previous sections, the operation of a single-pixel camera relies not only on an optical setup but also on an external controller that sets the light source modulation and synchronizes it with the detection system. The controller needs to present the SLM with a set of modulation masks, record the number of photons detected as a function of the masks, and use these mask-photon count pairs to reconstruct the images using the methods described in Chapter 2.

In this work, an external computer (Macbook Pro 2019, Apple) was used for instrument control by using the PyVisa library (National Instruments) in a python environment running in a jupyter notebook. During this study, a set of algorithms were created to allow for measurement and reconstruction of images using the raster, basis, and compressive scanning modes. The library was structured to allow for future extension and implementation of new measurement modes and encoding masks.

In this section, we'll explain the rationale behind the implementation of the library, the main code used to run the experiments, and an in depth overview of the implementation of the three acquisition modes, showing, when

possible, where the library could be extended to allow for future functionality.

3.2.1 Acquisition Modes Implementation

In terms of instrument automation, the only two parts of the system that depend on external control are the SLM and the detector. The single-photon sensor depends on an external counter for totalizing the number of photons hitting the detector in a given moment and storing that information as a vector, while the SLM relies on the choice of binary masks for modulation.

The **detector class** (code block 1) corresponds to the python class responsible for sending commands to the universal counter by USB-GPIB interface and storing the measurement information in NumPy arrays. This class can be broken down into three components:

(i.) **Initialization:** The class initialization given by the **"___init___"** command sets the instrument parameters used by the universal counter. In these experiments, the counter was operated in channel 1, with DC coupling, 50 impedance, and 10000ms timeout.

(ii.) **Timed Measurement:** The timed totalization measurement function given by the **"measure_gate"** class function. This function takes in two input parameters, the **"gate"** and **"meas_time"**. The **"gate"** parameter corresponds to the gate time that the counter will use, i.e., the time during which the counter will totalize the counts coming from the detector. The **"meas_time"** corresponds to the total measurement time in which the counter will be gated, and the results averaged. In an experiment that uses 0.1ms gates and 1ms measurement time, for example, that would mean that the detector would totalize the number of photons counted in each 0.1ms gate 10 times up to a total time of 1ms when it would average the gate measurements and return a single value and its standard deviation.

(iii.) **Externally Gated Measurement:** The externally gated measurement function given by the **"measure_external"** class function. This function takes in one input parameter, **"n"**, which corresponds to the number of repetitions. In this measurement mode, an external TTL signal is used to externally gate the universal counter and used to start and stop a totalization measurement, when the measurement will then be repeated n times. In the case of an optical chopper connected to the universal counter, the chopper's frequency will define the gate's window and will trigger the counter. Normally this measurement mode would be preferred to increase the signal-to-noise-ratio (SNR), but due to limitations with the SLM used in this work, the SNR didn't seem to improve. Due to its higher complexity and the lack of synchronization

port in the SLM, this mode was ultimately not used in this work but was implemented to allow for future improvements in the system.

Code Block 1: Detector Class

```
class detector:
    def __init__(self):
        rm = visa.ResourceManager()
        universal_Counter_GPIB = rm.list_resources()[0]
        self.Counter = rm.open_resource(universal_Counter_GPIB)
        self.Counter.timeout = 10000
        self.Counter.write(":INPut1:IMPedance 50")
        self.Counter.write(":INPut1:COUPling DC")
        self.Counter.write(":EVENT1:LEVEL 2")

    def measure_gate(self, gate, meas_time):
        self.Counter.write(":CONFigure:TOTalize:TIMed")
        crr_time = time.time()
        data = []
        while (time.time() - crr_time) < meas_time:
            data.append(float(
            self.Counter.query(":MEASure:TOTalize:TIMed? {}".format(gate))))
        data = np.array(data)
        data_stats = np.mean(data), np.std(data)
        return data_stats

    def measure_external(self, n=1):
        data = np.zeros(n)
        for i in range(n):
            self.Counter.write(":TOTalize:ARM:SOURce EXTernal")
            self.Counter.write(":TOTalize:ARM:STOP:SOURce EXTernal")
            self.Counter.write(":INITIATE")
            data[i] = float(self.Counter.query(":FETCH?"))
        return np.array([data.mean(), data.std()])
```

The SLM used in this work doesn't have any external control ports, having only a VGA interface. When connected to a computer, it will recognise the SLM as an external monitor, and graphics projected on it will be converted to grayscale and used to modulate the active region. Considering these limitations, the general mode of operation relies on generating binary masks on a

GUI window and projecting them on the SLM's monitor, which is controlled by class (code block 2). This class is composed of three class methods:

(i.) Initialization: The "**___init___**" class method will create a new window with a size equal to the SLM's resolution (800x600 pixels). When the window is first created, it will project a solid greyscale color (the greyscale given by the k value), allowing for the projection of totally white or black masks used for optical alignment.

(ii.) Mask Projection: The "**ShowArray**" method will take a 2D array as input and project it to the SLM's screen. In case the input array's shape is different from the SLM's resolution, it will be padded with black pixels around it. In this work, all binary masks used are $N \times N$ square masks, which means that black padding will always be projected around the masks.

(iii.) Exit: The exit class method is used to close all GUI windows and stop the measurements.

Code Block 2: SLM Class

```
class slm:
    def __init__(self, resolution = (600,800), k=255,
     native_res=(800,600)):
        self.resolution = resolution
        self.native_res = native_res
        self.slm_window = 'SPI Mask Generator'
        cv2.namedWindow(self.slm_window, cv2.WINDOW_NORMAL)
        mask = np.ones(self.resolution)*k
        mask_show = cv2.resize(mask.astype('uint8'), self.native_res)
        cv2.imshow(self.slm_window, mask_show.astype('uint8'))
        cv2.waitKey(0)

    def ShowArray(self, array, native_res=(600,800)):
        out = np.zeros(native_res)
        out[:array.shape[0], :array.shape[1]] = array
        cv2.imshow(self.slm_window,out)
        cv2.waitKey(1)

    def exit(self):
        cv2.destroyAllWindows()
```

The **Detector** and **slm** classes are the building blocks of the instrument automation. In the subsections below, we'll go through the implementation of each measurement function and how they use these methods described above.

3.2.1.1 Raster Scan Implementation

Raster scanning depends on setting each pixel individually to white, one at a time, and recording the correspondent photon counts. Whilst projecting each pixel separately is best for the measurement resolution, it will decrease the number of photons being collected at each iteration, decreasing the SNR. The raster scan implementation allows the binning of pixels to increase the measurement area per iteration by reducing the overall resolution to allow more refined control of the tradeoff between SNR and resolution.

As will be the case in the other modes, before being projected onto the SLM's display, the $N \times N$ binary masks will need to be resized to fit the SLM's resolution. To account for this difference in resolution, all measurement modes have a corresponding auxiliary function responsible for upscaling the $N \times N$ binary masks to fit into the SLM's display area. In the case of the raster scan mode, this functionality is given by the **"generate_encoded_raster"** function (code block 3).

Code Block 3: Raster Scan Encoder

```
def generate_encoded_raster(self, n, idx, native_res=(600,800)):

        seed = np.zeros((n, n))
        seed.ravel()[idx] = 1

        y_size = int(native_res[0]/n)
        x_size = int(native_res[1]/n)
        shape = min(y_size, x_size)

        encoded = np.repeat(seed.repeat(shape).reshape(n,n*shape), shape, axis=0)

        return encoded
```

The **"generate_encoded_raster"** function takes as input the binary mask size (n) and the corresponding pixel index and generates the correspondent binary mask. The upscale factor is calculated as a function of the mask's and SLM's resolution and used to generate a larger $N \times N$ mask that will be projected onto the SLM (code block 3).

The measurement itself is carried out by the **"raster_scan"** function. This function takes as input the $N \times N$ image size (**n**) and the gate and measurement times. For an $N \times N$ image, the **"raster_scan"** function iterates the **"generate_encoded_raster" function** to create a binary mask that

contains a single (or binned) white pixel and project it to the SLM. Once projected, the SLM modulates the laser light, and the universal counter totalizes the gate counts during the total measurement time. Each average count is then assigned as a value for that given pixel (or set of), allowing image reconstruction just by presenting the pixel values in a 2D array (code block 4). As each pixel is acquired individually, the total number of measurements will be equal to the total number of pixels in the image (N^2).

Code Block 4: Raster Scanning Mode

```
def raster_scan(self, n, gate=.1, meas_time=.5, Save=True):

        pd = detector()
        image = np.empty((n, n))
        for i in tqdm.tqdm_notebook(range(n**2)):
            crr_mask = self.generate_encoded_raster(n, i)*255
            self.ShowArray(crr_mask)
            image.ravel()[i] = pd.measure_gate(gate, meas_time)[0]

        if Save==True:
            encoded = np.ones((n, n))
            counts = image
            save_measurement(encoded, counts, image)


        return image
```


3.2.1.2 Hadamard Basis Scan Implementation

As previously discussed, in this work, we employ a Hadamard transform as an encoder to allow for a more effective sampling of the scene. While being the only mask encoder available in the current implementation of the PySPI, the code was written to allow for easy implementation of other transforms for image sampling. Independently of the transform used, as was the case in the raster scan, we'll have a set of mask generating and measurement functions for each measurement mode.

The mask generation is given by the **"generate_encoded_hadamard"** function. This function takes as input the $N \times N$ image size (**n**), the mask index (**idx**) and the seed encoder matrix (**encoded**). In the Hadamard transform, the encoder matrix corresponds to an $N^2 \times N^2$ Hadamard matrix generated

by the Numpy Walsh-Hadamard function. For each iteration, a column of the encoder matrix will be chosen (**idx**) and will be reshaped to a $N \times N$ 2D array representing one of the transform's frequencies. This reshaped column will be upscaled as a function of the SLM's resolution and projected (code block 5).

Code Block 5: Hadamard Basis Scan Encoder

```
def generate_encoded_hadamard(self, n, idx, encoded, native_res=(600,800)):

        seed = encoded[idx]

        y_size = int(native_res[0]/n)
        x_size = int(native_res[1]/n)
        shape = min(y_size, x_size)

        encoded = np.repeat(seed.repeat(shape).reshape(n,n*shape), shape, axis=0)

        return encoded
```

As discussed in chapter 2, the implementation of a differential measurement can improve the SNR by compensating for fluctuations in the illumination. The implementation of such measurement is done by the **"basis_scan_diff_hadamard_timed"**. This function will compute the $N \times N$ basis scan image of the scene by iteratively calling the **"generate_encoded_hadamard"** function to generate a given 2D Hadamard binary mask (representing one of the frequencies) from the $N^2 \times N^2$ Hadamard encoder and measuring the respective number of photons in a timed measurement. A similar measurement will be acquired after each iteration for the negative binary mask and the difference between the counts computed (differential counts). Subsequently, the mean of the set of differential counts and encoder will be subtracted from the respective arrays. Finally, the dot product will be computed between the differential Hadamard encoder (encoder minus its mean) and the differential counts (differential counts minus its mean). The final array is then reshaped into a $N \times N$ array that represents the reconstructed image (code block 6). As was the case of the Raster Scan measurement, image reconstruction will require a total of measurements equal to the number of pixels in the image (N^2).

Code Block 6: Hadamard Basis Scanning Mode

```
def basis_scan_diff_hadamard_timed(self, n, gate=.1, meas_time=.5, Save=True):
        pd = detector()
```

```
        encoded = la.hadamard(n**2)
        # np.save('hadamard_mask', encoded)
        counts = []

        for i in tqdm.tqdm_notebook(range(n**2)):
            crr_mask = ((self.generate_encoded_hadamard(n,
            i, encoded) -(-1))/(1-(-1)))*255

            self.ShowArray(crr_mask)
            photon_count = pd.measure_gate(gate, meas_time)[0]
            # photon_count = np.random.random()*10000

            self.ShowArray(1-crr_mask)
            photon_count_inv = pd.measure_gate(gate, meas_time)[0]
            # photon_count_inv = np.random.random()*10000

            diff_count = photon_count - photon_count_inv
            counts.append(diff_count)

        encoded_mean = encoded.mean(axis=1).reshape(n**2,1)
        diff_encoded = encoded - encoded_mean
        diff_counts = np.array(counts).reshape(n**2, 1)
        diff_counts = diff_counts - diff_counts.mean()
        reconstruct = np.dot(diff_encoded, diff_counts).reshape(n, n)

        # np.save('photon_count', diff_counts)

        if Save==True:
            save_measurement(encoded, diff_counts, reconstruct)

        return reconstruct
```

Although this code is meant for the Hadamard transform, the **"generate_encoded_hadamard"** function could be modified to take as input other transform encoders as seed (eg. Fourier, Cosine) and the **"basis_scan_diff_hadamard_timed"** modified to use these new transforms for the measurements. In the future, the PySPI library could allow for the implementation of different transforms, allowing for further work on how different transforms affect the basis scan measurement.

