Neural Sensors: Learning Pixel Exposures for HDR Imaging and Video Compressive Sensing with Programmable Sensors

Julien N. P. Martel, *Member, IEEE,* and L. K. Müller, and S. J. Carey, and P. Dudek, *Senior Member, IEEE*, and G. Wetzstein *Senior Member, IEEE*

Abstract—Camera sensors rely on global or rolling shutter functions to expose an image. This fixed function approach severely limits the sensors' ability to capture high-dynamic-range (HDR) scenes and resolve high-speed dynamics. Spatially varying pixel exposures have been introduced as a powerful computational photography approach to optically encode irradiance on a sensor and computationally recover additional information of a scene, but existing approaches rely on heuristic coding schemes and bulky spatial light modulators to optically implement these exposure functions. Here, we introduce neural sensors as a methodology to optimize per-pixel shutter functions jointly with a differentiable image processing method, such as a neural network, in an end-to-end fashion. Moreover, we demonstrate how to leverage emerging programmable and re-configurable sensor–processors to implement the optimized exposure functions directly on the sensor. Our system takes specific limitations of the sensor into account to optimize physically feasible optical codes and we evaluate its performance for snapshot HDR and high-speed compressive imaging both in simulation and experimentally with real scenes.

Index Terms—high-dynamic range imaging, video compressive sensing, high-speed imaging, programmable sensors, vision chip, deep neural networks, end-to-end optimization

1 Introduction

MOST contemporary digital cameras rely on a common working principle inherited from their analog ancestors: They capture an image by exposing photo-sensitive elements for a fixed exposure time. In modern digital sensors, this is either implemented using a global or a rolling shutter. In either case, the fixed exposure time of current sensors severely limits their ability to record natural scenes that exhibit a high dynamic range (HDR) or fast motion.

Computational photography has enabled us to overcome some of these challenges using optical coding strategies and computational image reconstruction. For example, HDR images can be recovered from multiple different exposures [1], [2] or from a single image with spatially varying pixel exposure [3], [4], [5]. Similarly, high-speed video can be estimated from a single image recorded with a per-pixel coded exposure [6], [7], [8], [9], [10]. However, these techniques have two drawbacks. First, they usually employ either heuristic or random optical coding strategies, which are suboptimal. Second, they typically require a high-speed spatial light modulator (SLM) to implement the optical coding. SLMs are expensive, they create bulky device form factors when integrated into an imaging system, and it is challenging to precisely align them at the required accuracy with respect to the sensor. To mitigate these shortcomings, specialized sensors have been designed for applications in HDR [11], [12], [13] or high-speed imaging [14], [15]. However, these devices cannot be easily reconfigured for other applications.

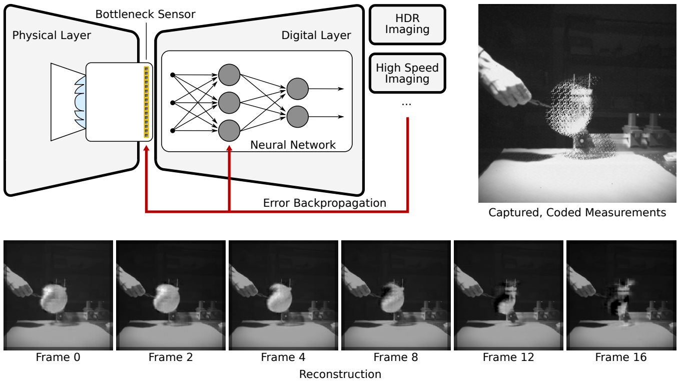

Fig. 1. **Illustration of our end-to-end neural sensor framework.** The exposure program of a sensor (physical layer) is learned end-to-end with a decoder (digital layer) for applications like video compressive sensing. Here, we show a single coded exposure captured with our prototype camera and several frames of the high-speed video reconstructed from this image showing an exploding balloon.

Here, we propose an end-to-end optimization strategy for jointly learning spatially varying pixel exposures and neural network–based image reconstruction algorithms for HDR and high-speed imaging. Rather than implementing these optical codes using SLMs, we build on emerging focal-plane sensor–processors [16], [17] that offer simultaneous sensing and processing capabilities in each pixel. These massively parallel, fine-grain processing capabilities allow us to control the specific exposure code electronically in each pixel. As illustrated in Figures 1 and 2, our system

- *J. N. P. Martel and G. Wetzstein are with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305, U.S.A.*
E-mail: jnmartel@stanford.edu
- *L. K. Müller is with IBM Research Zürich, Rüschlikon, 8803 ZH, Switzerland*
- *S. J. Carey and P. Dudek are with the School of Electrical and Electronic Engineering, The University of Manchester, Manchester, M13-9PL, U.K.*

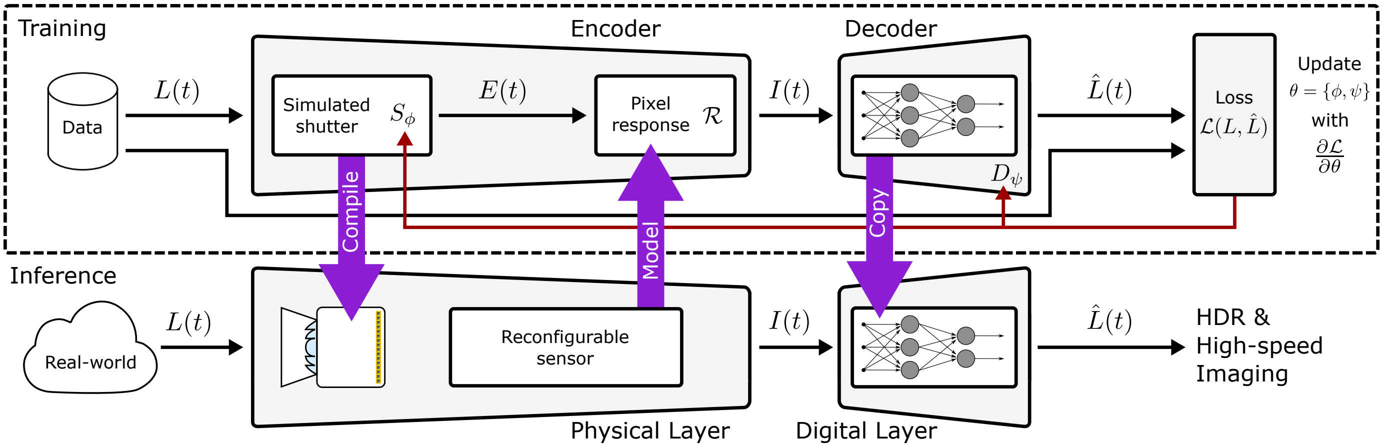

Fig. 2. **Diagram of the proposed neural sensor system.** During the training phase (top row) an electronic encoder—spatially varying pixel exposures parameterized by ϕ—is jointly optimized with a digital decoder—a neural network parameterized by ψ—in simulation using an application-specific loss function $\mathcal{L}$. The optimized shutter functions are compiled into a pixel code that is uploaded on the programmable sensor. During inference (bottom row), the sensor captures images by applying these exposure functions and the pre-trained network recovers the final image.

learns the optical coding strategies for a given application in simulation. This training phase accounts for the specific restrictions of a sensor–processor, which may limit the degrees of freedom of a feasibly shutter function. After training, the optical codes are compiled into an instruction set that is uploaded onto the reconfigurable sensor, which executes these shutter functions during inference. Our approach can be interpreted as an optical encoder, digital decoder system where the sensor implements the physical coding layer and a differentiable algorithm represents the digital decoder.

With this end-to-end neural sensor approach, we make the following contributions:

- We introduce the idea of learning the "sensing" strategy of a camera in an end-to-end fashion. Specifically, we present a differentiable neural sensor model to jointly optimize spatially varying pixel exposures and image processing networks.
- We demonstrate that this learned optical coding compares favorably to baselines with random coding for video compressive sensing and HDR imaging.
- We implement a prototype focal-plane sensor–processor with learned pixel exposures and we evaluate it with the aforementioned applications.

2 Related Work

HDR Imaging and Video Compressive Sensing: The limited dynamic range of conventional active pixel sensors has been addressed by a number of computational photography approaches. For example, several low-dynamic-range images can be recorded and computationally fused into a single HDR image [1], [2], [18], [19], [20]. However, capturing multiple images sequentially or with multiple sensors has several drawbacks, including challenges in dealing with motion and system calibration. As an alternative, several single-shot approaches have been introduced [3], [4], [5], [21], [22], [23], [24]. These systems either use external spatial light modulators (SLMs), neutral density (ND) filter arrays on the sensor, or varying ISO modulation to control the pixel exposures. A post-capture HDR image reconstruction step is required for most of these approaches. Unfortunately, none of these methods seems practical because SLMs are expensive and difficult to integrate robustly into an imaging system, ND filters require the sensor to be altered permanently, and spatially varying ISO is only available with very limited types exposure patterns. High-framerate compressive imaging and compressive video sensing have also been widely studied [6], [7], [8], [9], [10], [25], [26], [27], [28], [29], [30], [31] and prototyped by implementing high-speed coded exposures using various types of SLMs. Inspired by all of these methods, we propose an end-to-end framework that allows us to jointly optimize the spatially varying pixel exposures and reconstruction algorithms of HDR or high-speed imaging. Compared to the random or heuristic codes used in previous work, we demonstrate that optimized coding strategies for these applications significantly improve the image quality.

