2021 18th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE). Mexico City, Mexico. November 10-12, 2021

Edge computing SoC implementation of compressive sensing algorithm for single-pixel cameras

A. Manjarrés García[1], C. Osorio Quero[1], J. Rangel-Magdaleno[1], J. Martinez-Carranza[2] and D. Durini Romero[1]
[1] *Digital Systems Group - Electronics Department*
[2] *Computer Science Department*
Instituto Nacional de Astrofísica, Óptica y Electrónica. (INAOE)
Luis Enrique Erro 1, 72840 Tonantzintla, Puebla, Mexico
email:{manjarres, caoq, jrangel, carranza, ddurini}@inaoep.mx

***Abstract*—One of the most recent applications of compressive sensing is single-pixel cameras, where an image is generated by a reduced set of measurements. This new type of camera has already been implemented in many ways but always relying on a powerful CPU or GPU to run the reconstruction algorithms, limiting the portability and increasing the power consumption of the entire system. In this work, a well-known compressive sensing algorithm called TVAL3 is implemented in a Xilinx SoC embedded device. This algorithm has been used in multiples works of compressive sensing for single-pixel cameras due to its image reconstruction performance. The used device is a Zynq-7000 SoC that has an ARM CPU and allows hardware acceleration via FPGA. The hardware accelerator allows speed up the most time-consuming part of the algorithms to contrast the retard caused by the low-spec CPU. The proposed FPGA accelerator implementation was compared with the CPU-only version as regards quality reconstruction and speed. The results show a nearly 10x improvement in the consumption time and a no reduction in the quality of the reconstructions. The new implementation can be used to the initial phase of new, more portable, and faster single-pixel cameras.**

I. Introduction

Without any doubt, compressive sensing is and will remain a crucial technique with many applications. The reconstruction of a signal with a reduced number of samples brings new possibilities and also new problems. One of these new applications to compressive sensing theory is the single-pixel cameras [1]. These cameras use a single sensor to collect the reflected intensities of the scene after the projection of multiples patterns. These projections can be done using light sources with wavelengths different than the visible spectrum, like the near-infrared. This allows hyperspectral cameras, useful in applications like gas leakage detection [2]. Another important feature of single-pixel cameras is the ability to use the time-of-flight of the signals projected to estimate the deep of the objects on the scene, and build 3D images [3].

The single-pixel cameras use compressive sensing to reduce the number of patterns needed to reconstruct the signal. For example, a single pixel non-compressive-sensing approach that wants to produce a 64x64 image must generate and project 4096 orthogonal-based patterns. In addition, to collect and store the 4096 reflected intensities. These processes require a lot of time and effort, making single-pixel cameras less competitive against the traditional ones. With compressive sensing, this number of patterns is reduced drastically to fewer measurements than the Nyquist limit.

The field of single-pixel image reconstruction algorithms is constantly changing, with new algorithms increasingly focused on reducing computational cost and obtaining better quality reconstructions. Candès and Tao [4], [5] proposed some of the earliest work in this area, suggesting l_1-minimization to recover a sparse signal. l_1-magic [6] and basis pursuit denoising (BPDN) [7] are algorithms that apply this l_1-minimization. Other important kinds of algorithms are known as greedy algorithms, which iteratively attempt to find the local optimum at each stage by the signal decomposition into a linear combination of waveforms. In compressive sensing, Iterative Hard Thresholding (IHT) [8], Orthogonal Matching Pursuit (OMP) [9], and Compressive Sampling Matching Pursuit (CoSaMP) [10] are the most used greedy algorithms. The third category of algorithms commonly groups the total variation (TV) regularization algorithm. TV regularization set that the image gradient is sparse and try to recover the image from this convex problem. TV minimization by Augmented Lagrangian and ALternating direction ALgorithms (TVAL3) [12] and NESTA [13] are the commonly used algorithm for single-pixel that use TV minimization. Despite the large number of algorithms applied to single-pixel cameras, we will focus on TVAL3 because of its good performance [11].

Almost all the previous works in single-pixel imaging use a CPU or GPU station to reconstruct the image, limiting the overall speed to the network bandwidth. In complex systems where not just images are processes, the speed of the connections becomes the main bottleneck. An example of such a system is autonomous vehicles, in which 1 gigabyte of data is generated every second and requires real-time processing to make correct decisions [14]. This problem finds a solution in edge computing that enables the system to process the

978-1-6654-0029-9/21/$31.00 ©2021 IEEE

data at the network's edges, avoiding data movement. The main objective of this work is to provide an edge computing implementation of the TVAL3 algorithm that allows fast and portable cameras.