3.2.1.3 Hadamard Compressive Sensing Implementation

The compressive sensing imaging mode follows the same implementation rationale as the basis scan, differing only in the image reconstruction algorithm employed. In the case of the compressive measurement, an additional input (M) is given to the measurement function, representing the number of measurements that will be acquired to reconstruct the $N \times N$ image. If M is set equal to the number of pixels in the image (N^2), the result given by the compressive sensing algorithm will converge to that of the basis scan mode (code block 7).

In the compressive mode, the reconstruction algorithm will recreate the scene from $M \leq N^2$ measurements. As previously discussed, this reconstruction can be done by minimizing the L1 norm of the inverse linear problem $M = P \cdot I$, where P is the differential encoder matrix, I the $N^2 \times 1$ image vector, and M the differential count array.

Code Block 7: Compressive Hadamard Scanning Mode

```
def compressive_scan_diff_hadamard_timed(self, n, m, gate=.1,
meas_time=.5, Save=True):
            pd = detector()
            encoded = la.hadamard(n**2)
            counts = []
            encoded_m = encoded[:M,:]

            for i in tqdm.tqdm_notebook(range(m)):
                crr_mask = ((self.generate_encoded_hadamard(n,
                i, encoded) -(-1))/(1-(-1)))*255

                self.ShowArray(crr_mask)
                photon_count = pd.measure_gate(gate, meas_time)[0]
                # photon_count = np.random.random()*10000

                self.ShowArray(1-crr_mask)
                photon_count_inv = pd.measure_gate(gate, meas_time)[0]
                # photon_count_inv = np.random.random()*10000

                diff_count = photon_count - photon_count_inv
                counts.append(diff_count)
```

```
encoded_mean = encoded_m.mean(axis=1).reshape(m,1)
diff_encoded = encoded_m - encoded_mean
diff_counts = np.array(counts).reshape(m, 1)
diff_counts = diff_counts - diff_counts.mean()
diff_counts_m = diff_counts.squeeze().reshape(m, 1)



try:
    vx = cvx.Variable((n**2,1))
    objective = cvx.Minimize(cvx.norm(vx, 1))
    constraints = [encoded_m*vx == diff_counts_m]
    prob = cvx.Problem(objective, constraints)
    result = prob.solve(verbose=True)
    reconstruct = np.array(vx.value).squeeze().reshape(n,n)
except:
    print('Could not Reconstruct')

if Save==True:
    save_measurement(encoded_m, diff_counts_m, reconstruct)

return reconstruct
```

In code block 7 we see that the reconstruction starts by setting the $N^2 \times 1$ image vector as a variable for our linear problem (**vx**) and the L1 norm as the minimization objective (**objective**). The $Ax = B$ linear problem is set (**constraints**) from the **M** differential encoder and count measurements acquired during the scan (**encoded** and **diff_counts**, respectively). The algorithm will then try to minimize the L1 norm of the set problem and approximate the $N^2 \times 1$ image vector (**result**). Finally, if the minimization is successful, the image vector is reshaped into a $N \times N$ 2D image and returned. In case the M number of measurements isn't enough for image approximation, the function will return an error.

4
Results

As stated in chapter 1, this study aimed to provide a proof of concept of these novel imaging systems and build the foundations (both experimentally and computationally) that would allow further work in this research area. In the upcoming sections, we'll discuss the results obtained and demonstrate the system's functionalities and its limitations.

Section 4.1 will start by discussing the experiments carried out to characterize the system and to define the operational parameters used in the final dataset. Section 4.2 will then discuss the results obtained in the Raster and Basis scanning modes and how they compare with each other. Afterward, in Section 4.3, the focus will be given to the compressive Hadamard scanning mode and how its experimental parameters affect the final image acquisition.

4.1
System Characterization

Although the imaging modes described in this work use different image acquisition methods and restoration, they all depend on the binary mask modulation and photon counts provided by the SLM and detector, respectively. Considering this shared backend, before we can go ahead towards image acquisition and mode comparison, it's necessary to understand how the different experimental parameters affect the overall acquisition and to benchmark the capabilities allowed by the current experimental implementation.

In a nutshell, when optimizing the system's parameters, it's necessary to maximize the contrast between the photon counts given by white and black pixels, and minimize the total acquisition time. The former is necessary to ensure that the photon counts measured as a function of a given binary mask are primarily due to the white pixels in the modulation pattern, an essential requirement to allow image reconstruction. The latter is necessary to ensure that images can be captured in a reasonable time, as when measuring an $N \times N$ image, usually, N^2 measurements are needed.

To allow for optimization of these constraints, a curve was constructed by modulating the SLM with masks of solid greyscale color of increasing values ranging from black (RGB: 0,0,0) to white (RGB: 255,255,255) and measuring the respective photon totalization counts in different gate values.

Figure 4.1 shows the greyscale curves associated with different gate set points whilst maintaining a constant total measurement time of 1s. In Figure

4.1A, we see that a longer gate time leads to more accumulated counts when compared to shorter times. This offset in photon counts is expected as the photon flux reaching the detector is constant, and its order of magnitude ranges from around 100K photons/s to 10M photons/s for black and white masks, respectively. When normalizing the curves (Figure 4.1 B), we see that the overall curve shape remains the same regardless of the gate time, showing the differences only as an offset. Considering these results, we can attest that the SLM modulates the incoming light in a similar way in the parameter ranges chosen for this work. All in all, given the proportion of the photon flux in black and white pixels, we should expect a SNR in the order of magnitude of 100x. In this case, the SNR comes from the optical system itself and is limited by the maximum amplitude modulation that the SLM/Polarizer set can achieve.

After ensuring that the SLM's modulation response was comparable between different gate setpoints, the question of which gate and measurement times to use in image acquisition arises. Figure 4.2A shows how the integration time affects the counts hitting the detector and the overall SNR when the gate time is maintained constant at 10ms. The integration time seems to have little to no effect on the counts, maintaining the SNR primarily constant. This response is expected as an increase in the measurement times means essentially that there will be a larger number of gates being averaged, affecting predominantly the estimation of the mean. The experiments showed that even for low measurement times (0.1s) the SNR didn't seem to fluctuate much. The average SNR from all counts was calculated to be (133 +- 1.4).

Following the study of the effects of the total measurement time, a similar study was carried out for the gate time, maintaining the integration at 1s. Figure 4.2B shows that longer gate times lead to higher photon counts, whilst the SNR remains mostly constant around 130x. This response seems to corroborate the findings in the greyscale curve analysis, as we are seeing only an offset of the photon counts while the response remains the same.

Once measuring these parameters' effects on the SNR, experiments followed to describe what those results meant for image acquisition properly. Figure 4.3 shows the impact of the integration and gate times in the basis scan mode for an image of 16 by 16 pixels whilst maintaining the other parameter constant, just as before. All images in Figure 4.3 were normalized between 0 and 1 to allow for proper comparison. As shown in the figures below, there isn't much difference between the images, as the normalization would compensate for the offset caused by different gate times.

Considering the results obtained, a gate time of 10ms and an integration time of 100ms were determined to be a good compromise and defined as the

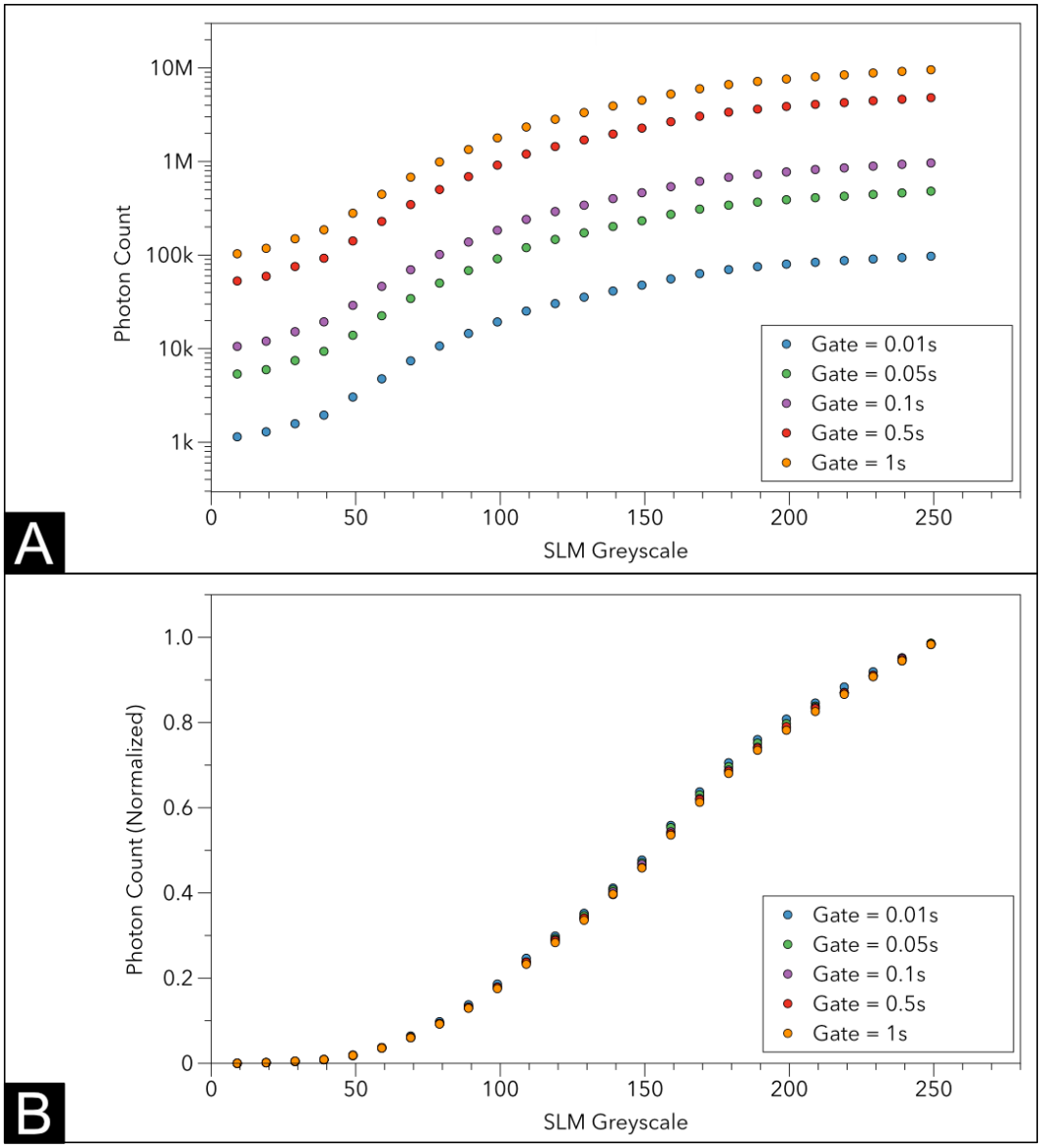

Figure 4.1: Effect of the gate time in the greyscale response curve of the spatial light modulator when maintaining the total measurement time constant at 1s. Although larger gate times yield more photon counts (a), when the curves are normalized, the overall shape stays the same (b).