The closest approach to our proposal is the concurrent work by Iliadis et al. [32]. This approach also uses a neural network to optimize the binary coding patterns for video compressive sensing. However, our work takes this idea one step farther in optimizing physically realizable pixel exposures for emerging programmable sensors and we evaluate our system in detail with a prototype sensor that does not rely on any external SLMs or ND filters to perform the optical exposure coding. Our system is the first to demonstrate both coded HDR imaging and video compressive sensing on a practical sensor with programmable pixel exposures.

Exotic Sensors: A plethora of unconventional sensors has been explored in previous work. For example, HDR imaging has been demonstrated with multiple exposures built into the pixels [11], [12], using events to signal overflow of the pixel buffer and reset it [33], using precomputation on the focal plane with spatially varying and adaptive exposures [34], [35], [36], using multiple photodiodes with various gains in each pixel [37], or using logarithmic compression at the photosensor [38], as well as many other approaches [13], [39], [40], [41]. Per-pixel coded exposures have also been realized in several dedicated sensors [42], [43], [44], [45], for example with applications to video compressive sensing. Finally, dynamic vision or event sensors [46], [47], [48] have been demonstrated to achieve high-dynamic range and high-speed imaging. However, the image quality achieved with all of these sensor technologies is fundamentally limited by their fixed function pipeline. None of them offer dynamically re-programmable pixel exposures.

Programmable and Reconfigurable Sensors: A rapidly increasing number of sensors provide unprecedented processing capabilities by co-locating both sensing and processing electronics in their pixels. This class of sensors has been coined *near-focal-plane sensor–processor* [16]. These devices were first designed with low-level image processing capabilities in mind [49], [50], [51] before being developed to provide some level of programmability [52], [53], [54], [55], [56]. Nowadays, the idea of embedding processing capabilities inside a pixel has (mostly) been instantiated in very high-frame rate sensors that can have complex in-pixel analog-to-digital conversion (ADC) with a limited amount of processing [14], [15]. Nevertheless, a few sensors such as SCAMP-5 [17] offer fully programmable pixels, and have been used in applications such as depth-from-focus [57], feature extraction [58] or ego-motion estimation [59]. Here, we introduce an end-to-end optimization paradigm to jointly designing the spatially varying pixel exposures offered by emerging sensor–processors and neural network-based image reconstruction frameworks for applications in HDR and high-framerate imaging. Our work paves the way for a new class of neural sensors with optimized application-specific capabilities.

End-to-end optimization of optics and image processing: Although the co-design of optics and algorithms is one of the core ideas behind computational photography, programming tools with automatic differentiation capabilities have only recently enabled a true joint design of hardware and software. This idea has previously been explored for a number of applications, such as color imaging and demosaicing [60], extended depth of field imaging [61], depth imaging [62], [63], image classification [64], and HDR imaging [65]. While the key idea of end-to-end optimization of camera parameters and algorithmic processing is similar in these approaches and ours, to our knowledge we are the first to propose a learning strategy for the spatially varying pixel exposures of a programmable sensor–processor.

3 End-to-end Optimization of Pixel Exposures and Image Reconstruction

In this section, we develop a mathematical framework for end-to-end optimization of spatially varying pixel exposures and a differentiable reconstruction algorithm, such as a neural network. We interpret this system as an electronic encoder, digital decoder system (see Fig. 1).

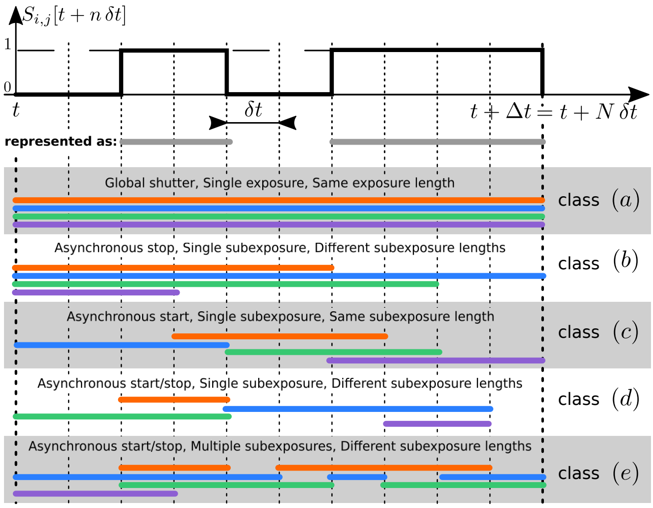
$S_{i,j}[t + n\,\delta t]$
1
0
t
δt
$t + \Delta t = t + N\,\delta t$
represented as:
Global shutter, Single exposure, Same exposure length
class (a)
Asynchronous stop, Single subexposure, Different subexposure lengths
class (b)
Asynchronous start, Single subexposure, Same subexposure length
class (c)
Asynchronous start/stop, Single subexposure, Different subexposure lengths
class (d)
Asynchronous start/stop, Multiple subexposures, Different subexposure lengths
class (e)

Fig. 3. **A diagram illustrating different shutter functions.** The top diagram shows how we represent a shutter function: A solid line on a given slot n indicates that for this slot: $S_{i,j}[t + n\,\delta t] = 1$, otherwise $S_{i,j}[t+n\,\delta t] = 0$. The bottom rows show four classes of shutter functions (from the simplest to the most general). For each class, four examples of shutter functions for different pixels are represented by different colors.

3.1 Modeling Pixel Exposures as an Encoder

We consider an exposure model in which a pixel at position (i, j) on a sensor array integrates the incident irradiance $L_{i,j}$ over the exposure time Δt as

$$E_{i,j}(t) = \int_t^{t+\Delta t} L_{i,j}(t')\,dt'. \tag{1}$$

Here, $E_{i,j}$ is the exposure, and relates to an image as $I_{i,j}(t) = \mathcal{R}\left(E_{i,j}(t)\right)$ in which $\mathcal{R}$ is the camera response function, and $I_{i,j}$ is the reported intensity value of each pixel.

Following recent work on coded and compressive imaging (see Sec. 2), we introduce a shutter function S that modulates pixel (i, j) throughout its exposure time:

$$E_{i,j}(t) = \int_t^{t+\Delta t} L_{i,j}(t') \cdot S_{i,j}(t')\,dt'. \tag{2}$$

In our work, we primarily consider *binary* shutter functions $S_{i,j}$ defined on N *discrete time slots*. Thus, we rewrite Equation (2) as

$$E_{i,j} = \sum_{n=0}^{N-1} L_{i,j}\left[n\right] \cdot S_{i,j}\left[n\right]. \tag{3}$$

A common approach is to write this discrete model as a matrix–vector multiplication $\mathbf{E} = \mathbf{S}\,\mathbf{L}$, where $\mathbf{E} \in \mathbb{R}^M$ is the vectorized form of the measured exposure of all M pixels, $\mathbf{L} \in \mathbb{R}^{MN}$ is the vectorized form of the temporally varying irradiance incident on all pixels, and $\mathbf{S} \in \mathbb{R}^{M\times MN}$ is the measurement matrix. Due to the fact that $\mathbf{S}$ often has fewer rows than columns, and is thus not invertible, a compressive sensing approach is commonly applied to estimate $\mathbf{L}$ from $\mathbf{E}$ using some form of sparsity-based regularization.

Here, we adopt a slightly different notation that represents the shutter functions in a parameterized way. Specifically, we assume that these functions are fully described by a set of parameters ϕ, such that we can define an encoding operator $\mathcal{S}_\phi : \mathbb{R}^{MN} \to \mathbb{R}^M$ to represent Equation 3. This operator can represent several different classes of shutter functions, each defined by their own limited degrees of freedom, as illustrated in Figure 3.

We consider scenarios in which each pixel (i, j) on the sensor can realize its own shutter function, hence there is a set of parameters $\phi_{i,j}$ for each pixel. We further assume that the shutter functions do not vary from one exposure to another. The most general class of such shutter functions defines each time slot as either “on” or “off” and thus consists of N free parameters for each pixel $S_{i,j}[n] \in \{0, 1\}, \forall n \in \{0, ..., N-1\}$ (see class (e) in the last row of Fig. 3).

3.2 Recovering an Image with a Decoder

To estimate the irradiance from the coded measurements, we use a differentiable reconstruction algorithm or decoder $\mathcal{D}_\psi : \mathbb{R}^M \to \mathbb{R}^{MN}$, which is defined by the set of parameters ψ. Given a pre-trained encoder–decoder, one would capture measurements with the corresponding shutter functions $E = \mathcal{S}_\phi(L)$ and then estimate the irradiance as $\widehat{L} = \mathcal{D}_\psi(E)$. This is commonly referred to as the inference stage.

To determine an optimal set of parameters $\{\phi, \psi\}$, we train the encoder–decoder end-to-end with some dataset containing K ground truth irradiance maps using the loss function $\mathcal{L}$ by minimizing the following objective function with some variant of stochastic gradient descent:

$$\underset{\{\phi,\psi\}}{\arg\min} \sum_{k=1}^{K} \mathcal{L}\left(\mathcal{D}_\psi \circ \mathcal{S}_\phi\left(L^{(k)}\right), L^{(k)}\right). \tag{4}$$

This is a self-supervised learning scenario in which the encoder implements a physically feasible shutter function and the decoder recovers the image. Note that both encoder and decoder are jointly optimized in this case, so the optical codes parameterized by ϕ influence the reconstruction algorithm ψ and vice versa. Moreover, the optimal choice of both encoder and decoder depends on the loss function, which could be a low-level image metric, such as mean squared error (MSE), or a higher-level loss such as image classification accuracy.