To implement the algorithm, we use a Zynq-7000 SoC as the development platform along with the accelerator kernel flow from Vitis. With our implementation, we obtain a 10x speedup in the reconstruction time for different image sizes and different compression ratios compared with the ARM-CPU version. Also, we evaluate the use of fixed-point arithmetic and find an almost 20 dB decrease in the peak signal-to-noise ratio (PSNR) when using this data type. To the best of our knowledge, this is the first attempt to implement a compressive-sensing algorithm focused on single-pixel cameras on an FPGA or even on an embedded platform. Although the results are far from real-time requirements, we hope to open up the opportunity for new designs focused on more portable and speed single-pixel cameras.

Section II, survey the TVAL3 algorithm, focused only on the most common version of it. Section III outlines our methods for deploy the algorithm in the embedded platform and how we implemented the accelerator. Section IV presents the relevant findings of the research.

II. TVAL3 Algorithm

The TVAL3 was proposed by Li in [12], as a combination of classical optimization methods applied to a total variation regularization problem in image reconstruction. In this section, we introduce the algorithm to set the basis of our work. First, the problem to solve is the total variation regularization applied to compressed sensing:

$$\min_u = \sum_i \|D_i u\|, \quad \text{s.t. } Au = b, \tag{1}$$

or with an extra auxiliary variable,

$$\min_u = \sum_i \|w_i\|, \quad \text{s.t. } Au = b \text{ and } D_i u = w_i, \tag{2}$$

where u is the image to reconstruct, $D_i u$ the discrete gradient of u at pixel i, A the measurement matrix, b the measure. To solve (1) Li in [12] proposes an algorithm that applied the alternating direction method to minimized their augmented Lagrangian function. The augmented Lagrangian function redefines the original constrained problem to a sequence of unconstrained subproblems. Applied to (2) the augmented Lagrangian is:

$$\begin{aligned}\mathcal{L}_A(w_i, u) = &\sum_i \left(\|w_i\| - \nu_i^T (D_i u - w_i) + \frac{\beta_i}{2} \|D_i u - w_i\|_2^2 \right) \\ &- \lambda^T (Au - b) + \frac{\mu}{2} \|Au - b\|_2^2\end{aligned} \tag{3}$$

where ν and λ are the Lagrangian multipliers that must be updated in each iteration, and β and μ are the penalties that also must be re-computed after each iteration. To minimize this new function Li applied the alternating direction method [15] to separate the problem into two sub-problems: u minimization and w minimization.

A. w minimization: Shrinkage-like Formulas

The minimization of w begin with the definition of the w-subproblem from the original augmented Lagrangian:

$$\min_{w_i} \sum_i \left(\|w_i\|_1 - \nu_i^T (D_i u_k - w_i) + \frac{\beta_i}{2} \|D_i u_k - w_i\|_2^2 \right) \tag{4}$$

where k denotes the index of the k-iteration for the inner iteration loop. To solve this problem the original work proposes a shrinkage-like formula:

$$w_{i,k+1} = max \left\{ \left| D_i u_k - \frac{\nu_i}{\beta_i} \right| - \frac{1}{\beta_i}, 0 \right\} sgn\left(D_i u_k - \frac{\nu}{\beta} \right). \tag{5}$$

B. u minimization: One-step Steepest Descent

After computing $w_{i,k+1}$, is possible to compute u_{k+1} using the one-step steepest descent scheme. Like before first the authors define u-subproblem

$$\begin{aligned}\min_u \sum_i &\left(-\nu_i^T (D_i u - w_{i,k+1}) + \frac{\beta_i}{2} \|D_i u - w_{i,k+1}\|_2^2 \right) \\ &- \lambda^T (Au - b) + \frac{\mu}{2} \|Au - b\|_2^2,\end{aligned} \tag{6}$$

and then to apply the steepest descent scheme, they compute the gradient of the above expression

$$\begin{aligned}d_k(u) = &\sum_i \left(\beta_i D_i^T (-D_i u - w_{i,k+1}) - D_i^T \nu_i \right) \\ &+ \mu A^T (Au - b) - A^T \lambda.\end{aligned} \tag{7}$$

Finally, to accelerate the process, TVAL3 only computes one step of the steepest descent scheme, so it is necessary to define an aggressive step length. To achieve this the author uses the Barzilai-Borwein method [16]

$$\alpha_k = \frac{s_k^T s_k}{s_k^T y_k}, \tag{8}$$

where $s_k = u_k - u_{k-1}$ and $y_k = d_k(u_k) - d_k(u_{k-1})$, followed by the non-monotone line search method proposed by Zhang and Hager in [17]. Algorithm 1 is an overview of the complete TVAL3 build by Li.