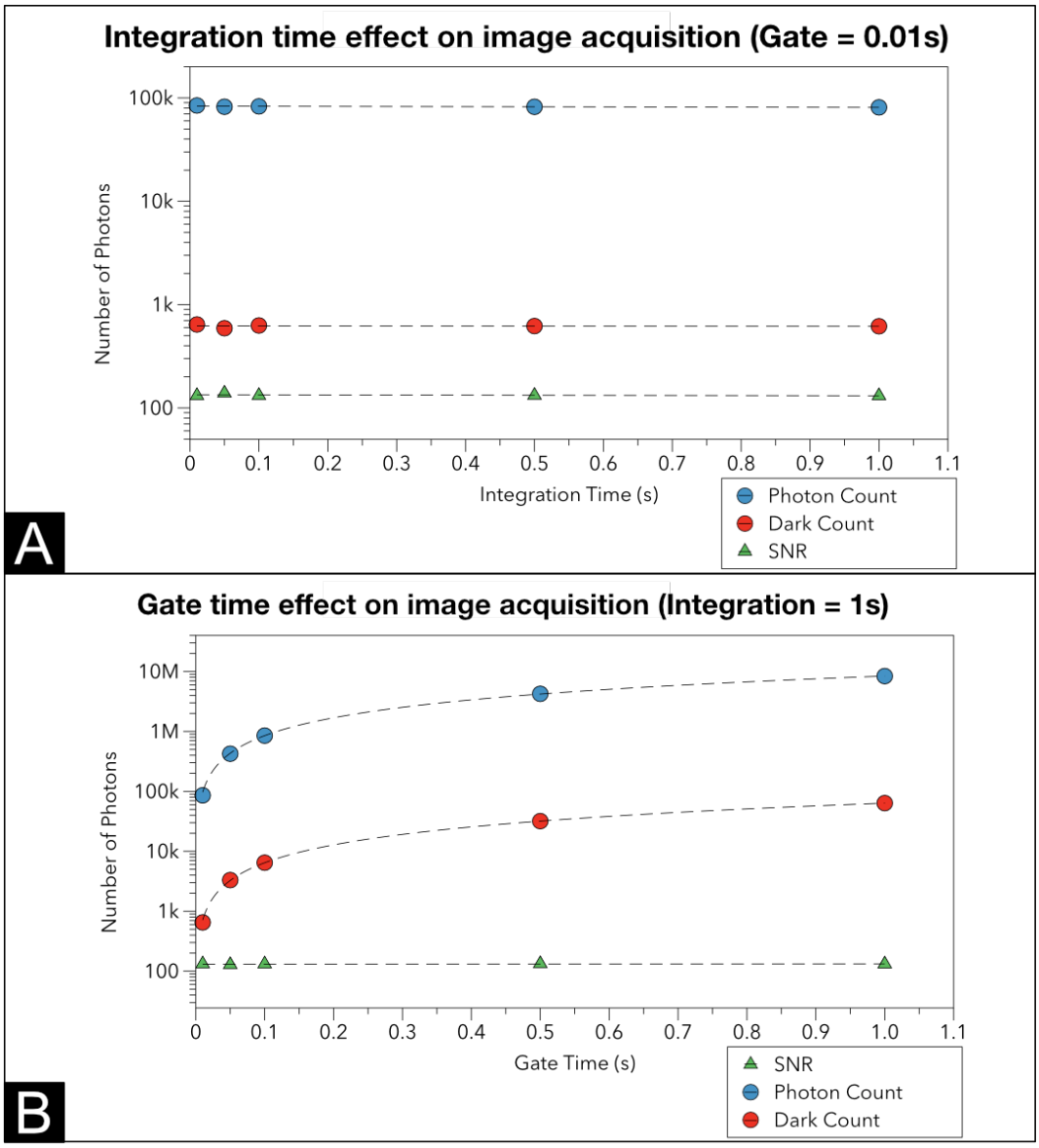

Figure 4.2: The effects of the integration (a) and gate (b) times on the overall signal to noise ratio.

parameters for subsequent acquisitions.

4.2 Raster and Basis Scan

Before moving to a compressive measurement, we first needed to characterize the image acquisition in the non-compressive modes, i.e., raster and basis scanning. With the gate and integration parameters defined as described in the last section (10ms and 100ms, respectively), several **NxN** square images with sizes ranging from 8x8px to 128x128px were capture using these modes (Figure 4.4).

Figure 4.4A compares the normalized image acquisition between the raster and basis scanning. Overall, the SNR seems dependent on both the image size and the scanning mode. In terms of the image resolution effect, the SNR seems to be inversely proportional to the acquisition size, with larger images appearing to have lower contrasts and more noise, which would make sense given that each binned pixels would be receiving less light. When comparing both modes, the raster scan leads to clearer images and less noise when compared to the other, but the SNR seems to decrease at a higher rate. When comparing the 128x128px image, we see that although the target is occluded in the raster scan measurement, it can still be distinguished in the Hadamard scan. Another issue in the Hadamard scan is the appearance of fringes on the image resembling interference patterns not present in the raster scan. One possible explanation for the appearance of the interference patterns is the laser light's coherence. As the light is reflected by different pixels within the SLM, the beams interfere with each other and cause the patterns to show. In the case of a raster scan measurement, as only one pixel (or binned set) is being used per iteration, the interference is decreased. Figure 4.5 shows the values of the linearized image (with the pixels of the image distributed along the x axis) array for all cases and how the noise floor changes as a function of each acquisition.

To better gauge how distinguishable the cross-shaped target is in these images, a binary segmentation was calculated using the Yen algorithm from the sci-kit image python library (Figure 4.4B). In these results, it's possible to see that this simple binarization allows, in most cases, to clearly distinguish the cross-shaped target, the only exception being the 128x128px raster scan image. Here again, we see that although the overall contrast generated in the raster scan measurement is greater, the Hadamard basis scan measurement scales better with the increase in resolution. Again, it's possible to distinguish clear interference patterns in the Hadamard scan, which are not present in

the other mode. These binary images show the promise of the SPI system, as additional post-processing steps could be implemented (e.g., morphological reconstruction and machine learning classification) to allow for detection of the target even with a small set of measurements.

Finally, when comparing the acquisition time, we see that the Hadamard basis scan takes longer, with a total acquisition time 2x the one found in the raster scan (Figure 4.6). This longer acquisition is expected as the differential Hadamard scan will carry out two sets of measurements (mask and its inverse) for each mask, whereas the raster scan will only acquire one.

4.3 Compressive Hadamard Scan

As discussed in Chapter 3, the Hadamard compressive sampling scan relies on the same measurement algorithm as the basis scan discussed in Section 4.2, differing only in the image restoration step. In this section, we'll demonstrate how the compressive sensing of the scene enables the reconstruction of the target's shape with fewer measurements and discuss how it compares with the other modes.

As introduced in section 3.2.1.3, the compressive scan works by acquiring $M \leq N^2$ measurements of the scene and applying an optimization algorithm to estimate the image vector. To allow a fair comparison with the results obtained in the basis scan measurement, the compressive reconstruction was applied to the set of encoders and associated photon counts acquired in the measurements in Section 4.2.

Code block 8 shows the reconstruction code used to compressively reconstruct the dataset from the basis scan measurement. In summary, given that the acquisition code is the same for both modes, a set of measurements M (and its respective masks) were selected from the N^2 measurements in the basis scan dataset and used as input to the reconstruction algorithm. Subsequently, the RMSE between the compressively reconstructed images and the basis scan image (equivalent to a sampling ratio of 100%) was calculated.

Code Block 8: Compressive Sampling Reconstruction

```
def spi_image_reconstruct(encoded, counts, pixel_size, sampling, verbose=True):
    N = int(pixel_size**2)
    M = int(pixel_size**2 * sampling)

    encoded_m = encoded[:M,:]
    counts_m = counts[:M].squeeze().reshape(M,1)
```

```
    vx = cvx.Variable((N,1))
    objective = cvx.Minimize(cvx.norm(vx, 1))
    constraints = [encoded_m@vx == counts_m]
    prob = cvx.Problem(objective, constraints)
    result = prob.solve(verbose=verbose)
    Xat2 = np.array(vx.value).squeeze()

    return Xat2.reshape(pixel_size, pixel_size)
```

Figure 4.7 shows how the sampling ratio affects the image acquisition as a function of the image size. It's possible to see both a dependency of the sampling ratio and image size in the image quality. As expected, the more measurements are taken into account, the closer the reconstructed image comes to the reference image. When comparing different resolutions, it seems that images with a larger number of pixels need fewer measurements to allow for image reconstruction. To provide a quantitative account, Figure 4.8 shows the calculated RMSE as a function of the sampling ratio and image size. The RMSE appears to decrease linearly as a function of the sampling ratio, while the image size reduces the number of measurements required to arrive at the same value.

To understand how these effects affect the final image quality and its capacity to distinguish the target, images with sizes ranging from 8x8px to 64x64px were reconstructed with sampling ratios between 0.1 and 1. Also, to provide a better account of the distinguishability of the target, the images were binarized using the same segmentation algorithm as in Section 4.2 (Figures 4.9 4.10, 4.11, 4.12).

Looking at the binarized images, it's possible to corroborate that larger images allow the target reconstruction with fewer measurements. While in the 8x8px image, a sampling ratio of 60-70% seems to be required to discern the target, in the 64x64px image, a ratio of 30-40% is enough.

Finally, when we consider the acquisition time, it decreases linearly as a function of the sampling ratio, as expected (Figure 4.13). Compared to raster scanning, a sampling ratio downwards or equal to 50% is necessary to make it faster than an equivalent raster scan. As we've shown in the images above, for images larger or equal to 16x16px, a sampling ratio between 40-50% is achievable, making the compressive sampling the faster approach. That said, it is expected that an increase in SNR and post-processing techniques would further decrease the sampling ratio necessary for image reconstruction, potentially allowing the compressive scan to be the faster approach in smaller images.

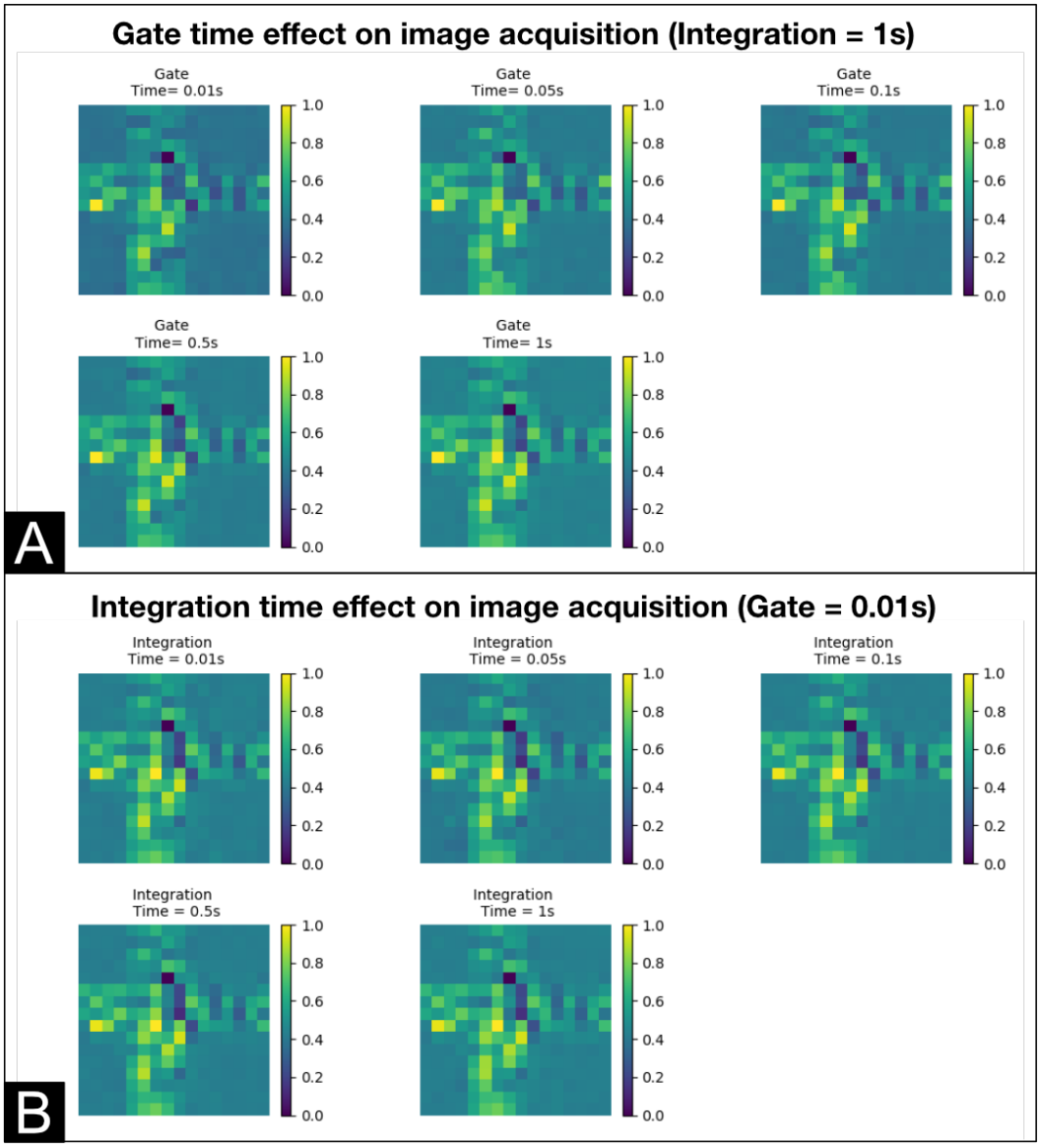

Figure 4.3: The effects of the integration (a) and gate (b) times on the image acquisition.