The encoder–decoder system presented here is somewhat generic and could be applied to many different coded computational photography problems. In the following, we discuss how to incorporate specific constraints of our programmable sensor and model them in a differentiable way using the encoder $\mathcal{S}_\phi$. We also discuss different choices for the decoder $\mathcal{D}_\psi$ for the specific problems of HDR and high-speed compressive imaging.

4 Differentiable & Programmable Sensor–Processor

4.1 A Programmable Sensor–Processor

To implement the optimized shutter functions we use SCAMP-5 [17], a reconfigurable near-focal plane sensor–processor. The sensor–processor is a 256×256 pixel array. Each pixel contains a photo-sensitive element collocated with a general purpose programmable Processing Element (PE).

Algorithm 1: Pseudocode for the implementation of our framework on a near-focal plane processor
The for-loops highlighted in grey can be thought as “parallel-for” executed by compute kernels. PIX is the photosensitive element, whose value is at any instant the integrated value of irradiance from the last reset (one can alternatively think about every inner loop incrementing PIX as seen in the comment in the pseudocode).

```
for all pixels $(i, j)$ do
    $\mathcal{C}_{i,j} \leftarrow f(S_{i,j})$
for all frames do
    for all pixels $(i, j)$ do
        $\mathrm{Frame}_{i,j} \leftarrow 0$
    for all slots $n \in \{0, ..., N-1\}$ do
        for all pixels $(i, j)$ do
            if $g(\mathcal{C}_{i,j}; n) = 0$ then
                $\mathrm{PIX}_{i,j} \leftarrow 0$
            else if $g(\mathcal{C}_{i,j}; n) = 1$ then
                $\mathrm{Frame}_{i,j} \leftarrow \mathrm{Frame}_{i,j} + \mathrm{PIX}_{i,j}$
        // $\mathrm{PIX}_{i,j} \leftarrow \mathrm{PIX}_{i,j} + L_{i,j}(t) \cdot \delta t$
    Readout Frame
```

Each PE consists of a simple Arithmetic Logic Unit (ALU) implemented with a mixed-signal circuit along with a few memories. Those are 7 analog memories as well as 13 single bit digital memories. Analog memories can store values from the photo-sensitive element as well as the results of analog operations carried out by the ALU. Digital memories on the other hand can be fed by the output of a comparator acting on the analog memories as well as set and cleared by dedicated instructions.

The integrated value of a photo-sensitive element (since its last reset) can be read out non-destructively at any point in time. It is thus possible to “sense” and “compute” simultaneously. The SCAMP-5 vision chip we use is harnessed by an NXP LPC4357 micro-controller, dispatching instructions to the vision chip, reading out data, and acting as a server when communicating with an external computer.

Code executed by the micro-controller harnessing the vision-chip is written in C++, code that is dispatched to the vision-chip is written in a Domain Specific Language (DSL) in the form of what can be thought as “compute kernels” executed in parallel by all the pixels’ PE on their local piece of data following the Single Instruction Multiple Data paradigm. These compute kernels consist of a set of macros and high-level instructions. An example of such a high-level instruction is transferring the content of an analog memory to another.

Each high-level instruction from the DSL compiles down to a stream of instruction code words (ICW) in a syntax directed translation fashion, i.e. the parser drives the generation of the ICW stream. This ICW stream can be thought of as microcode that indicates at each instruction clock, which “gates” are to be switched on chip to, say, write the content of a cell implementing an analog memory to another.

4.2 Implementation and Execution of Shutter Functions

In our implementation, at boot up, each pixel is distributed a pixel code $\mathcal{C}_{i,j} \in \mathbb{N}$, that we also represent as a vector of B

TABLE 1

Pairs of encoding and decoding functions for certain class of shutter functions For compactness we denote the max rectifier $x^+ = \max(x,0)$ and the min rectifier $x^- = -\min(x,0)$ as well as the backward difference operator $\nabla x[n] = x[n] - x[n-1]$.

Class	$f(S)$	$B = \lvert\underline{\mathcal{C}}\rvert$ (bits)	$g(\mathcal{C};n)$
(a)	-	0	1
(b)	$\sum_{n=0}^{N-1} \mathbf{S}[n]$	$\log N$	$H(\mathcal{C}-n)$
(c)	$\sum_{n=1}^{N-1} n(\nabla S[n])^+$	$\log N$	$\Pi\left(L\cdot\left(n+\frac{1}{2}\right)-\mathcal{C}\right)$
(d)	$\sum_{n=1}^{N-1} n\cdot(\nabla S[n])^+ + N\sum_{n=1}^{N-1} n\cdot(\nabla S[n])^-$	$2\log N$	$H\left(n-(\mathcal{C} \bmod N)\right)\cdot H((\mathcal{C}/N)-n)$
(e)	$\sum_{i=0}^{N-1} \mathbf{S}[i]\, 2^i$	N	$\mathrm{proj}_n(\mathcal{C}) = \lfloor \mathcal{C}\cdot 2^{-n}\rfloor \bmod 2$

bits: $\underline{\mathcal{C}}_{i,j} = \{0,1\}^B$ which stores an encoding of the shutter function $S_{i,j}$ at the pixel level through the function:

$$\begin{aligned} f : \{0,1\}^N &\rightarrow \mathbb{N} \\ f(S_{i,j}) &= \mathcal{C}_{i,j}. \end{aligned} \tag{5}$$

For each frame, during the exposure ΔT, the microcontroller dispatches N times, to all the pixels, a global signal occurring precisely at the beginning of each slot. The signal is an integer corresponding to the slot number $n \in \{0, ..., N-1\}$. All the pixels' PE evaluate, in parallel, some decoding function g of the pixel code and of the global signal (the slot number):

$$\begin{aligned} g : \mathbb{N}\times\{0,...,N-1\} &\rightarrow \{0,1\} \\ g(\mathcal{C}_{i,j};n) &= p_{i,j,n}. \end{aligned} \tag{6}$$

The function g encodes the state $p_{i,j,n}$ the pixel (i,j) should be in for the timeslot n. When $g(\mathcal{C};n) = 1$ the pixels turns "on", while $g(\mathcal{C}_{i,j};n) = 0$ it is switched "off". The pair of encoding function f and decoding function g depends on the particular class of shutter function we implement and has to satisfy the identity:

$$g\big(f(S_{i,j}), n\big) = S_{i,j}[n]\ \forall(i,j),\ \forall n, \tag{7}$$

in order to be correct. These encoding/decoding pairs are not unique as we shall see in the following. A pseudo-code summarizing how our exposure program is implemented on our device is given in Algorithm 1. Note that the encoding function f is executed only once, at boot up, and does not need to be computed in-pixel: it can be precomputed (in the microcontroller) and distributed to the pixel, simply setting its digital memories. Therefore, it can be arbitrarily complex. On the other hand, the decoding function g needs to be simple enough that it can be evaluated by the pixel ALU. In all our implementations, the function g can be expressed as a few digital comparisons and simple digital operations (requiring only using a few OR and NOT evaluations, which are the two gates "natively" built-in the pixel ALU). Furthermore, since each pixel has a limited memory, there exists a trade-off between the complexity of the shutter function against its time-resolution, i.e the number of slots it runs on.

4.3 General Shutter Functions

Shutter functions can be classified according to their complexity as well as their intended use: for HDR, for high-speed etc. The next subsections describe the classes outlined in Figure 3 and Table 1 in detail.

The most general class of shutter functions that can be defined on N slots prescribes independently for each slot whether the shutter is "on" or "off" at pixel (i,j) (class (e) in Table 1) . Using such a general class of shutter functions, a system can only use $N = B$ slots, constrained by the number of bits B that can be locally stored in each pixel digital memories. On SCAMP-5 this means we cannot use more than 13 slots.

Class (e): Such a shutter function can be parameterized by a vector $\underline{\phi}^{(e)}_{i,j} \in \{0,1\}^N$. The n^{th} entry of this vector states whether the shutter is on (1) or off (0) at slot n:

$$S^{(e)}_{i,j}[n] = \sum_{n'=0}^{N-1} \underline{\phi}^{(e)}_{i,j}[n']\,\delta[n-n'], \tag{8}$$

in which $\delta[n]$ is the discrete delta function. Examples of such class (e) functions for different pixels are illustrated in Figure 3. The parameters to be learned for our encoder are these vectors $\underline{\phi}^{(e)}_{i,j}$ for all pixels. A natural encoding function f is the conversion of this binary code in an integer:

$$f^{(e)}(S_{i,j}) = \sum_{n=0}^{N-1} S_{i,j}[n]\,2^n. \tag{9}$$

The decoding function g is simply the projector function that selects the n^{th} bit of $\underline{\mathcal{C}}_{i,j}$:

$$\begin{aligned} g^{(e)}(\mathcal{C}_{i,j};n) &= \mathrm{proj}_n(\mathcal{C}_{i,j}) \\ &= \underline{\mathcal{C}}_{i,j}[n] = \lfloor \mathcal{C}_{i,j}\cdot 2^{-n}\rfloor \bmod 2. \end{aligned} \tag{10}$$

4.4 Structured Shutter Functions

Less "general" (more structured) shutter functions typically require an encoding on fewer bits, allowing us to implement longer shutter functions, with more slots! Hence, for a fixed code length B, there is a trade-off: one can describe long simple shutter functions, or arbitrarily complex (random) but short shutter functions. Indeed, regularities in the shutter function can be used by the encoding and decoding functions. Examples of such longer but simpler shutter functions are functions of class (b),(c),(d) as illustrated in Figure (3). These are the functions we use for HDR imaging–class (b)–and high frame rate imaging–class (c), (d) and (e).