III. SoC Implementation

A. Platform

Before starting with the description of the accelerator, we will describe the platform that we use to justify some of the software decisions. The accelerator was implemented on the MicroZed®-7010 board, a device based on the Xilinx Zynq-7000 SoC. This SoC combines dual-core ARM Cortex-A9 processors with 28nm Artix-7 based programmable logic. To facilitate the deployment, we used the Vitis™ acceleration kernel flow. We started by designing the software platform, as it is not officially supported. The platform runs Linux and uses the OpenCL™ API and the Linux-based Xilinx Runtime (XRT) to schedule kernels and control data movement.

In table I, we present the total amount of resources used to the platform, these resources are used to define the interfaces

Algorithm 1: TVAL3 Algorithm (Adapted of [12])

Initialize $\nu_i^0, \beta_i^0, \lambda^0, \mu^0$, *and starting points* w_i^0, u^0 *for all* i;
while *outer stopping criteria unsatisfied* **do**
  Set $w_{i,0}^{k+1} = w^k$ *and* $u_0^{k+1} = u^k$;
  Initialize $0 < \delta, \rho, \eta < 1$;
  Set $Q_0 = 1$ *and* $C_0 = \mathcal{L}_A(w_{i,0}, u_0)$;
  while *inner stopping criteria unsatisfied* **do**
    Compute $w_{i,j+1}$ *based on shrinkage-like formula*;
    Set α_j *through BB-like formula*;
    while *nonmonotone Armijo condition unsatisfied* **do**
      Backtrack $\alpha_j = \rho\alpha_j$;
    Compute u_{j+1} *by one-step steepest descent method*;
    Set C_{j+1};
  Update multipliers $\left(\nu_i^{k+1}, \lambda^{k+1}\right)$;
  Choose new penalty parameters $\beta_i^{k+1} \geq \beta_i^k$ *and* $\mu^{k+1} \geq \mu^k$;
return u;

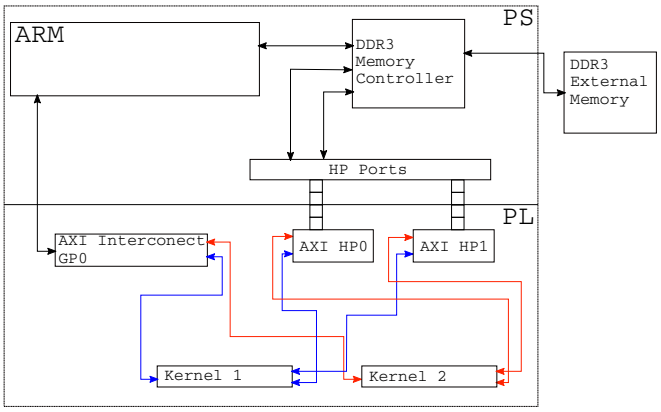


Fig. 1. The general structure of the platform. The upper piece, the PS, represents the blocks inside the processing system part. The PL bottom section represents the internal block of the FPGA.

that handle different types of requests by the host. The last row in the table indicates the number of resources available for implementing the kernels. Figure 1 represents the general structure of the platform. The processing system (PS) uses general-purpose AXI ports to control the execution and data transfers of the Programmable Logic (PL) side. The FPGA uses high-performance (HP) ports to transfer data to the external memory. On the MicroZed board, this external memory is a 1 GB DDR3 memory.

B. Kernels

As discussed in the previous section, the TVAL3 algorithm solves the compressive sensing problem by the Alternating Direction Algorithm of the Lagrangian. This alternating direction algorithm split the Lagrangian problem into two new subproblems, resolved by the steepest descent method and shrinkage-like formulas.

These two subproblems include the computation of finite differences, vector scaling, and matrix-vector multiplications. Since the latter operation is the most computationally expensive, its acceleration will dominate the performance improvement.

We use C++ to write the two kernel codes for the two matrix-vector operations. The code for one kernel is provided in listing 1, the other kernel has a similar structure. We select C++ mainly to provide a scalable implementation that can be reconfigured to any size of the operands and is capable of being deployed on any Vitis platform. The two inputs and the output are transferred over two separate High Performance (HP) AXI ports to the global memory. In the matrix-vector product, the matrix was arranged in column-major order, allowed us to avoid dependencies in the accumulator and achieve an initiation interval (II) of 1. Also, we employed dataflow optimization to improve the kernel performance by task-level parallelism. The kernels then executes three tasks (read, compute and write) in a dataflow pipelining way. The design uses first-in-first-out (FIFO) memory as a data transferring channel between each of the tasks.