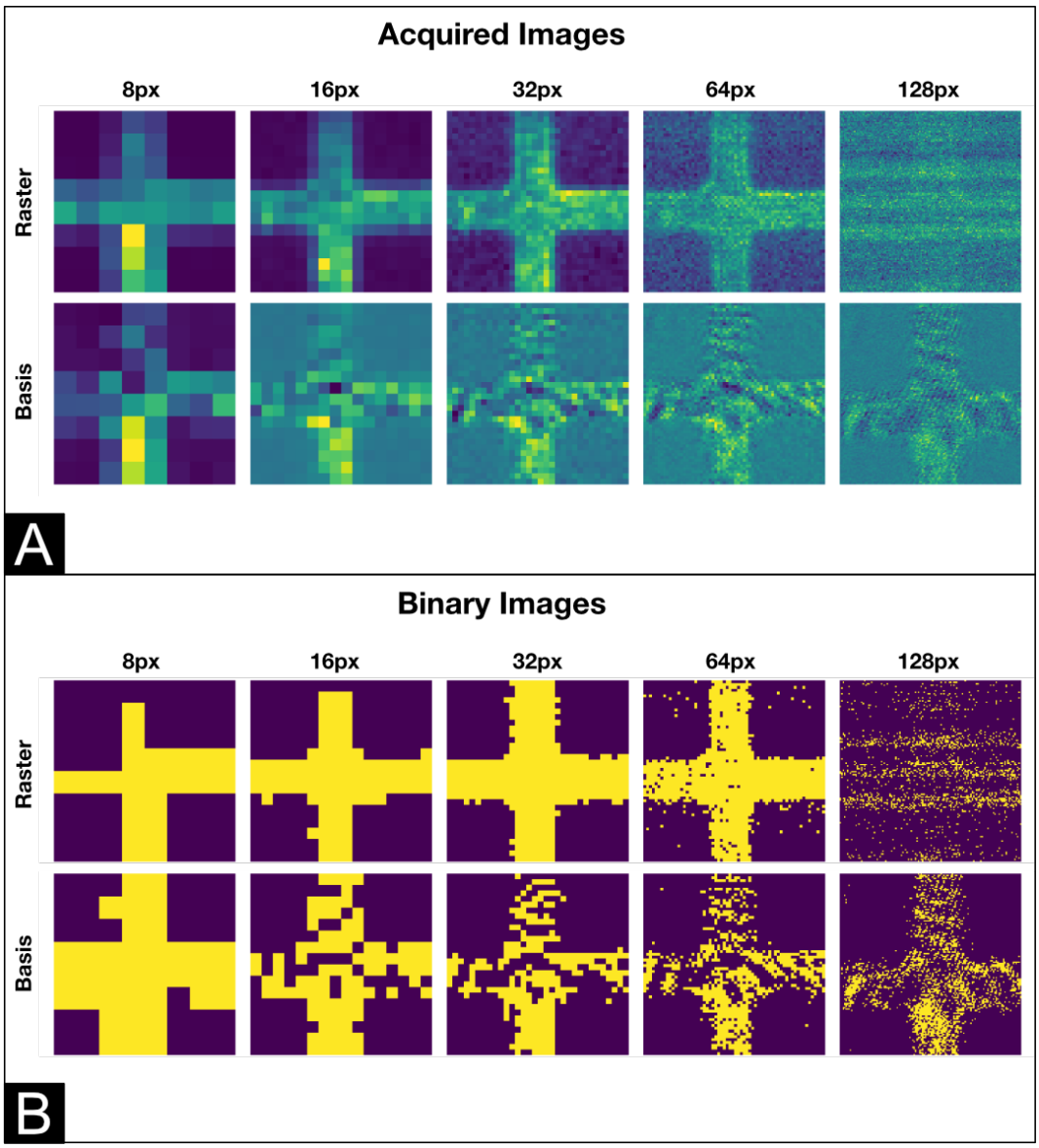

Figure 4.4: Comparison of the image acquisition from the raster and basis scanning modes. (a) shows the normalized image reconstruction. (b) Shows the output of the imaged after binarization with a Yen thresholding algorithm.

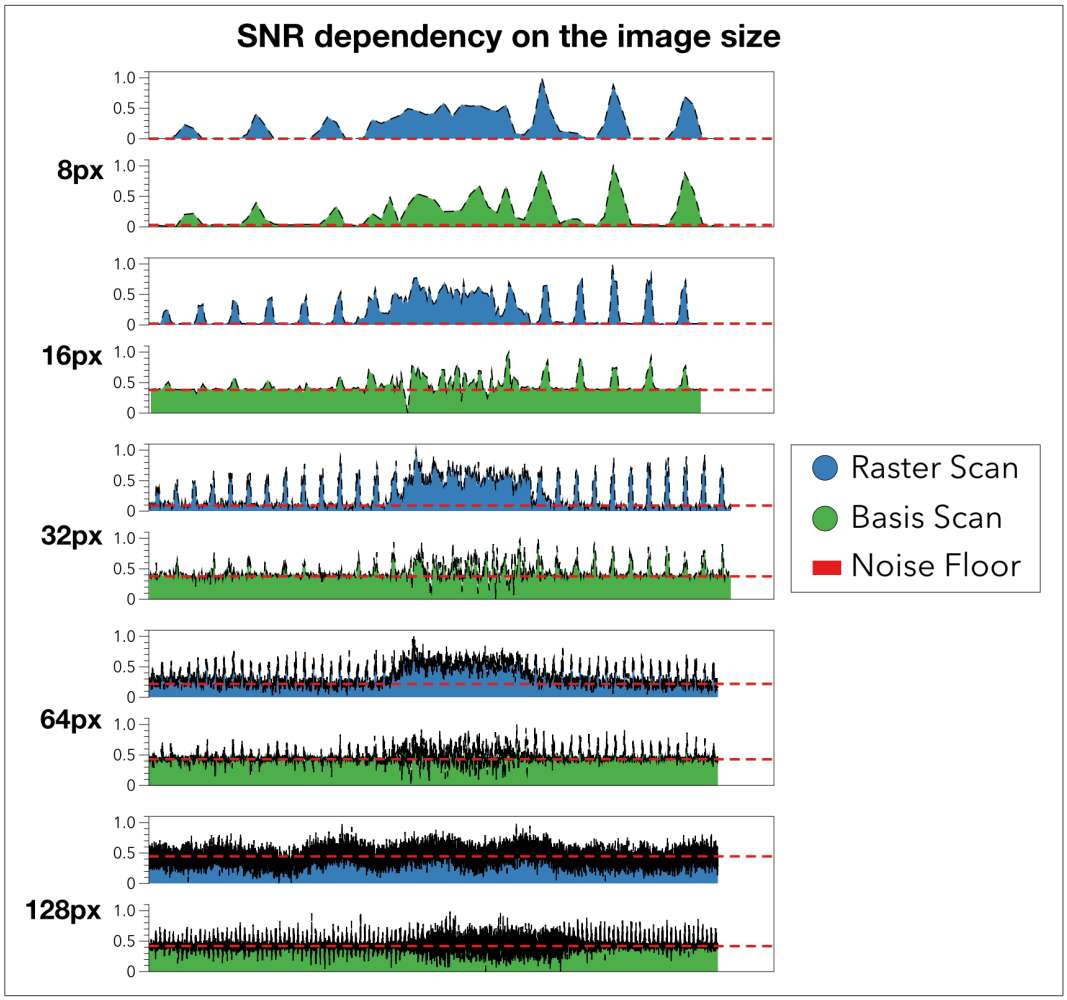

Figure 4.5: Comparison of the linearized (pixels of the image distributed along the x axis) intensity of the normalized images acquired from the raster and basis scanning modes and the respective noise floors.

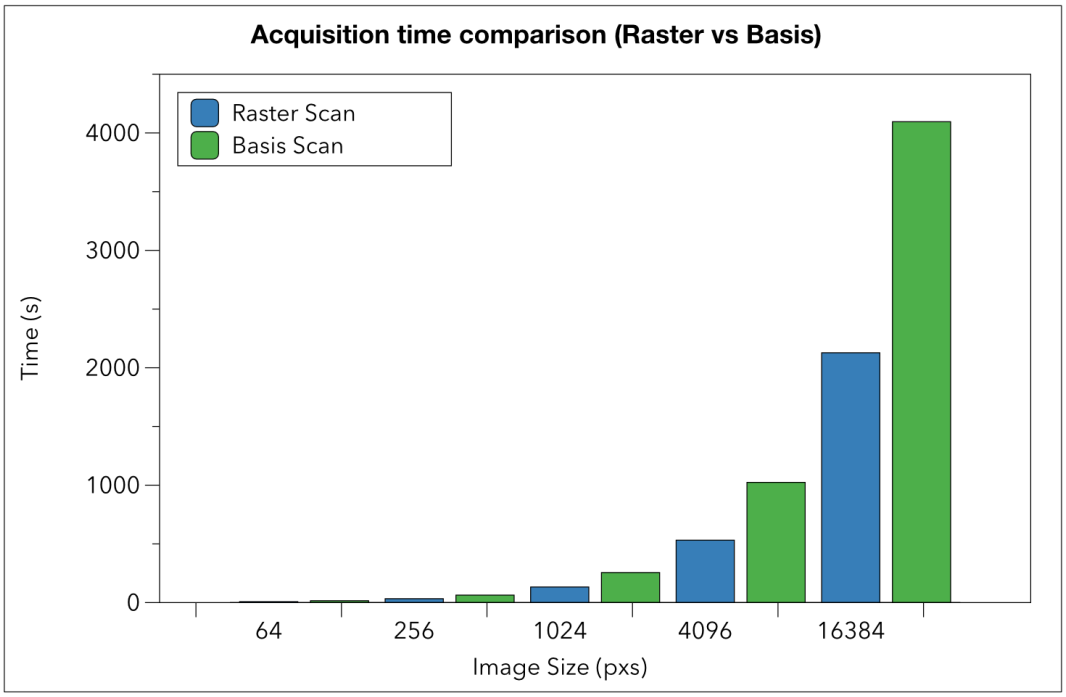

Figure 4.6: Comparison between the acquisition time from the raster and basis scanning modes for different image sizes.