HDR Imaging

A first use-case for learning optimized shutter functions is high-dynamic range imaging. We assume the imaged scene does not move (much) throughout the exposure time of a single frame. In this scenario, even if we consider shutter function consisting of multiple "on" bumps interspersed during the whole exposure time Δt, this is equivalent to consider a shutter function in which those were contiguous and the shutter function for a pixel starts “on” at $n = 0$ and goes “off” after some amount of time. This is because, assuming $L_{i,j}[n] = L_{\text{cst.}}$ for the whole $\Delta t = N \cdot \delta t$ we can rewrite Equation 3 as:

$$E_{i,j} = L_{\text{cst.}} \sum_{n=0}^{N-1} S_{i,j}[n] = L_{\text{cst.}}\, \phi_{i,j}^{(b)}, \tag{11}$$

which is independent of the position of the “on” bumps in the shutter function and simply weighs the incoming irradiance $L_{\text{cst.}}$ by the amount of time $\phi_{i,j}^{(b)} = \sum_{n=0}^{N-1} S_{i,j}[n]$ the shutter is “on”. Hence, those can be parameterized with a single parameter corresponding to this “weight”. This class of shutter functions is one of the simplest (it has only one parameter to learn for each shutter function) but can certainly achieve HDR imaging as different pixels (i, j) will learn different shutter functions corresponding to different exposure times controlled by $\phi_{i,j}^{(b)}$.

Class (b): Specifically these functions can be parameterized by starting the exposure in the “on” state at $n = 0$ and using the single parameter to encode when the shutter should stop. These are functions of class (b) in Figure 3. This parametrization of $S_{i,j}$ is the horizontally flipped Heaviside function H shifted by $\phi_{i,j}^{(b)}$ on the time axis:

$$S_{i,j}^{(b)}[n] = H(\phi_{i,j}^{(b)} - n). \tag{12}$$

Since a function of class (b) only needs to encode the slot number at which a pixel stops integrating, if we consider such a function defined on N slots, only $B = \log N$ bits are required to encode that number. Another way to see this, is that with B available digital memories one can store a number that addresses 2^N slots. This is obviously greater than the N slots one could implement for a function of class (e), the most general class. An encoding function for this class can be written as:

$$f^{(b)}(S_{i,j}) = \sum_{n=0}^{N-1} S_{i,j}[n]. \tag{13}$$

Given all the bits of $S_{i,j}$ are 0 after a given n_{end} that encodes the end of integration, f just encodes n_{end} as an integer. As a consequence the paired decoding function can simply encode $\mathcal{C}_{i,j} = \phi_{i,j}^{(b)}$ and be:

$$g^{(b)}(\mathcal{C}_{i,j}; n) = H(\mathcal{C}_{i,j} - n). \tag{14}$$

Video Compressive Sensing

We further experimented with a class of shutter functions for high frame rate imaging. In general, it is possible to reconstruct a video sequence at high frame rate from a single frame as long as the shutter functions start at different offsets (and eventually stop at different offsets) within the exposure. Indeed, if all shutter functions at all pixels were to start at the same point in time $n = n_{\text{start}}$, and would stop, say at n_{end}, it would be impossible to reconstruct the irradiance out of the time interval $[n_{\text{start}}, n_{\text{end}}]$ based on some real, captured information.

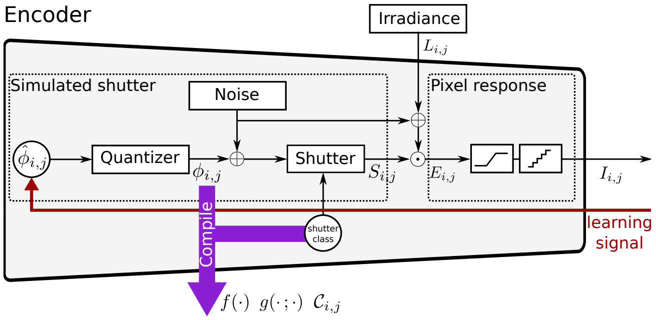

Fig. 4. **A zoomed-in diagram of the encoder.** This diagram illustrates the implementation details of our encoder for a given pixel (i, j) and a single parameter $\phi_{i,j}$. The learned parameter is the real-valued $\hat{\phi}_{i,j}$. The quantized parameter $\phi_{i,j}$ is the one compiled on the sensor in $\mathcal{C}_{i,j} = f(S_{i,j})$.

Class (c): Thus, the simplest class of shutter functions one can implement is obtained by offsetting the starting point of the shutter functions of each pixel, and exposing a single time, for a fixed duration L. The shutter is “on” for a fixed amount of time L and then goes “off” for the rest of the exposure duration. This can be described using a single parameter $\phi_{i,j}^{(c)}$ (for instance the start time n_{start}) and can be parameterized as:

$$S_{i,j}^{(c)}[n] = \Pi\left(L \cdot \left(n + \frac{1}{2}\right) - \phi_{i,j}^{(c)}\right), \tag{15}$$

in which Π is the rectangle function centered in 0, and $\Pi(x) = 1$ when $x \in [-0.5, 0.5]$ and 0 everywhere else. Such shutter functions belong to class (c) as illustrated in Figure 3. They also require $B = \log N$ bits for their encoding, as they can also be described by a single free parameter as in Equation (15), in which $\phi_{i,j}^{(c)}$ represents the slot number when the integration starts. Since there is a single subexposure of a fixed duration L that is prescribed a-priori, the slot at which the pixel stops integrating is simply $n_{end} = n_{\text{start}} + L$. Note that, alternatively, the function could be encoded by the time slot at which integration stops, which makes no difference, but shows that these encoding/decoding pairs are not unique. An encoding function for this class can be written as:

$$f^{(c)}(S_{i,j}) = \sum_{n=1}^{N-1} n \cdot \max\left(S_{i,j}[n] - S_{i,j}[n-1], 0\right), \tag{16}$$

which simply encodes the starting slot as an integer, while a decoding function is again simply reflecting the parametrization of the shutter function, with $\mathcal{C}_{i,j} = \phi_{i,j}^{(c)}$:

$$g^{(c)}(\mathcal{C}_{i,j}; n) = \Pi\left(L \cdot (n + \frac{1}{2}) - \mathcal{C}_{i,j}\right). \tag{17}$$

Class (d): For more complex behavior, one can add a second parameter to this shutter function. This parameter encodes, for instance, the end time n_{end} of the “bump” instead of integrating at each pixel for a fixed time L. Since all the bumps have different durations (in addition of being offset) this might be beneficial when the aim is

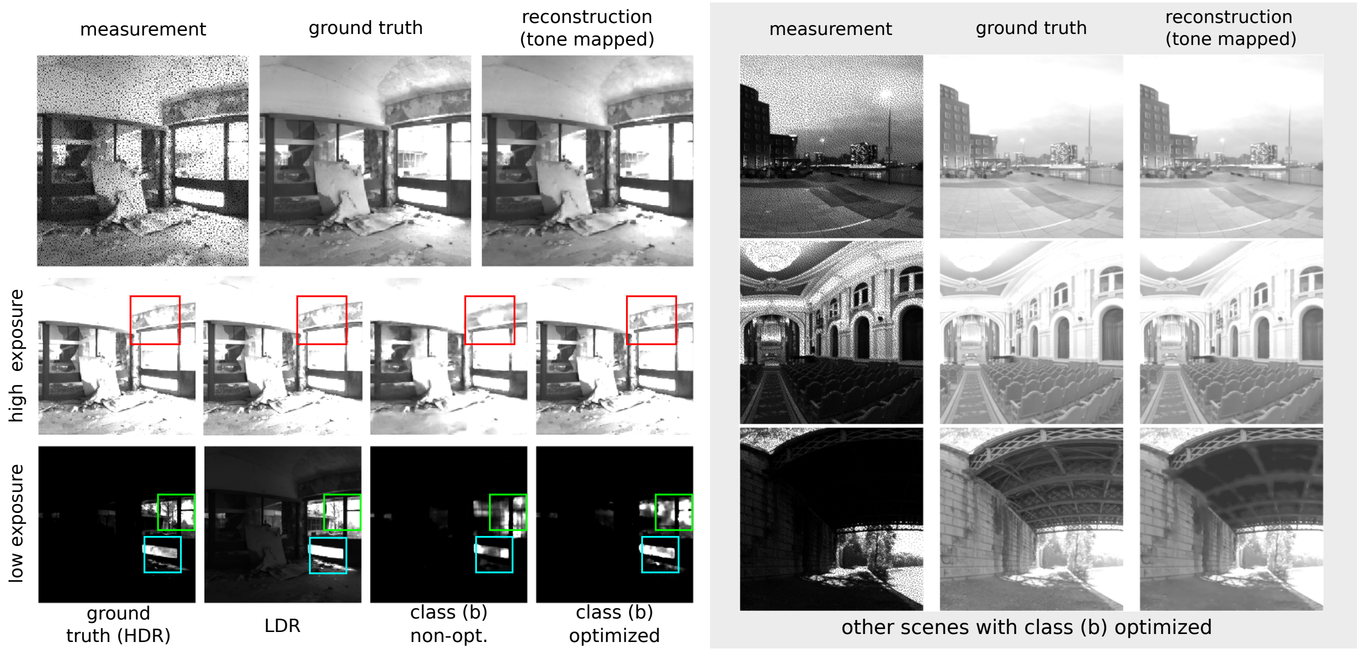

Fig. 5. **HDR results in simulation.** On the right pane: the images show ground truths, measurements and reconstructions for different HDR scenes captured with shutter functions of class (b) in simulation along with some baseline comparisons. The left pane shows other scenes captured with optimized shutters of class (b).

to recover HDR information in addition to high-frame rate reconstruction. A parameterization for such functions is:

$$S^{(d)}_{i,j} = H(n-\phi^{(d),1}_{i,j})\cdot H(\phi^{(d),2}_{i,j}-n), \tag{18}$$

effectively realizing a rectangle function with two Heaviside (one is flipped) shifted respectively by the two parameters $n_{\text{start}} = \phi^{(d),1}_{i,j}$ and $n_{\text{end}} = \phi^{(d),2}_{i,j}$.