TABLE I
RESOURCE CONSUMPTION OF MICROZED PLATFORM. THE USER BUDGET IS THE REMAINING NUMBER OF RESOURCES AVAILABLE FOR THE ACCELERATOR CORES.

	LUT	LUTAsMem	REG	BRAM (36Kb)	DSP
Total	6929	899	9958	13	0
Available	17600	6000	35200	60	80
Utilization (%)	39.3	14.9	28.3	21.7	0.0
User Budget	10671	5101	25242	47	80

Listing 1
MATRIX-VECTOR MULTIPLICATION HLS CODE

```
for (int j = 0; j < FEATURES; j++) {
 float b = b_buffer[j];
 for (int i = 0; i < MEASURES; i++) {
  #pragma HLS PIPELINE II=1
  A_stream >> a;
  float prev = (j==0) ? 0 : c_buffer[i];
  c_buffer[i] = prev + (b * a);
 }
}
```

Figure 2 shows the kernel organization with the main blocks of the computing task. We use internal BRAM to implement the accumulator (N) and a temporal array that stores the input vector (M). In 32-bit float-point precision, the adder and the multiplier used a total of 5 DSP48E modules. The latency in cycles of the module (thanks to the previously mentioned optimizations) is nearly the value of elements on the matrix. In the next section, we evaluate the performance of our implementation by comparison against a full CPU version

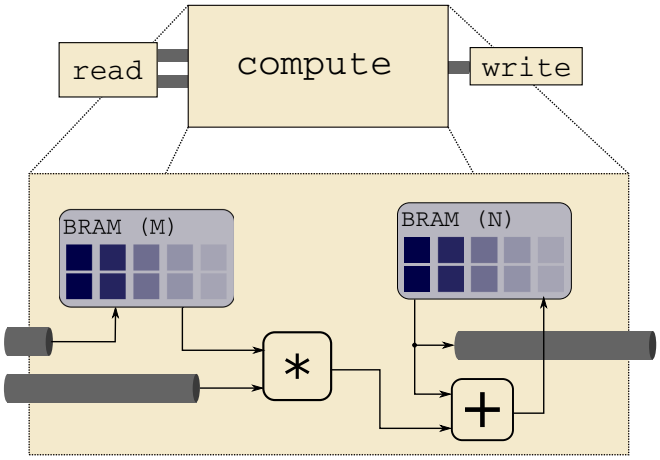


Fig. 2. Proposed structure for matrix-vector kernel implementation. The gray cylinders represent the FIFO connections inside the dataflow optimization.

that also runs on the MicroZed board but does not use the FPGA.

IV. Results

To measure the performance of our design, we test the TVAL3 algorithm with and without the accelerator in three different scenarios: First, we wanted to evaluate the changes in resource consumption and runtime for three data accuracies and use these results to select the best accuracy. Then, with this accuracy, we evaluated the performance by setting the image size to 64x64 pixels and reconstructing it with five different compression levels. Finally, we set the compression level to 70% and changed the image size. All the experiments run on the same board and try to reconstruct the phantom image (Figure 3a) with the same TVAL3 parameters.

Table II shows the performance of three implementations varying the data type. The time spends in all the implementation is almost the same, but the PSNR of the reconstruction has a significant decrease when uses the fixed-point data type. This decrease is meaningful when we look at the consumption resources. Since in 4 resources categories, the fixed-point implementation has less consumption, these decreases are not above 2.5%. Meanwhile, the LUT amount increase by more than 20.00 %. These results show that a fixed point implementation does not improve the performance of the accelerator and the use of float arithmetic is most justified.

Then, we test the time performance of the accelerator over multiple compression rates. Figure 4 illustrates the results compared with the full CPU version, showing that the hardware accelerator implementation improves the time in all the compression rates in at least one magnitude order. In terms of image quality, the full CPU and the hardware accelerator implementations present the same results. It is relevant to note that the last two compression levels (75 and 70) finish the algorithm by the tolerance error limit, while the other stop by the max iteration criteria. Figure 3 also illustrates a quality example of the reconstruction using different compression rates.

TABLE II
Resource consumption and performance of three TVAL3 implementations, with different data types. The percentage of the resource consumption is calculated relative to the platform user's budget and not on the full number of SoC elements.