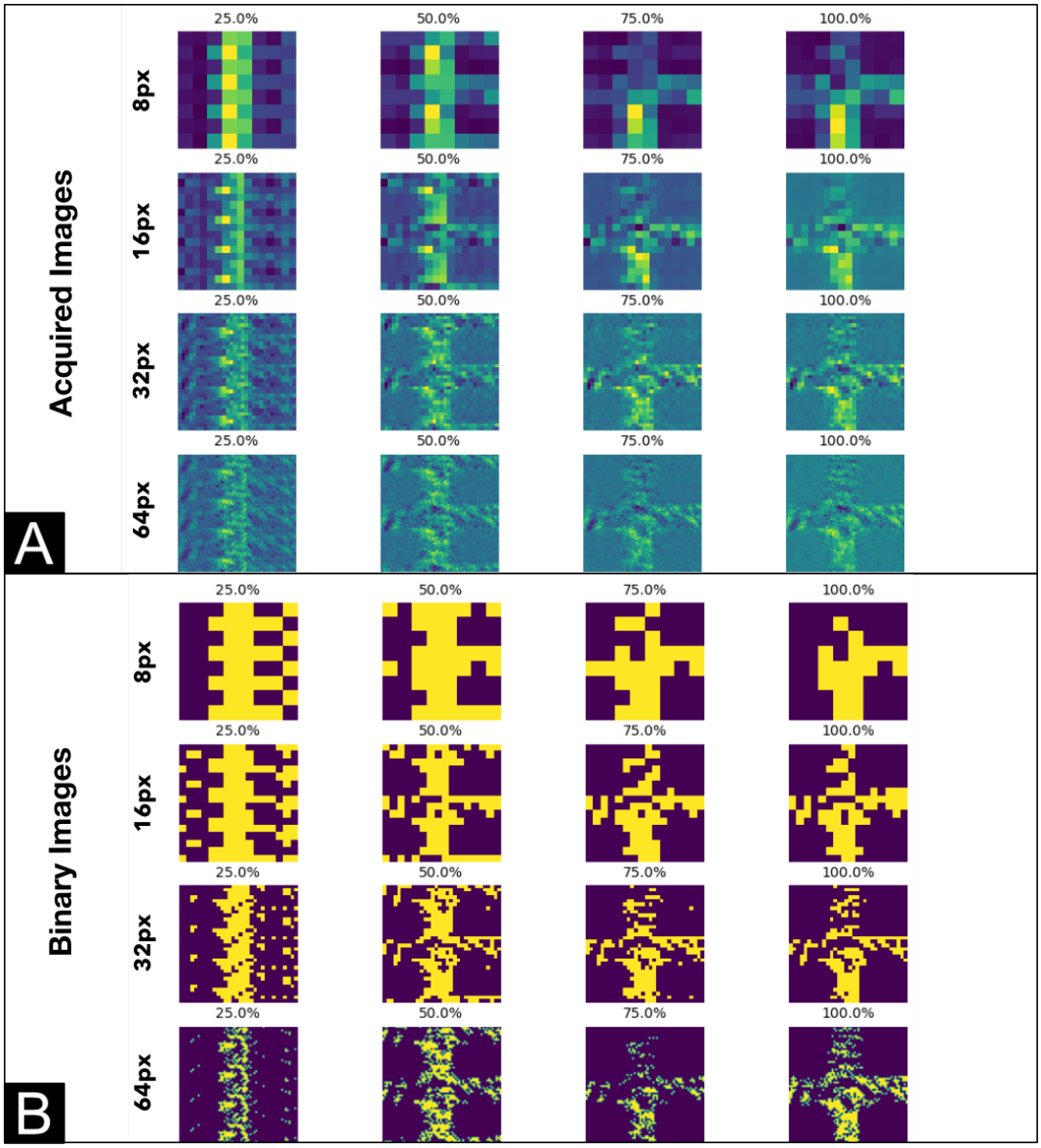

Figure 4.7: Effect of the sampling ratio on the image reconstruction from a compressive sampling acquisition mode (a) shows the normalized image reconstruction for different sampling rations. (b) Shows the output of the imaged after binarization with a Yen thresholding algorithm.

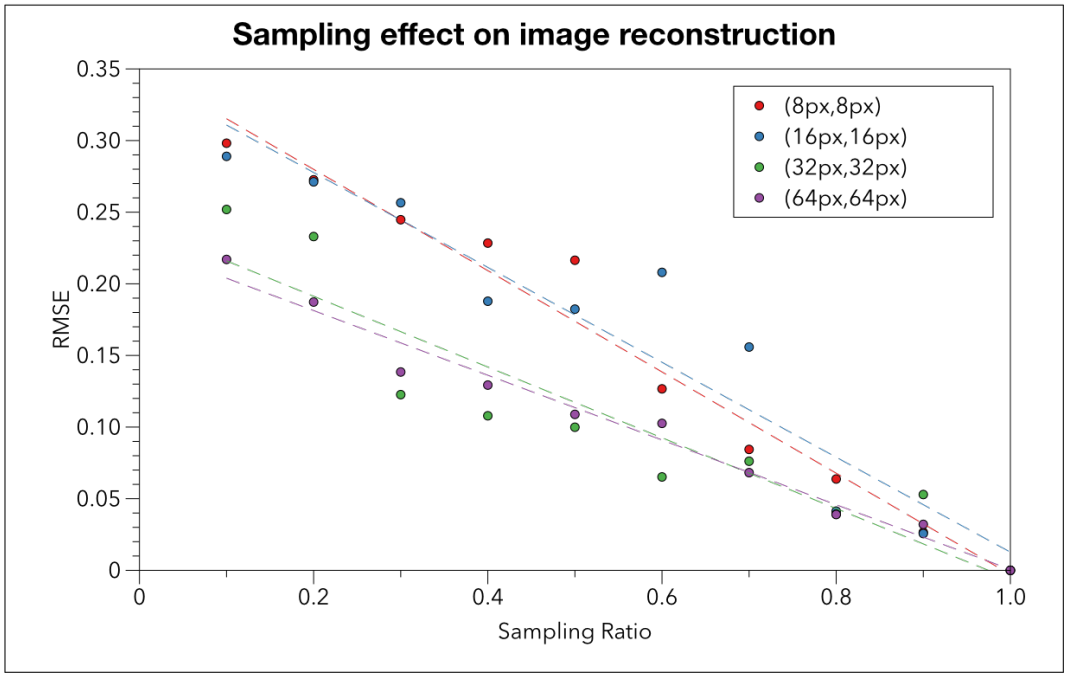

Figure 4.8: The RMSE of the image reconstruction as a function of the sampling ratio with respect to a 100% sampled image.

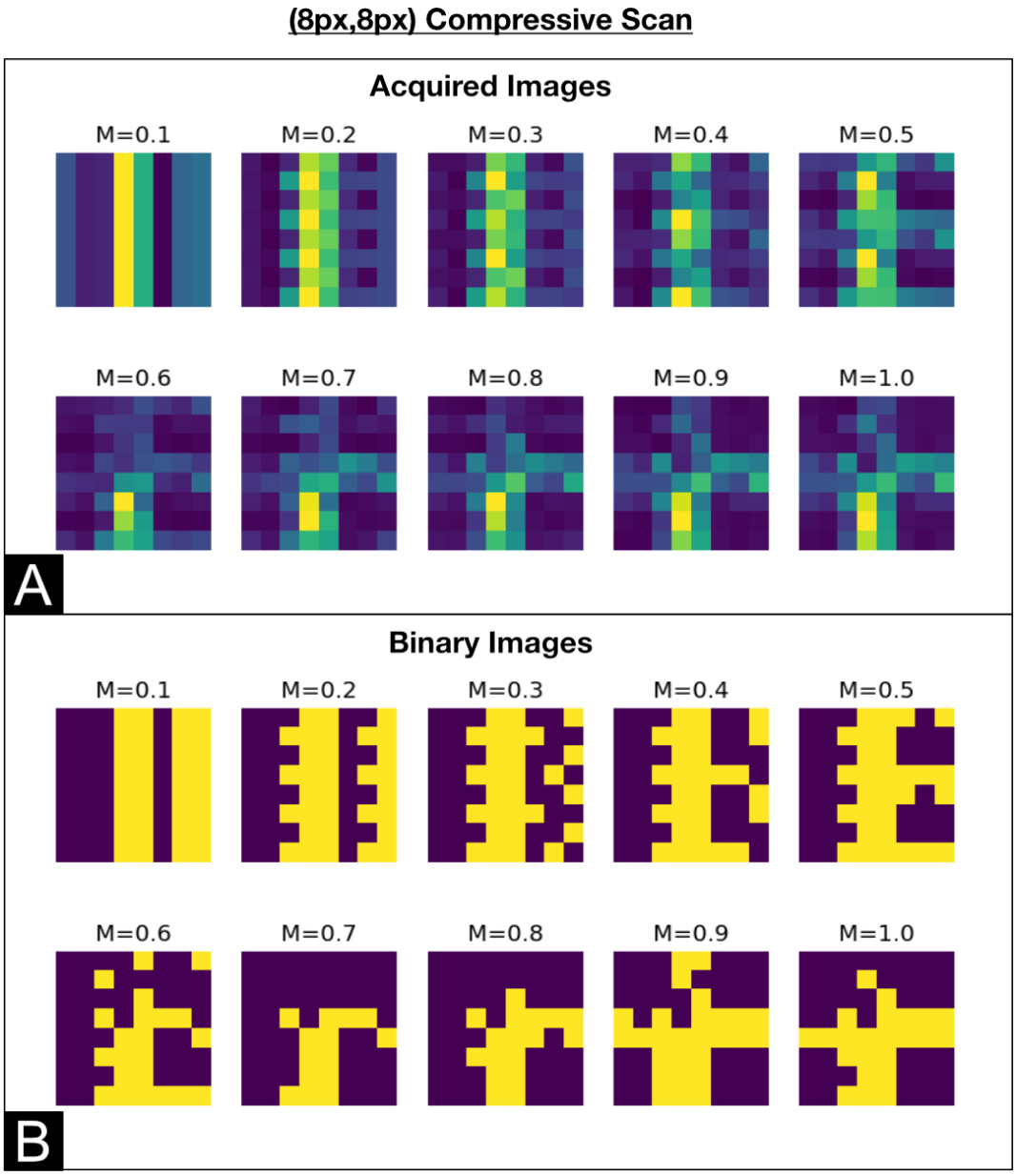

Figure 4.9: Effect of the sampling ratio on a (8px, 8px) image reconstruction from a compressive sampling acquisition mode (a) shows the normalized image reconstruction for different sampling ratios. (b) Shows the output of the imaged after binarization with a Yen thresholding algorithm.