These shutter functions correspond to class (d) in Figure 3. In this case, two integers $(n_{\text{start}}, n_{\text{end}})$, need to be encoded (and decoded) in a single integer $\mathcal{C}$. This can easily be done by the use of a pairing function. Thanks to the constraints: $n_{\text{start}} < N$ and $n_{\text{end}} < N$, one can choose a very simple pairing, for instance relying on the uniqueness of the Euclidean division: $\mathcal{C} = n_{\text{start}} + n_{\text{end}}\cdot N$, the pairing inversion is $n_{\text{start}} = \mathcal{C} \mod N$ and $n_{\text{end}} = \lfloor \mathcal{C}/N \rfloor$. An encoding we propose is:

$$f^{(d)}(S_{i,j}) = n_{\text{start}} + N\cdot n_{\text{end}}, \tag{19}$$

with, $n_{\text{start}} = \sum_{n=1}^{N-1} n\cdot \max(S[n]-S[n-1],0)$, and, symmetrically: $n_{\text{end}} = -\sum_{n=1}^{N-1} n\cdot \min(S[n]-S[n-1],0)$ that both capture the start and end time of the subexposure bump making use of the sign of the forward difference of $S_{i,j}$. This time, the decoding is less trivial, and uses our pairing inversions:

$$g^{(d)}(\mathcal{C}_{i,j};n) = H(n-\mathcal{C}_{i,j} \bmod N)\cdot H(n-\mathcal{C}_{i,j}/N), \tag{20}$$

which is very simple to perform in hardware.

4.5 Learning with Non-differentiable Encoders $\mathcal{S}_\phi$

When it comes to learning the parameters $\phi_{i,j}$ of our encoders, there are two main difficulties to address. A first challenge is that we consider binary shutter functions. This is because it simplifies their implementations in electronic by a simple switch instead of a variable, continuous gain: All the shutter functions we presented implement some kind of hard thresholding (using Heaviside, rectangle, or projection functions). A hard threshold does not yield useful gradients for training. A second, and major difficulty is that the domain of the parameters ϕ is discrete: the parameters for all the shutter functions we presented essentially encode the slot at which sub-exposure bumps start and stop.

In the neural network literature, settings in which the outputs of neurons (the analog of our shutter functions) are binary and their weights (the analog of our parameters) are quantized are more difficult to optimize and require some tricks, we adopt similar methods to [66], using a real variable $\hat{\phi}_{i,j}$ optimized during training and, quantized in the discrete $\phi_{i,j}$ during the forward pass as shown in Figure 4.

4.6 Modeling of the Encoder: Noise and Pixel Response

Despite the fact we learn the encoder on real data, this data has not been captured by our sensor. Since we aim at compiling the shutter functions on our real-system, it is crucial the encoder trained in simulation models the operation of our sensor-processor closely enough. In practice, we consider two aspects: First the pixel response function is approximately linear and saturates, also it is quantized (the sensor reports 8-bits images). Secondly, performance in the reconstruction is increased when using noise. We model both the response function of the sensor and noise in the encoder. Noise is modelled both in the quantization step of the real parameter we optimize over (modelling the fact the shutter function might not be realized correctly), and in the sensor capture (on the incoming irradiance). The implementation of our encoder is illustrated in Figure 4.

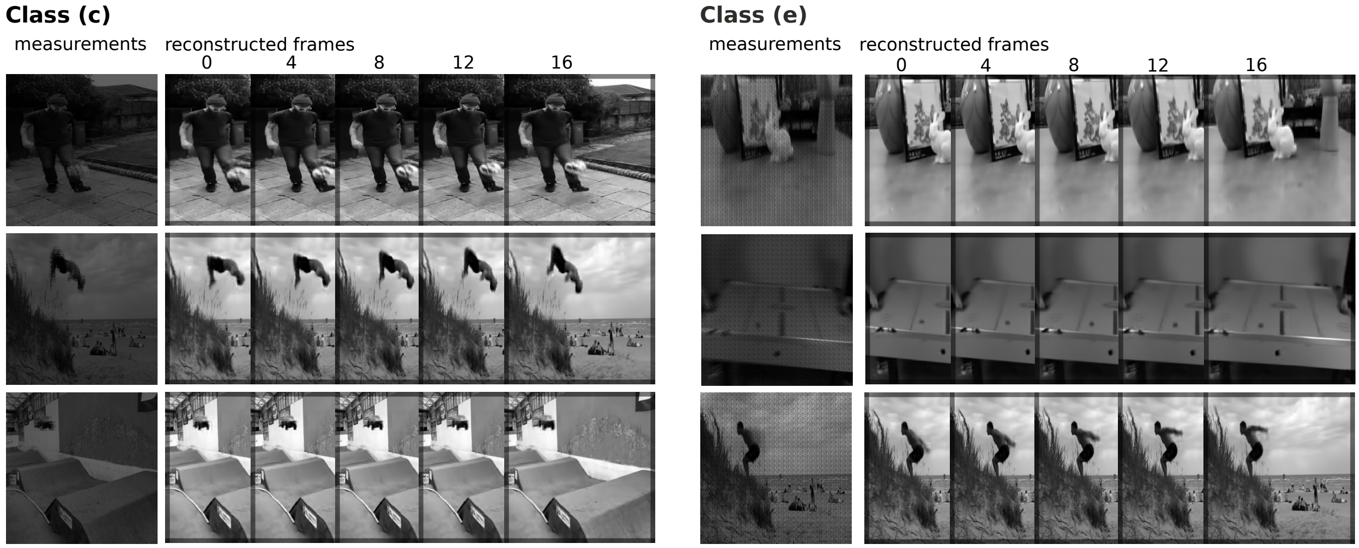

Fig. 6. **Video compressive sensing results in simulation.** These images show individual coded images (measurements) along with the reconstructed video clips for several high-speed scenes in simulation, using shutter functions of class (c) and (e).

TABLE 2
Results for HDR imaging. We compare PSNR, SSIM for non-optimized and optimized codes of different classes.

Experiment	Shutter class	PSNR (dB)	SSIM
no-coding, LDR	-	13.65	0.2834
random code + CSC [23]	(b)	28.71	0.6476
random code + U-Net	(b)	32.76	0.8904
optimized code + U-Net (ours)	(b)	**35.42**	**0.9464**

4.7 Neural Network Architectures $\mathcal{D}_\psi$

We use two types of DNN models implementing D_ψ through our experiments: Fully-Connected Neural Networks (FC-nets) as well as Convolutional Neural Networks (CNNs) architectures.

Setup for Fully-Connected Networks: A major challenge in using FC-nets is to keep the number of parameters reasonable. Concretely, for a $M = R \times C$ coded exposure produced by our sensor, the first layer of the FC-net would need to contain $R \cdot C$ input neurons, while the output would need to consist of $M \cdot N = R \cdot C \cdot N$ neurons, which means the network has $R^2 \cdot C^2 \cdot N$ parameters not even assuming a hidden layer! For a modest size 256×256 image using 16slots this is already 69 million parameters, which is prohibitively high. Therefore, one cannot consider whole images as inputs of a FC-net. Instead, to reduce the number of parameters we consider chopping the image in smaller patches of size $W_p \times H_p$. To avoid blocky artifacts that would be created by having each patch reconstructed independently from its neighbour, we learn our network on a small tiling of these patches $N_p \times W_p \times N_p \times H_p$. For instance, considering 8×8 patches, and a tiling of 2 patches in each dimension, we obtain a 16×16 block. The FC-net is learned on these blocks, and at reconstruction can be evaluated on each block dividing a whole image, shifted by the size of a patch. Each patch (apart from the borders) is covered by $N_p \cdot N_p$ overlapping blocks and thus benefits from $N_p \cdot H_p$ reconstructions that can be averaged together and thus vary smoothly over the image. A similar approach is taken in [32].

As a consequence, the parameters ϕ of the encoder also need to be shared for each patch. Hence using FC-nets one can only implement $W_p \times H_p$ different shutter functions $S_{i,j}$ (that are all different within a patch), thus under-using our sensor-processor that can implement independent pixel-wise exposures for a whole array.

Setup for Convolutional Neural Networks: Another way to reduce the number of parameters of our model is to use built-in weight sharing via convolutional architectures. The main challenge when using CNN based architectures—that have proven to work well in the problems of image reconstruction, such as U-Nets [67]—is to match their inputs and outputs so they can learn the correlations existing between them in the space they live in. Specifically, in the high-speed imaging learning scenario, the input is a two-dimensional coded exposure image, while the output is a three-dimensional (two dimension and time) sequence of images. This setting does not "naturally" lends itself to 2D U-Nets, and would ignore correlations across one of the dimensions (for instance time if the two first dimensions are the space dimensions). Hence we lift-up the input coded exposure image in three dimensions, by copying it in time (acting like a blur) and applying the shutter function to the obtained volume. This is a typical trick that can be shown to be equivalent to applying the transpose of the measurement matrix to the measurement: $\hat{\mathbf{L}} = \mathbf{S}^\intercal\mathbf{E}$. This is now given as an input to a 3D U-Net, that sees a blurry volume with "holes" where data was not captured.

5 Results

5.1 HDR Imaging

For our experiments in simulation, we use 277 irradiance images extracted from HDR Haven (https://hdrihaven.com), rescaled to a size of 1024×512 using bilinear downsampling. For evaluation, 28 are

TABLE 3
Results for video compressive sensing. We compare PSNR, SSIM for non-optimized and optimized codes of different classes

	Experiment	Shutter class	PSNR (dB)	SSIM
FC-net	non-optimized structured code	(c)	27.73	0.9445
	optimized structured code	(c)	27.78	0.9449
	non-optimized structured code	(d)	27.85	0.9443
	optimized structured code	(d)	28.16	0.9449
	non-optimized random-code	(e)	28.02	0.9430
	optimized random-code	(e)	**28.78**	**0.9502**
3D U-Net	non-optimized structured code	(c)	32.67	0.9225
	optimized structured code	(c)	32.87	0.9272
	non-optimized random-code	(e)	32.19	0.9254
	optimized random-code	(e)	**33.56**	**0.9374**

TABLE 4
Baseline comparisons for video compressive sensing Those are are run on 200 video-clips randomly sampled from the NFS dataset.

	Experiment	Shutter class	PSNR (dB)	SSIM
Others	Bilinear (rand. code + bilin. interp.)	(c)	18.95	0.4133
	Thin out (subsamp. + bilin. interp.)	n/a	20.65	0.4556
	Hitomi et al. [9]	(c)	26.22	0.7258
Ours	FC-net (optim. structured code)	(c)	**38.14**	**0.9555**
	U-Net (optim. structured code)	(c)	32.40	0.8118

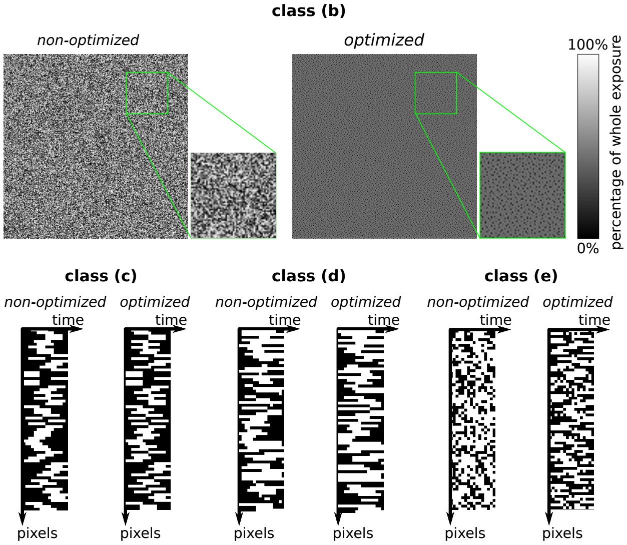

Fig. 7. **Examples of non-optimized and optimized shutter functions.** Different classes of shutter functions are shown for different pixels, when optimized and non-optimized. For functions of class (b) we encode in a grey level image the stopping time of the integrations. For all the other classes we choose a subset of 8×8 pixels for which we show the shutter functions in time: black means the pixel is 'off', white it is 'on'.

randomly selected among those. The remaining images are used for training the 2D U-Net (6 double conv. layers Conv/ReLU/Conv/ReLU, with 3 × 3 kernels, split by avg. pooling in the downsampling branch and bilinear upsampling in the upward branch, the number of kernels starts at 32 up and grows up to 1024 in each layer with a growth factor of 2). We feed randomly cropped 256×256 sub-images (our sensor size) as input to the encoder implementing shutter functions of class (b). Inputs are augmented with different contrast, brightness, rotations and flips in space. Those networks are trained with ADAM [68], on an L_1 loss computed between the ground truth HDR images and the HDR output of the network (not tone mapped), training details are given in our supplemental. Qualitative results for HDR imaging are shown in Fig 5.

Table 2 shows quantitative comparisons of several snapshot HDR approaches. Random optical codes with a convolutional sparse coding (CSC) reconstruction algorithm [23] improve upon the conventional LDR baseline. Random pixel exposures with our U-Net reconstruction show further improvements over CSC. Note that the insight that random coding with a neural network–based reconstruction outperforms other snapshot approaches was also recently made in [24]. Finally, our pixel exposures, learned end-to-end with the U-Net, perform best in this experiment, outperforming the best random approach by about 3dB.

In related work [36], the same sensor–processor implements on-sensor tone mapping resulting in an 8-bit LDR image. The LDR image it captures cannot be directly compared with the HDR reconstruction technique presented here, although our results could also be tone mapped.

5.2 Video Compressive Sensing

For high-speed imaging, we train shutter functions of class (c), (d), and (e) jointly with their decoders with the Need For Speed dataset [69] using the 100 videos captured at 240FPS. Among those, 10 are randomly selected for evaluation. We trained both FC-nets (4 FC layers of size 4096) and 3D U-Nets (5 double conv. layers Conv/ReLU/Conv/ReLU, with $3 \times 3 \times 3$ kernels, split by avg. pooling in the downsampling branch and bilinear upsampling in the upward branch, number of kernels starting at 24 up to 768 with a growth factor of 2 in each layer) as decoders, on the output of the encoders of the various shutter classes. Inputs are 12.5 million randomly selected $16 \times 16 \times 16$ blocks in the FC-net and full $256 \times 256 \times 16$ images in the U-Net. Blocks and images are augmented with rotations and flips in space, and time-reversal. The networks are trained with ADAM [68], optimizing an L_1 loss on the discrepancy between ground-truth and reconstructed blocks (for FC-net) and video-clip / space-time volumes (for 3D U-Nets). Other training details are given in our supplemental. Results comparing optimized and non optimized codes across different shutter classes are shown in Figs 6, 7, 8 and summarized in Table 3. We also present quantitative comparisons to various baselines in Table 4. Notably, we evaluate the coded exposures of Hitomi et al. [9] performed with random codes and sparse coding reconstructions, on a random subset of 200 videos of our test set. For a fair comparison, we use the "single bump" shutter functions of class (c), as described in [9]. We also show a comparison to both "thin out" and "bilinear" baselines. The thinned out baseline considers the high-speed ground truth video and subsamples it in space as much as it is "super-resolved" in time. This emulates a camera running at high-speed and reading out only a subset of pixels so as to keep the amount of data it would need to transmit constant: Concretely the reconstruction of 16 frames in a space-time volume is simulated by subsampling

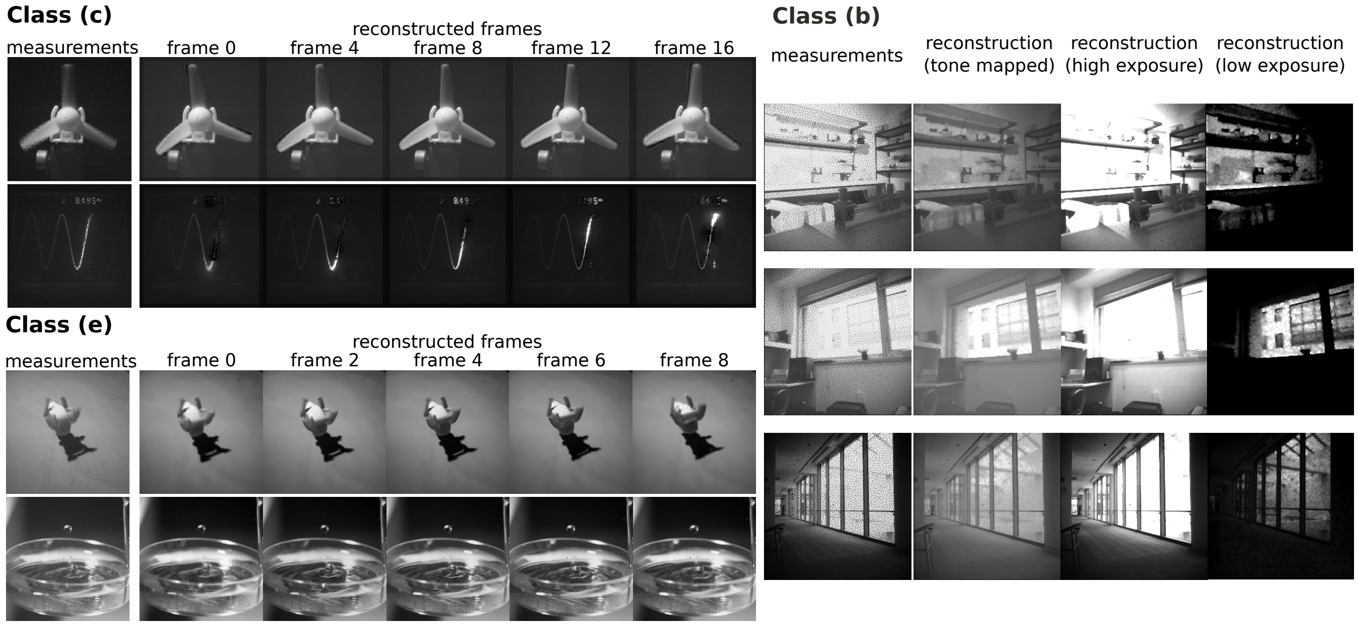

Fig. 8. **Experimental high-speed & HDR captures with our SCAMP-5 prototypes.** The images show measurements captured with our sensor as well as reconstructed frames for different class of shutter functions ((b), (c) and (e)) and decoders (FC-net for class (c) and U-Nets for class (b), (e)). For the HDR images produced using class (b), the tone mapped images are produced using a global tone mapping $I_{\mathrm{TM}} = I_{\mathrm{HDR}}^{\gamma}$ with $\gamma = 0.5$, low and high exposures are produced by clipping the HDR image to a high and low range: eg $I_{\mathrm{low}}(x,y) = \max(\min(I(x,y), h_{\mathrm{low}}), l_{\mathrm{low}})$, $\forall(x,y)$, $h > l$.

each spatial dimension of a ground-truth frame by 4 before upsampling the frame again in the reconstruction. This process involves no coding. The "bilinear" baseline, simply inpaints the missing information in the randomly coded space-time volume (before integration) by bilinear interpolation. Qualitative comparisons and additional details of these baselines are discussed in the supplemental material.