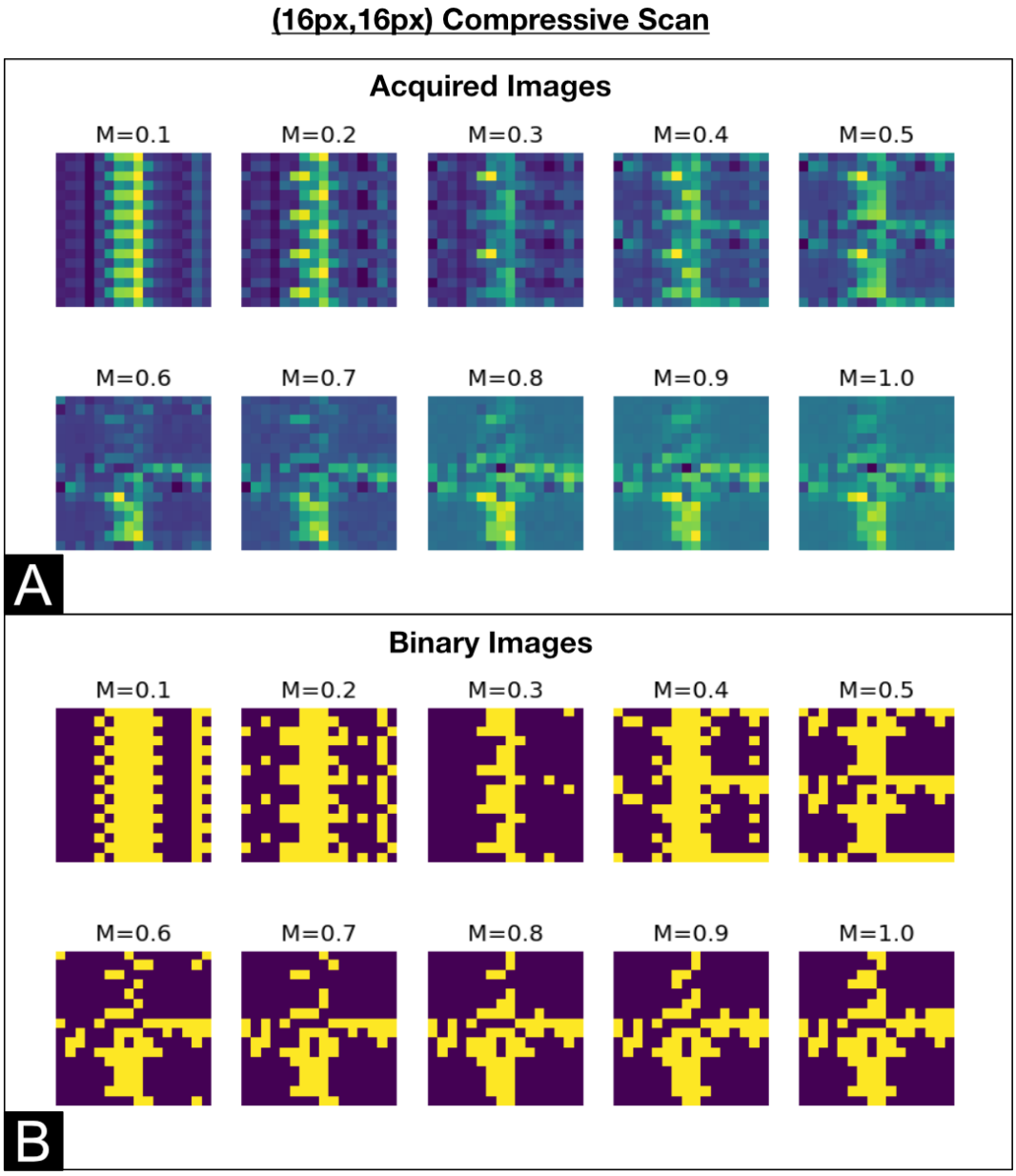

Figure 4.10: Effect of the sampling ratio on a (16px, 16px) image reconstruction from a compressive sampling acquisition mode (a) shows the normalized image reconstruction for different sampling ratios. (b) Shows the output of the imaged after binarization with a Yen thresholding algorithm.

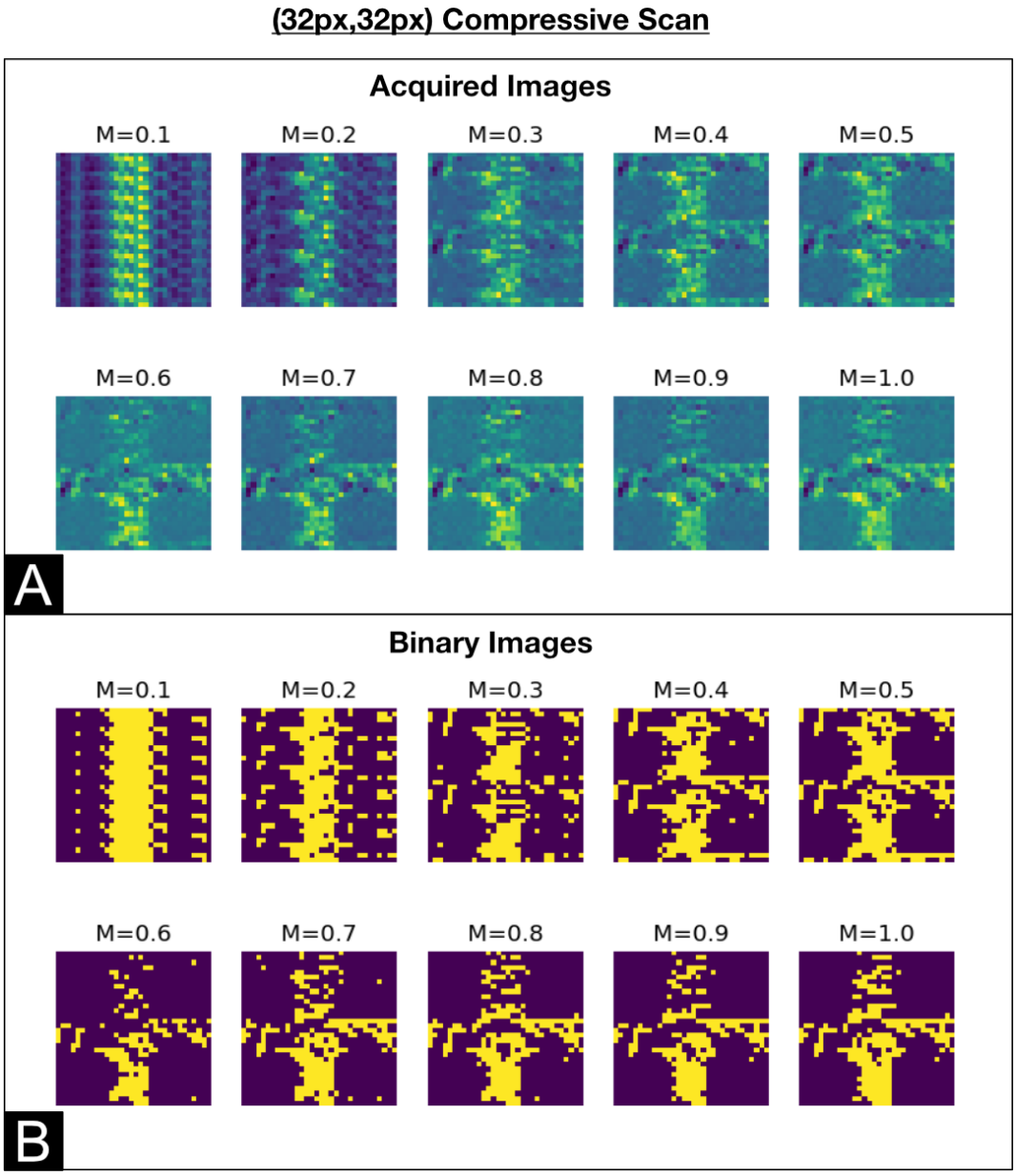

Figure 4.11: Effect of the sampling ratio on a (32px, 32px) image reconstruction from a compressive sampling acquisition mode (a) shows the normalized image reconstruction for different sampling ratios. (b) Shows the output of the imaged after binarization with a Yen thresholding algorithm.

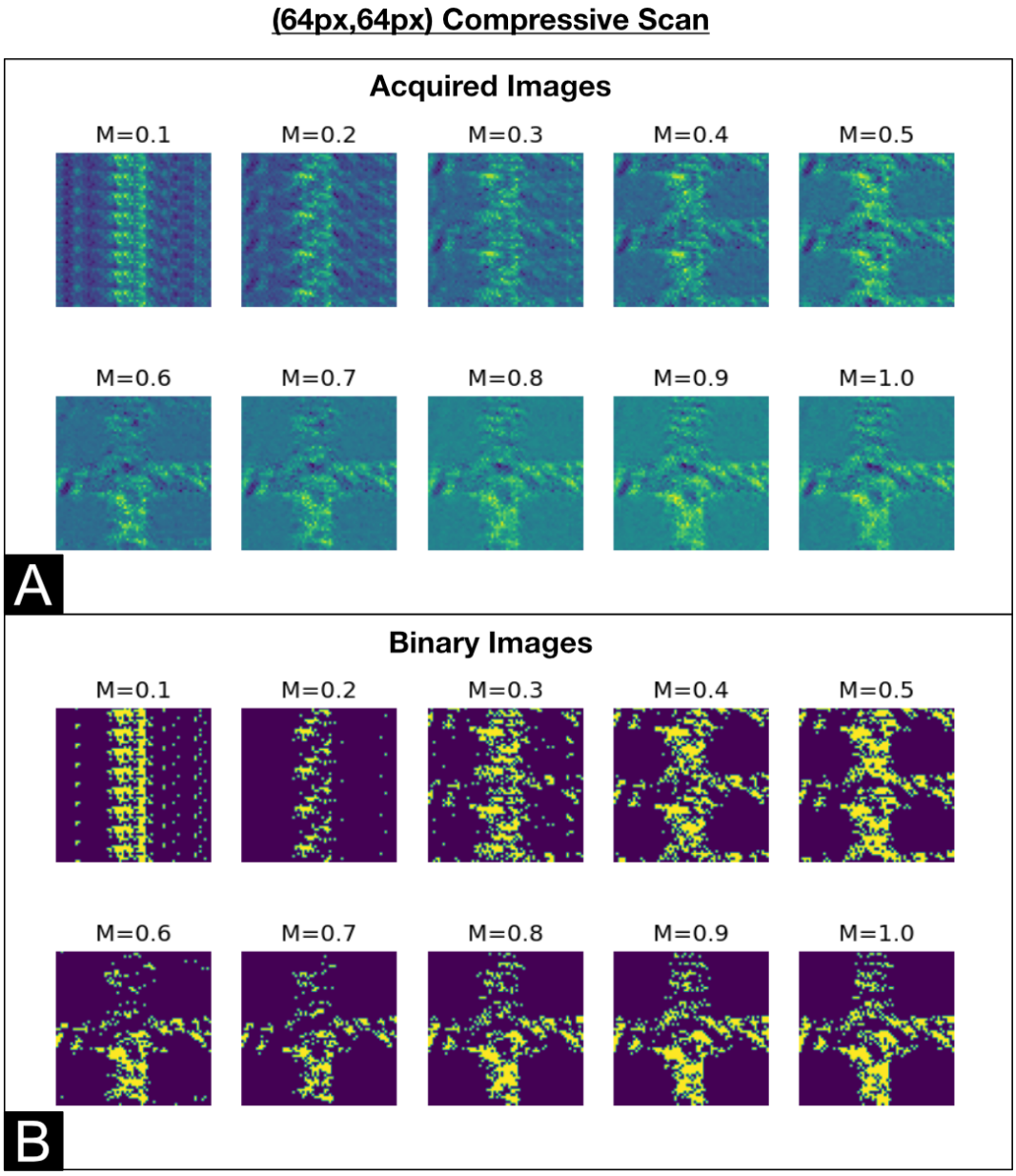

Figure 4.12: Effect of the sampling ratio on a (64px, 64px) image reconstruction from a compressive sampling acquisition mode (a) shows the normalized image reconstruction for different sampling ratios. (b) Shows the output of the imaged after binarization with a Yen thresholding algorithm.

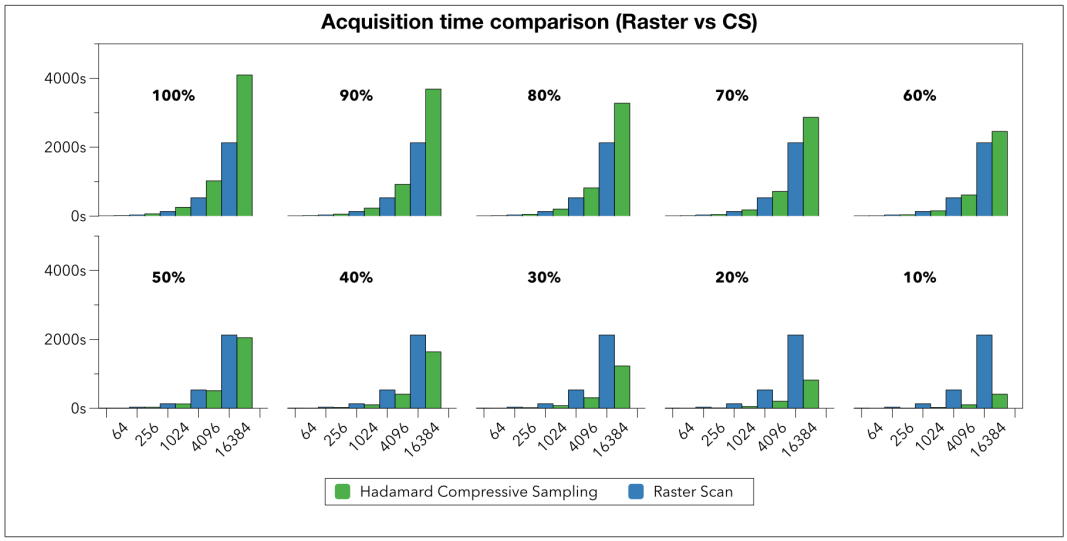

Figure 4.13: Comparison between the acquisition time from the raster and compressive scanning modes for different sampling ratios.