6 Discussion

We presented a method and a system to perform HDR imaging and video compressive sensing with coded pixel exposures that can be learned. We think the end-to-end learning of the components of the imaging pipeline is a useful paradigm, ultimately these components should all be jointly considered and their design should rather be derived from principles that aim at optimizing a given task or objective. This is in line with the idea of the purposive camera: the next generation cameras might be learned end-to-end to fulfil a particular task. This being said it is not clear this camera will be a camera at all (in the sense that it reports images), we advocate the idea of sensors that report visual codes, for instance coded exposures. Therefore, the question to ask is, for a given task: what is the right visual code? That is, how to capture light (under some hardware constraints) and what is the format it should report, that is not necessarily digestable by a human but can be efficiently decoded with some algorithm, that has been jointly optimized to read those and produce human interpretable information.

A current limitation of this framework is adaptivity. Even though the exposure program has been learned on examples, it does not change in a scene dependent fashion. Since processing and sensing are collocated they could influence each other, based on what is being captured. This is supported by our hardware and is, we think, an important future research direction.

On the hardware side, we imagine general purpose programmable sensor processor such as SCAMP-5 to spread in the computational photography and imaging communities. Those offer the flexibility of programming low-level hardware features and close the gap between short development cycles on the algorithmic side when one aims at prototyping a new idea and long VLSI sensor and processor design cycles. This is at the expense of specificity: a programmed sensor processor is not as optimized for any given task as a Application Specific Integrated Circuit (ASIC) would be. Nevertheless, those programmed sensor-processor could be starting points for the design of ASIC, keeping the hardware features used for a given task, optimizing them while crippling them from the unused ones.

Acknowledgments

We thank Greg Zaal for the access to the HDR images (http://hdrihaven.com), used in our experiments. J.N.P.M. was supported by a Swiss National Foundation (SNF) Fellowship (P2EZP2_181817), G.W. was supported by an NSF CAREER Award (IIS 1553333), a Sloan Fellowship, by the KAUST Office of Sponsored Research through the Visual Computing Center CCF grant, and a PECASE by the ARL.

References

[1] P. E. Debevec and J. Malik, "Recovering high dynamic range radiance maps from photographs," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1997, pp. 369–378.

[2] S. W. Hasinoff, D. Sharlet, R. Geiss, A. Adams, J. T. Barron, F. Kainz, J. Chen, and M. Levoy, "Burst photography for high dynamic range and low-light imaging on mobile cameras," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, p. 192, 2016.

[3] S. K. Nayar and T. Mitsunaga, "High dynamic range imaging: Spatially varying pixel exposures," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, 2000, pp. 472–479.

[4] S. K. Nayar and V. Branzoi, "Adaptive dynamic range imaging: Optical control of pixel exposures over space and time," in *IEEE Int. Conference on Computer Vision (ICCV)*, 2003, p. 1168.

[5] S. K. Nayar, V. Branzoi, and T. E. Boult, "Programmable imaging: Towards a flexible camera," *Int. Journal of Computer Vision*, vol. 70, no. 1, pp. 7–22, 2006.

[6] J. Gu, Y. Hitomi, T. Mitsunaga, and S. Nayar, "Coded rolling shutter photography: Flexible space-time sampling," in *2010 IEEE Int. Conference on Computational Photography (ICCP)*, 2010, pp. 1–8.

[7] A. C. Sankaranarayanan, P. K. Turaga, R. G. Baraniuk, and R. Chellappa, "Compressive acquisition of dynamic scenes," in *European Conference on Computer Vision*. Springer, 2010, pp. 129–142.

[8] A. Veeraraghavan, D. Reddy, and R. Raskar, "Coded strobing photography: Compressive sensing of high speed periodic videos," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 4, pp. 671–686, 2010.

[9] Y. Hitomi, J. Gu, M. Gupta, T. Mitsunaga, and S. K. Nayar, "Video from a single coded exposure photograph using a learned over-complete dictionary," in *Int. Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 287–294.

[10] R. G. Baraniuk, T. Goldstein, A. C. Sankaranarayanan, C. Studer, A. Veeraraghavan, and M. B. Wakin, "Compressive video sensing: algorithms, architectures, and applications," *IEEE Signal Processing Magazine*, vol. 34, no. 1, pp. 52–66, 2017.

[11] M. Mase, S. Kawahito, M. Sasaki, Y. Wakamori, and M. Furuta, "A wide dynamic range cmos image sensor with multiple exposure-time signal outputs and 12-bit column-parallel cyclic a/d converters," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 12, pp. 2787–2795, 2005.

[12] D. X. Yang, A. El Gamal, B. Fowler, and H. Tian, "A 640/spl times/512 cmos image sensor with ultra wide dynamic range floating-point pixel-level adc," in *IEEE Solid-State Circuits Conference*, 1999, pp. 308–309.

[13] A. Xhakoni and G. Gielen, "A 132-db dynamic-range global-shutter stacked architecture for high-performance imagers," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 6, pp. 398–402, 2014.

[14] T. Yamazaki, H. Katayama, S. Uehara, A. Nose, M. Kobayashi, S. Shida, M. Odahara, K. Takamiya, Y. Hisamatsu, S. Matsumoto *et al.*, "A 1ms high-speed vision chip with 3d-stacked 140gops column-parallel pes for spatio-temporal image processing," in *IEEE Int. Solid-State Circuits Conference (ISSCC)*, 2017, pp. 82–83.

[15] L. Millet, S. Chevobbe, C. Andriamisaina, E. Beigné, F. Guellec, T. Dombek, L. Benaissa, E. Deschaseaux, M. Duranton, K. Benchehida *et al.*, "A 5500fps 85gops/w 3d stacked bsi vision chip based on parallel in-focal-plane acquisition and processing," in *2018 IEEE Symposium on VLSI Circuits*. IEEE, 2018, pp. 245–246.

[16] Á. Zarándy, *Focal-plane sensor-processor chips*. Springer Science & Business Media, 2011.

[17] S. J. Carey, A. Lopich, D. R. Barr, B. Wang, and P. Dudek, "A 100,000 fps vision sensor with embedded 535gops/w 256× 256 simd processor array," in *2013 Symposium on VLSI Circuits*. IEEE, 2013, pp. C182–C183.

[18] S. Mann and R. Picard, "Being 'undigital' with digital cameras: Extending dynamic range by combining differently exposed pictures," *Proceedings of IS&T*, 1995.

[19] S. W. Hasinoff and K. N. Kutulakos, "Multiple-aperture photography for high dynamic range and post-capture refocusing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, no. 1, pp. 1–17, 2009.

[20] M. D. Tocci, C. Kiser, N. Tocci, and P. Sen, "A versatile hdr video production system," in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 4, 2011, p. 41.

[21] G. Wetzstein, I. Ihrke, and W. Heidrich, "Sensor saturation in fourier multiplexed imaging," in *Proc. CVPR*. IEEE, 2010, pp. 545–552.

[22] S. Hajisharif, J. Kronander, and J. Unger, "Adaptive dualiso hdr reconstruction," *EURASIP Journal on Image and Video Processing*, vol. 2015, no. 1, p. 41, 2015.

[23] A. Serrano, F. Heide, D. Gutierrez, G. Wetzstein, and B. Masia, "Convolutional sparse coding for high dynamic range imaging," in *Computer Graphics Forum*, vol. 35, no. 2, 2016, pp. 153–163.

[24] M. M. Alghamdi, Q. Fu, A. K. Thabet, and W. Heidrich, "Reconfigurable snapshot hdr imaging using coded masks and inception network," in *Vision Modeling and Visualization*, 2019.

[25] R. Raskar, A. Agrawal, and J. Tumblin, "Coded exposure photography: motion deblurring using fluttered shutter," in *ACM transactions on graphics (TOG)*, vol. 25, no. 3. ACM, 2006, pp. 795–804.

[26] A. C. Sankaranarayanan, C. Studer, and R. G. Baraniuk, "Cs-muvi: Video compressive sensing for spatial-multiplexing cameras," in *IEEE Int. Conference on Computational Photography (ICCP)*, 2012, pp. 1–10.

[27] J. Holloway, A. C. Sankaranarayanan, A. Veeraraghavan, and S. Tambe, "Flutter shutter video camera for compressive sensing of videos," in *IEEE Int. Conference on Computational Photography (ICCP)*, 2012, pp. 1–9.

[28] D. Reddy, A. Veeraraghavan, and R. Chellappa, "P2c2: Programmable pixel compressive camera for high speed imaging," in *CVPR*. IEEE, 2011, pp. 329–336.

[29] T. Portz, L. Zhang, and H. Jiang, "Random coded sampling for high-speed hdr video," in *IEEE Int. Conference on Computational Photography (ICCP)*, 2013, pp. 1–8.

[30] M. Iliadis, L. Spinoulas, and A. K. Katsaggelos, "Deep fully-connected networks for video compressive sensing," *Digital Signal Processing*, vol. 72, pp. 9–18, 2018.