5
Conclusion

This study aimed to develop a proof of concept of a SPI system and create a framework that would allow this topic to be further studied. To that objective, we've discussed the theoretical foundations necessary to understand a SPI system, described and implemented an optical setup to carry out such measurements, and finally, developed an open-source python library for system control and automation.

After identifying the raster, basis, and compressive scanning modes as the fundamentals of the single-pixel image acquisition, we have implemented and demonstrated the use of all three modes for scene detection.

Although the raster scan measurement yields faster and less noisy results, the basis scanning mode allows for imaging in higher resolutions. When using a compressive sampling approach, it's possible to achieve faster image acquisition when compared with the raster scan for images sizes larger or equal to 16x16 px. Considering these results, the compressive sampling approach seems to be the best candidate for imaging higher resolution images in a relatively short time compared to the other modes. Here, we have also demonstrated how experimental parameters such as the integration time, gate time, sampling ratio, and image size affect image quality and object occlusion.

During this study, we have identified several areas that could be further developed to yield better results. In terms of the optical system, it became clear that the current implementation is limited by the overall SNR, and improvements in this area could significantly impact the compressive measurements, allowing scene reconstruction with fewer acquisitions. It was also discussed how improvements in the PySPI library could allow for further exploration of new encoding bases and post-processing techniques for scene reconstruction and classification through ML models.

Overall, this study aimed to provide the foundations of SPI systems without exhausting new possibilities in the optical and computational domains. We hope that the work discussed here can empower new research by decreasing the entry barrier for such an exciting and novel field, allowing further exploration of the SPI framework and its applications for new technologies.

6
Appendix A - Python Single Pixel Imaging Library (PySPI)

```
##########################################
## Python Single Pixel Imaging Library - pySPI
## Matheus Esteves Ferreira
## August 2020
## LAB-OPTO
## CETUC, PUC-RIO, Rio de Janeiro, Brazil
##########################################
##Importing Packages

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import cv2
from skimage import data
import skimage.transform
import scipy as spy
import os
import sys
import time
import visa
import tqdm
import scipy.linalg as la
import os
from PIL import Image
import random
import scipy.optimize as spopt
import scipy.fftpack as spfft
import scipy.ndimage as spimg
import cvxpy as cvx


#############################################################################
##Useful Functions

def savefig(img, normalize=False, format='png'):
    ##Saves a numpy array as an image with name equal to user input.
    if normalize == True:
        img = ((img - img.min())/(img.max() - img.min()))*255
    fig = Image.fromarray(img.astype('uint8'))
    fig.save('{}.{}'.format(input('Enter image name: '), format))

def Directory(message="Enter the directory name "):

```

```
    # Create directory ,it's a directory name which you are going to create.

    Directory_Name = input(message)
    #try and catch block use to handle the exceptions.
    try:
        # Create  Directory  MyDirectory
        os.mkdir(Directory_Name)
        #print if directory created successfully...
        print("Directory " , Directory_Name ,  " Created ")
    except FileExistsError:
        ##print if directory already exists...
        print("Directory " , Directory_Name ,  " already exists...")
    return Directory_Name

def save_measurement(encoded, counts, reconstruct, format='png'):
    cwd = os.getcwd()
    folder = Directory(message="Enter the measurement name ")
    np.save(cwd+'/'+folder+'/encoded_mask', encoded)
    np.save(cwd+'/'+folder+'/photon_count', counts)
    np.save(cwd+'/'+folder+'/reconstruct', reconstruct)

    fig = reconstruct
    fig = ((fig - fig.min())/(fig.max() - fig.min()))*255
    fig = Image.fromarray(fig.astype('uint8'))
    fig.save(cwd+'/'+folder+'/reconstruct.{}'.format(format))
    print('DONE!!!!!')

def RMSE(img1,img2):
    n = len(img1)
    dif = img1 - img2
    dif2 = dif ** 2
    rmse = np.sqrt(np.sum(dif2) / (n))
    return rmse

def normalize_img(img):
    return (img - img.min())/(img.max() - img.min())
##########################################################################
##Detector Programming - Agilent 53131A

class detector:
    def __init__(self):
        ##Initializes the Agilent Universal counter. Any default settings may be added here.
        rm = visa.ResourceManager()
        universal_Counter_GPIB = rm.list_resources()[0]
        self.Counter = rm.open_resource(universal_Counter_GPIB) # "GPIB0::4"
        self.Counter.timeout = 10000
        self.Counter.write(":INPut1:IMPedance 50")
        self.Counter.write(":INPut1:COUPling DC")
```

```
92          self.Counter.write(":EVENT1:LEVEL 2")
93
94      def measure_gate(self, gate, meas_time):
95          ##Initializes a Totalize measurement in Channel 1
96          for a total time equal meas_time and a geta time
97          equal gate.
98          self.Counter.write(":CONFigure:TOTalize:TIMed")
99          crr_time = time.time()
100         data = []
101         while (time.time() - crr_time) < meas_time:
102             data.append(float(self.Counter.query(":MEASure:TOTalize:TIMed? {}".format(gate))))
103         data = np.array(data)
104         data_stats = np.mean(data), np.std(data)
105         return data_stats
106
107     def measure_external(self, n=1):
108         ##Initializes a Totalize measurement in Channel 1 by using the optical chopper
109         frequency output as gate reference. This measurement will be repeated n times.
110         data = np.zeros(n)
111         for i in range(n):
112             self.Counter.write(":TOTalize:ARM:SOURce EXTernal")
113             self.Counter.write(":TOTalize:ARM:STOP:SOURce EXTernal")
114             self.Counter.write(":INITIATE")
115             data[i] = float(self.Counter.query(":FETCH?"))
116         return np.array([data.mean(), data.std()])
117
118 ##########################################################################
119 ##SLM Programming
120
121 class slm:
122
123     ############################UI Calls#####################################
124     def __init__(self, resolution = (600,800), k=255,
125      native_res=(800,600)):
126         self.resolution = resolution
127         self.native_res = native_res
128         self.slm_window = 'SPI Mask Generator'
129         cv2.namedWindow(self.slm_window, cv2.WINDOW_NORMAL)
130         mask = np.ones(self.resolution)*k
131         mask_show = cv2.resize(mask.astype('uint8'), self.native_res)
132         cv2.imshow(self.slm_window, mask_show.astype('uint8'))
133         cv2.waitKey(0)
134
135     def showImage(self, oriimg, W, H, native_res=(800,600)):
136       height, width= oriimg.shape
137       scaleWidth = float(W)/float(width)
138       scaleHeight = float(H)/float(height)
139       native_res = native_res[::-1]
```

```
140
141         if scaleHeight>scaleWidth:
142             imgScale = scaleWidth
143         else:
144             imgScale = scaleHeight
145
146         newX,newY = oriimg.shape[1]*imgScale, oriimg.shape[0]*imgScale
147         newimg = cv2.resize(oriimg,(int(newX),int(newY)))
148         projection = np.zeros(native_res)
149         projection[:int(newY), :int(newX)] = newimg
150         cv2.imshow(self.slm_window,projection.astype('uint8'))
151         cv2.waitKey(1)
152
153     def ShowArray(self, array, native_res=(600,800)):
154         out = np.zeros(native_res)
155         out[:array.shape[0], :array.shape[1]] = array
156         cv2.imshow(self.slm_window,out)
157         cv2.waitKey(1)
158
159     def exit(self):
160         cv2.destroyAllWindows()
161
162     ############################General Measurements#######################
163     def Calculate_Exp_Param(self, gate=.1, meas_time=.5):
164         ## Measures the photon_flux, dark_count for a given measurement time & gate.
165         pd = detector()
166         light_mask = np.ones(self.native_res[::-1]) * 255
167         dark_mask = np.zeros(self.native_res[::-1])
168
169         ## Measuring Photon Count
170         cv2.imshow(self.slm_window, light_mask.astype('uint8'))
171         cv2.waitKey(1)
172         photon_count = pd.measure_gate(gate, meas_time)[0]
173
174         ## Measuring Dark Count
175         cv2.imshow(self.slm_window, dark_mask.astype('uint8'))
176         cv2.waitKey(1)
177         dark_count = pd.measure_gate(gate, meas_time)[0]
178
179         self.photon_count = photon_count
180         self.dark_count = dark_count
181         self.meas_time = meas_time
182
183         return photon_count, dark_count, meas_time
184
185     def slm_greyscale_curve(self, k, gate=.1, meas_time=.5):
186         pd = detector()
187         encoded = np.ones((600, 800))
188         encoded[:,600:] = 0
```

```
189         counts = []
190 
191         for i in tqdm.tqdm_notebook(range(256)):
192             crr_mask = encoded*i
193             photon_count = np.empty(k)
194             cv2.imshow(self.slm_window, crr_mask.astype('uint8'))
195             cv2.waitKey(1)
196             for j in range(k):
197                 photon_count[j] = pd.measure_gate(gate, meas_time)[0]
198             counts.append(photon_count.mean(axis=0))
199 
200         return counts
201     ############################Raster Scanning#############################
202 
203     def generate_encoded_raster(self, n, idx, native_res=(600,800)):
204 
205         seed = np.zeros((n, n))
206         seed.ravel()[idx] = 1
207 
208         y_size = int(native_res[0]/n)
209         x_size = int(native_res[1]/n)
210         shape = min(y_size, x_size)
211 
212         encoded = np.repeat(seed.repeat(shape).reshape(n,n*shape), shape, axis=0)
213 
214         return encoded
215 
216     def raster_scan(self, n, gate=.1, meas_time=.5, Save=True):
217         ## Executes a raster scan by binning the total SLM resolution
218         by pixel_bin and measuring for a given gate and meas_time.
219 
220         pd = detector()
221         image = np.empty((n, n))
222         for i in tqdm.tqdm_notebook(range(n**2)):
223             crr_mask = self.generate_encoded_raster(n, i)*255
224             self.ShowArray(crr_mask)
225             image.ravel()[i] = pd.measure_gate(gate, meas_time)[0]
226 
227         if Save==True:
228             encoded = np.ones((n, n))
229             counts = image
230             save_measurement(encoded, counts, image)
231 
232 
233         return image
234 
235 
236     ############################Basis Scanning#############################
```

```
237     ######Auxiliary Functions#################################################
238     def generate_encoded_random(self,height,width):
239             encoded = np.random.choice([0,1],
240             p=[.5,.5], size=(height*width, height*width)).astype('float64')
241             if np.linalg.det(encoded) != 0:
242                 break
243         return encoded * 255
244 
245     def generate_encoded_hadamard(self, n, idx, encoded, native_res=(600,800)):
246 
247         seed = encoded[idx]
248 
249         y_size = int(native_res[0]/n)
250         x_size = int(native_res[1]/n)
251         shape = min(y_size, x_size)
252 
253         encoded = np.repeat(seed.repeat(shape).reshape(n,n*shape), shape, axis=0)
254 
255         return encoded
256 
257 
258     # def generate_encoded_hadamard_disk(n, idx, seed, native_res=(600,800)):
259 
260     #     seed = seed[idx]
261 
262     #     y_size = int(native_res[0]/n)
263     #     x_size = int(native_res[1]/n)
264     #     shape = min(y_size, x_size)
265 
266     #     encoded = np.repeat(seed.repeat(shape).reshape(n,n*shape), shape, axis=0)
267 
268     #     return encoded
269 
270 
271 
272 
273         ######Measurement Functions#######
274 
275     ######Measurement Functions#################################################
276 
277     def basis_scan_diff_hadamard_timed(self, n, gate=.1, meas_time=.5, Save=True):
278         pd = detector()
279         encoded = la.hadamard(n**2)
280         # np.save('hadamard_mask', encoded)
281         counts = []
282 
283         for i in tqdm.tqdm_notebook(range(n**2)):
284             crr_mask = ((self.generate_encoded_hadamard(n, i, encoded) -(-1))/(1-(-1)))*255
```

```
285
286             self.ShowArray(crr_mask)
287             photon_count = pd.measure_gate(gate, meas_time)[0]
288             # photon_count = np.random.random()*10000
289
290             self.ShowArray(1-crr_mask)
291             photon_count_inv = pd.measure_gate(gate, meas_time)[0]
292             # photon_count_inv = np.random.random()*10000
293
294             diff_count = photon_count - photon_count_inv
295             counts.append(diff_count)
296
297         encoded_mean = encoded.mean(axis=1).reshape(n**2,1)
298         diff_encoded = encoded - encoded_mean
299         diff_counts = np.array(counts).reshape(n**2, 1)
300         diff_counts = diff_counts - diff_counts.mean()
301         reconstruct = np.dot(diff_encoded, diff_counts).reshape(n, n)
302
303         # np.save('photon_count', diff_counts)
304
305         if Save==True:
306             save_measurement(encoded, diff_counts, reconstruct)
307
308         return reconstruct
309
310     def basis_scan_diff_hadamard_external(self, n, k, Save=True):
311         pd = detector()
312         encoded = la.hadamard(n**2)
313         # np.save('hadamard_mask', encoded)
314         counts = []
315
316         for i in tqdm.tqdm_notebook(range(n**2)):
317             crr_mask = ((self.generate_encoded_hadamard(n, i, encoded) -(-1))/(1-(-1)))*255
318
319             self.ShowArray(crr_mask)
320             photon_count = pd.measure_external(n=k)[0]
321             # photon_count = np.random.random()*10000
322
323             self.ShowArray(1-crr_mask)
324             photon_count_inv = pd.measure_external(n=k)[0]
325             # photon_count_inv = np.random.random()*10000
326
327             diff_count = photon_count - photon_count_inv
328             counts.append(diff_count)
329
330         encoded_mean = encoded.mean(axis=1).reshape(n**2,1)
331         diff_encoded = encoded - encoded_mean
332         diff_counts = np.array(counts).reshape(n**2, 1)
333         diff_counts = diff_counts - diff_counts.mean()
```

```
334         reconstruct = np.dot(diff_encoded, diff_counts).reshape(n, n)
335 
336         # np.save('photon_count', diff_count)
337         if Save==True:
338             save_measurement(encoded, diff_counts, reconstruct)
339 
340         return reconstruct
341 
342     def compressive_scan_diff_hadamard_timed(self, n, m, gate=.1, meas_time=.5, Save=True):
343             pd = detector()
344             encoded = la.hadamard(n**2)
345             counts = []
346             encoded_m = encoded[:M,:]
347 
348             for i in tqdm.tqdm_notebook(range(m)):
349                 crr_mask = ((self.generate_encoded_hadamard(n,
350                 i, encoded) -(-1))/(1-(-1)))*255
351 
352                 self.ShowArray(crr_mask)
353                 photon_count = pd.measure_gate(gate, meas_time)[0]
354                 # photon_count = np.random.random()*10000
355 
356                 self.ShowArray(1-crr_mask)
357                 photon_count_inv = pd.measure_gate(gate, meas_time)[0]
358                 # photon_count_inv = np.random.random()*10000
359 
360                 diff_count = photon_count - photon_count_inv
361                 counts.append(diff_count)
362 
363             encoded_mean = encoded_m.mean(axis=1).reshape(m,1)
364             diff_encoded = encoded_m - encoded_mean
365             diff_counts = np.array(counts).reshape(m, 1)
366             diff_counts = diff_counts - diff_counts.mean()
367             diff_counts_m = diff_counts.squeeze().reshape(m, 1)
368 
369 
370 
371             try:
372                 vx = cvx.Variable((n**2,1))
373                 objective = cvx.Minimize(cvx.norm(vx, 1))
374                 constraints = [encoded_m*vx == diff_counts_m]
375                 prob = cvx.Problem(objective, constraints)
376                 result = prob.solve(verbose=True)
377                 reconstruct = np.array(vx.value).squeeze().reshape(n,n)
378             except:
379                 print('Could not Reconstruct')
380 
381             if Save==True:
```

```
382                 save_measurement(encoded_m, diff_counts_m, reconstruct)
383
384             return reconstruct
385
386     def compressive_scan_diff_hadamard_external(self, n, m, k, Save=True):
387             pd = detector()
388             encoded = la.hadamard(n**2)
389             idx = list(np.arange(n**2))
390             idx = np.array(random.sample(idx, m))
391             idx.sort()
392             counts = []
393             encoded_m = encoded[idx,:]
394
395             for i in tqdm.tqdm_notebook(range(m)):
396                 crr_mask = ((self.generate_encoded_hadamard(n,
397                 idx[i], encoded) -(-1))/(1-(-1)))*255
398
399                 self.ShowArray(crr_mask)
400                 photon_count = pd.measure_external(n=k)[0]
401                 # photon_count = np.random.random()*10000
402
403                 self.ShowArray(1-crr_mask)
404                 photon_count_inv = pd.measure_external(n=k)[0]
405                 # photon_count_inv = np.random.random()*10000
406
407                 diff_count = photon_count - photon_count_inv
408                 counts.append(diff_count)
409
410             encoded_mean = encoded_m.mean(axis=1).reshape(m,1)
411             diff_encoded = encoded_m - encoded_mean
412             diff_counts = np.array(counts).reshape(m, 1)
413             diff_counts = diff_counts - diff_counts.mean()
414             diff_counts_m = diff_counts.squeeze().reshape(m, 1)
415
416
417
418             try:
419                 vx = cvx.Variable((n**2,1))
420                 objective = cvx.Minimize(cvx.norm(vx, 1))
421                 constraints = [encoded_m*vx == diff_counts_m]
422                 prob = cvx.Problem(objective, constraints)
423                 result = prob.solve(verbose=True)
424                 reconstruct = np.array(vx.value).squeeze().reshape(n,n)
425             except:
426                 print('Could not Reconstruct')
427
428             if Save==True:
429                 save_measurement(encoded_m, diff_counts_m, reconstruct)
```

```
430
431             return reconstruct
```

7
References

[1] VANGINDERTAEL, J.; CAMACHO, R.; SEMPELS, W.; MIZUNO, H.; DEDECKER, P. ; JANSSEN, K. P.. **An introduction to optical super-resolution microscopy for the adventurous biologist**. Methods and Applications in Fluorescence, 6, 3 2018. 1

[2] ALTMANN, Y.; PADGETT, M. J.. **Non-line-of-sight 3d imaging with a single-pixel camera**. 2019. 1

[3] SUN, B.; EDGAR, M. P.; BOWMAN, R.; VITTERT, L. E.; WELSH, S.; BOWMAN, A. ; PADGETT, M. J.. **3d computational imaging with single-pixel detectors**. Science, 340:844–847, 5 2013. (document), 1, 2.14, 2.2.4.2

[4] LI, Z.; SUO, J.; HU, X.; DENG, C.; FAN, J. ; DAI, Q.. **Efficient single-pixel multispectral imaging via non-mechanical spatio-spectral modulation**. Scientific Reports, 7, 1 2017. (document), 1, 2.2.4, 2.2.4.1, 2.13

[5] ASPDEN, R. S.; GEMMELL, N. R.; MORRIS, P. A.; TASCA, D. S.; MERTENS, L.; TANNER, M. G.; KIRKWOOD, R. A.; RUGGERI, A.; TOSI, A.; BOYD, R. W.; BULLER, G. S.; HADFIELD, R. H. ; PADGETT, M. J.. **Photon-sparse microscopy: Trans-wavelength ghost imaging**. volumen 9824, p. 982407. SPIE, 5 2016. (document), 1, 2.2, 2.1.1

[6] PADGETT, M. J.; BOYD, R. W.. **An introduction to ghost imaging: Quantum and classical**. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 375, 8 2017. (document), 2.1, 2.1, 2.1.1, 2.1.2, 2.3

[7] GIBSON, G. M.; JOHNSON, S. D. ; PADGETT, M. J.. **Single-pixel imaging 12 years on: a review**. Optics Express, 28:28190, 9 2020. (document), 2.1, 2.4, 2.2.1.1, 2.2.1.4, 2.2.2.1, 2.10, 2.2.2.2, 2.2.2.2, 2.2.2.2, 2.11, 2.2.2.3, 2.2.2.3, 2.16

[8] SHAPIRO, J. H.. **Computational ghost imaging**. 7 2008. 2.1.2

[9] ZHANG, A. X.; HE, Y. H.; WU, L. A.; CHEN, L. M. ; WANG, B. B.. **Table-top x-ray ghost imaging with ultra-low radiation**, 9 2017. (document), 2.2.1.1, 2.2.1.1, 2.5

[10] DUARTE, M. F.; DAVENPORT, M. A.; TAKHAR, D.; LASKA, J. N.; SUN, T.; KELLY, K. F. ; BARANIUK, R. G.. **Single-pixel imaging via compressive sampling**. IEEE Signal Processing Magazine, 25:83–91, 3 2008. (document), 2.9, 2.2.3.3, 2.2.3.3, 2.2.3.3, 2.2.3.3

[11] DIAMOND, S.; BOYD, S.. **CVXPY: A Python-embedded modeling language for convex optimization**. Journal of Machine Learning Research, 17(83):1–5, 2016. 2.2.3.3

[12] AGRAWAL, A.; VERSCHUEREN, R.; DIAMOND, S. ; BOYD, S.. **A rewriting system for convex optimization problems**. Journal of Control and Decision, 5(1):42–60, 2018. 2.2.3.3

[13] JIAO, S.; FENG, J.; GAO, Y.; LEI, T. ; YUAN, X.. **Visual cryptography in single-pixel imaging**, 2019. 2.2.4

[14] GIBSON, G. M.; SUN, B.; EDGAR, M. P.; PHILLIPS, D. B.; PADGETT, M. J.; HEMPLER, N.; MAKER, G. T. ; MALCOLM, G. P.. **Real-time imaging of methane gas leaks using a single-pixel camera**, 10 2017. (document), 2.12, 2.2.4.1

[15] SATAT, G.; TANCIK, M. ; RASKAR, R.. **Lensless imaging with compressive ultrafast sensing**. IEEE Transactions on Computational Imaging, 3:398–407, 3 2017. (document), 2.15, 2.2.4.3

[16] PYVISA AUTHORS. **PyVISA documentation**, 2018. 3.1.4

[17] SALEH, B. E. A.; TEICH, M. C.. **Fundamentals of photonics; 2nd ed.** Wiley series in pure and applied optics. Wiley, New York, NY, 2007. (document), 2.6, 2.7, 2.8

[18] JOHNSON, S. D.; MOREAU, P. A.; GREGORY, T. ; PADGETT, M. J.. **How many photons does it take to form an image?**, 6 2020.

[19] FERRI, F.; MAGATTI, D.; LUGIATO, L. A. ; GATTI, A.. **Differential ghost imaging**. Physical Review Letters, 104, 6 2010.

[20] ROSALES-GUZMÁN, C.; FORBES, A.. **How to shape light with spatial light modulators**. 2017.

[21] ALTMANN, Y.; PADGETT, M. J.. **Non-line-of-sight 3d imaging with a single-pixel camera**. 2019.

[22] EDGAR, M. P.; GIBSON, G. M. ; PADGETT, M. J.. **Principles and prospects for single-pixel imaging**. Nature Photonics, 13:13–20, 2019.

[23] ROUSSET, F.. **Single-pixel imaging: development and applications of adaptive methods**.

[24] GONZALEZ, R. C.; WOODS, R. E.. **Digital image processing**. Pearson, 2018.

[25] ERKMEN, B. I.; SHAPIRO, J. H.. **Ghost imaging: from quantum to classical to computational**. Advances in Optics and Photonics, 2:405, 12 2010. 2.1.1