[31] N. Antipa, P. Oare, E. Bostan, R. Ng, and L. Waller, "Video from stills: Lensless imaging with rolling shutter," in *IEEE Int. Conference on Computational Photography (ICCP)*, 2019, pp. 1–8.

[32] M. Iliadis, L. Spinoulas, and A. K. Katsaggelos, "Deepbinarymask: Learning a binary mask for video compressive sensing," *Digital Signal Processing: A Review Journal*, vol. 96, 2020.

[33] J. A. Lenero-Bardallo, R. Carmona-Galán, and Á. Rodríguez-Vázquez, "A high dynamic range image sensor with linear response based on asynchronous event detection," in *Circuit Theory and Design (ECCTD)*, 2015, pp. 1–4.

[34] P. Dudek, "Adaptive sensing and image processing with a general-purpose pixel-parallel sensor/processor array integrated circuit," in *Int. Workshop on Computer Architecture for Machine Perception and Sensing*. IEEE, 2006, pp. 1–6.

[35] R. Wagner, Á. Zarándy, and T. Roska, "High dynamic range perception with spatially variant exposure," in *IEEE Int. Workshop*, 2004.

[36] J. N. Martel, L. K. Müller, S. J. Carey, and P. Dudek, "Parallel hdr tone mapping and auto-focus on a cellular processor array vision chip," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2016, pp. 1430–1433.

[37] J. Fernández-Berni, R. Carmona-Galán, R. del Río, and A. Rodríguez-Vázquez, "High dynamic range adaptation for roi tracking based on reconfigurable concurrent dual sensing," in *IET Electronics Letters*, vol. 50, no. 24, 2014, pp. 1832–1834.

[38] M. Loose, K. Meier, and J. Schemmel, "A self-calibrating single-chip cmos camera with logarithmic response," *IEEE Journal of Solid-state circuits*, vol. 36, no. 4, pp. 586–596, 2001.

[39] N. Akahane, R. Ryuzaki, S. Adachi, K. Mizobuchi, and S. Sugawa, "A 200db dynamic range iris-less cmos image sensor with lateral overflow integration capacitor using hybrid voltage and current readout operation," in *IEEE Solid-State Circuits Conference*, 2006, pp. 1161–1170.

[40] E. Teixera, F. Santos, and A. Mesquita, "High fill factor cmos aps sensor with extended output range," in *IET Electronics Letter*, vol. 46, no. 25, 2010, pp. 1658–1659.

[41] C. Ma, D. S. S. Bello, C. Hoof, and A. Theuwissen, "High dynamic range hybrid pixel sensor," *Electronics letters*, vol. 47, no. 12, pp. 695–696, 2011.

[42] V. Majidzadeh, L. Jacques, A. Schmid, P. Vandergheynst, and Y. Leblebici, "A (256× 256) pixel 76.7 mw cmos imager/compressor based on real-time in-pixel compressive sensing," in *IEEE Int. Symposium on Circuits and Systems*, 2010, pp. 2956–2959.

[43] Y. Oike and A. El Gamal, "Cmos image sensor with per-column $\sigma\delta$ adc and programmable compressed sensing," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 318–328, 2012.

[44] M. Dadkhah, M. J. Deen, and S. Shirani, "Compressive sensing image sensors-hardware implementation," *Sensors*, vol. 13, no. 4, pp. 4961–4978, 2013.

[45] M. Wei, N. Sarhangnejad, Z. Xia, N. Gusev, N. Katic, R. Genov, and K. N. Kutulakos, "Coded two-bucket cameras for computer vision," in *European Conference on Computer Vision*, 2018, pp. 54–71.

[46] P. Lichtsteiner, C. Posch, and T. Delbruck, “A 128×128 120 db 15μs latency asynchronous temporal contrast vision sensor,” *IEEE journal of solid-state circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[47] C. Posch, D. Matolin, and R. Wohlgenannt, “A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds,” *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, 2010.

[48] T. Serrano-Gotarredona and B. Linares-Barranco, “A 128×128 1.5% contrast sensitivity 0.9% fpn 3 μs latency 4 mw asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers,” *IEEE Journal of Solid-State Circuits*, vol. 48, no. 3, pp. 827–838, 2013.

[49] J.-E. Eklund, C. Svensson, and A. Astrom, “Vlsi implementation of a focal plane image processor-a realization of the near-sensor image processing concept,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 3, pp. 322–335, 1996.

[50] F. Paillet, D. Mercier, and T. M. Bernard, “Second generation programmable artificial retina,” in *IEEE Int. ASIC/SOC Conference*, 1999, pp. 304–309.

[51] M. Ishikawa, K. Ogawa, T. Komuro, and I. Ishii, “A cmos vision chip with simd processing element array for 1 ms image processing,” in *IEEE Int. Solid-State Circuits Conference*, 1999, pp. 206–207.

[52] W. Miao, Q. Lin, W. Zhang, and N.-J. Wu, “A programmable simd vision chip for real-time vision applications,” *IEEE Journal of Solid-State Circuits*, vol. 43, no. 6, pp. 1470–1479, 2008.

[53] A. Lopich and P. Dudek, “An 80× 80 general-purpose digital vision chip in 0.18 μm cmos technology,” in *IEEE Int. Symposium on Circuits and Systems*, 2010, pp. 4257–4260.

[54] A. Rodríguez-Vázquez, R. Domínguez-Castro, F. Jiménez-Garrido, S. Morillas, A. García, C. Utrera, M. D. Pardo, J. Listan, and R. Romay, “A cmos vision system on-chip with multi-core, cellular sensory-processing front-end,” in *Cellular nanoscale sensory wave computing*. Springer, 2010, pp. 129–146.

[55] W. Zhang, Q. Fu, and N.-J. Wu, “A programmable vision chip based on multiple levels of parallel processors,” *IEEE Journal of Solid-State Circuits*, vol. 46, no. 9, pp. 2132–2147, 2011.

[56] J. Fernández-Berni, R. Carmona-Galán, and Á. Rodríguez-Vázquez, “Flip-q: A qcif resolution focal-plane array for low-power image processing,” in *Low-Power Smart Imagers for Vision-Enabled Sensor Networks*. Springer, 2012, pp. 67–109.

[57] J. N. Martel, L. K. Müller, S. J. Carey, and P. Dudek, “High-speed depth from focus on a programmable vision chip using a focus tunable lens,” in *IEEE Int. Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

[58] J. Chen, S. J. Carey, and P. Dudek, “Feature extraction using a portable vision system,” 2017.

[59] L. Bose, J. Chen, S. J. Carey, P. Dudek, and W. Mayol-Cuevas, “Visual odometry for pixel processor arrays,” in *IEEE Int. Conference on Computer Vision*, 2017, pp. 4604–4612.

[60] A. Chakrabarti, “Learning sensor multiplexing design through back-propagation,” in *Advances in Neural Information Processing Systems*, 2016, pp. 3081–3089.

[61] V. Sitzmann, S. Diamond, Y. Peng, X. Dun, S. Boyd, W. Heidrich, F. Heide, and G. Wetzstein, “End-to-end optimization of optics and image processing for achromatic extended depth of field and super-resolution imaging,” *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, p. 114, 2018.

[62] Y. Wu, V. Boominathan, H. Chen, A. Sankaranarayanan, and A. Veeraraghavan, “Phasecam3d—learning phase masks for passive single view depth estimation,” in *Int. Conference on Computational Photography (ICCP)*. IEEE, 2019, pp. 1–12.

[63] J. Chang and G. Wetzstein, “Deep optics for monocular depth estimation and 3d object detection,” in *ICCV*, 2019.

[64] J. Chang, V. Sitzmann, X. Dun, W. Heidrich, and G. Wetzstein, “Hybrid optical-electronic convolutional neural networks with optimized diffractive optics for image classification,” *Scientific reports*, vol. 8, no. 1, p. 12324, 2018.

[65] C. A. Metzler, H. Ikoma, Y. Peng, and G. Wetzstein, “Deep optics for single-shot high-dynamic-range imaging,” in *Conference on Computer Vision and Pattern Recognition*, 2020.

[66] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks,” in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[67] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Int. Conf. on Medical image computing and computer-assisted intervention*, 2015, pp. 234–241.

[68] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[69] H. K. Galoogahi, A. Fagg, C. Huang, D. Ramanan, and S. Lucey, “Need for speed: A benchmark for higher frame rate object tracking,” *arXiv preprint arXiv:1703.05884*, 2017.

Julien Martel is a postdoctoral research fellow in the Stanford Computational Imaging Group. His main interest is in the design of novel vision algorithms and systems coupling sensing and processing in-pixel.

Lorenz Müller is a post-doctoral research fellow in the Neuromorphic Devices and Systems group at IBM Research Rüschlikon. His main interest is in neurally inspired information processing.

Stephen Carey is a research fellow at the Microelectronics Lab at the School of Electrical and Electronics Engineering, The University of Manchester. His main interest is in the area of integrated circuit design in the development of novel sensor-processor systems.

Piotr Dudek is a Professor of Circuits and Systems in the School of Electrical and Electronic Engineering, The University of Manchester, leading the Microelectronics Design Lab. His research interest is in the area of integrated circuit design, especially vision sensors, cellular processor arrays, analogue and mixed-mode processing hardware, neuromorphic engineering and brain-inspired systems.

Gordon Wetzstein is an Assistant Professor of EE and, by courtesy, of CS at Stanford University, leading the Stanford Computational Imaging Lab and a faculty co-director of the Stanford Center for Image Systems Engineering.