	32-Float	Q9.32	Q8.27
Latency 1	4199454	4199456	4199453
Latency 2	4199454	4199456	4199453
Fmax (MHz)	136.99	136.99	136.99
Reconst Time (s)	35.35	35.38	32.51
PSNR (dB)	55.9	41.5	31.9
LUT	3739 [35.04%]	6633 [62.16%]	6198 [58.08%]
LUTAsMem	432 [8.47%]	440 [8.63%]	410 [8.04%]
REG	5718 [22.65%]	6830 [27.06%]	6366 [25.22%]
BRAM	12 [25.53%]	12 [25.53%]	10 [21.28%]
DSP	10 [12.50%]	8 [10.00%]	8 [10.00%]

Finally, we evaluated the implementation at three different image sizes, 16x16, 32x32, and 64x64, all with a compression level of 70%. Figure 5 shows the results also compared to the CPU version. The findings confirm the improvement of the FPGA version over the CPU version, even at small image sizes.

V. Conclusion

Compressive sensing algorithms for a new application like single-pixel cameras represent a rich field for embedded development. It was found that a small system can run complex reconstruction algorithms and improve the time performance by hardware acceleration. Also, the fixed-point implementation shows no improvement over the floating arithmetics and a decrease in the quality reconstruction. These last results can be overcome in future works by using some artifact that compensates for the precision issues. Our findings could be applied to other FPGA+CPU systems, providing better results, and further improving the single-pixel field in terms of portability, power consumption, and runtime.

References

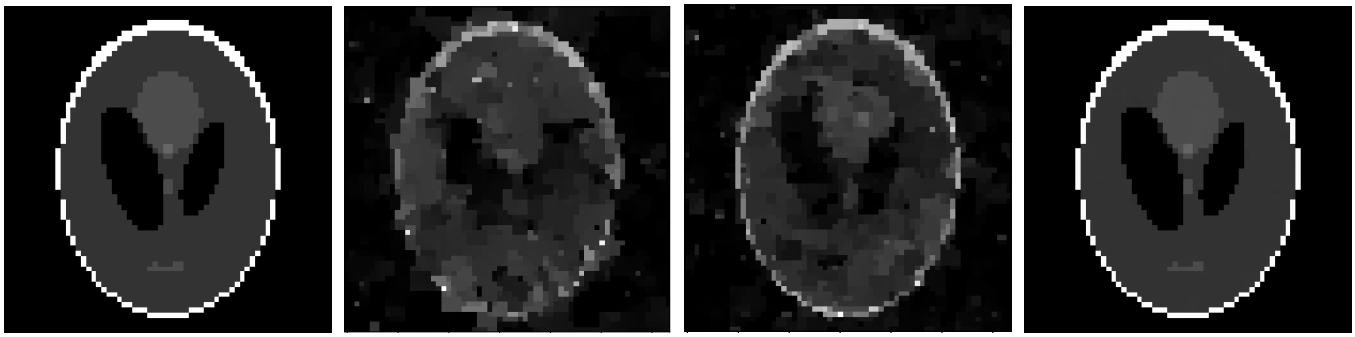
[1] M.F. Duarte, M.A. Davenport, D. Takhar, J.N. Laska, T. Sun, K.F. Kelly, R.G. Baraniuk, "Single-pixel imaging via compressive sampling," *IEEE Signal Processing Magazine*., vol. 25, pp. 83–91, March 2008, doi: 10.1109/MSP.2007.914730.

[2] G.M.Gibson, B.Sun, M.P.Edgar, D.B.Phillips, N.Hempler, G.T. Maker, G.P.A. Malcolm, and M.J.Padgett, "Real-time imaging of methane gas leaks using a single-pixel camera,". *Opt. Express*, vol. 25, no. 4, pp.2998–3005, 2017.

[3] M.P. Edgar, M. Sun, G.M. Gibson, G.C. Spalding, D.B. Phillips, and M.J. Padgett. "Real-time 3D video utilizing a compressed sensing time-of-flight single-pixel camera," in *Proc. SPIE 9922, Optical Trapping and Optical Micromanipulation XIII*, San Diego, California, Sep. 2016, [Online]. doi: 10.1117/12.2239113

[4] E. J. Candes and T. Tao, "Near-Optimal Signal Recovery From Random Projections: Universal Encoding Strategies?," in IEEE Transactions on Information Theory, vol. 52, no. 12, pp. 5406-5425, Dec. 2006, doi: 10.1109/TIT.2006.885507.

[5] E. J. Candes, J. Romberg and T. Tao, "Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information," in IEEE Transactions on Information Theory, vol. 52, no. 2, pp. 489-509, Feb. 2006, doi: 10.1109/TIT.2005.862083.



(a) Original (b) PSNR: 16.723684 dB (c) PSNR: 18.983104 dB (d) PSNR: 53.198341 dB

Fig. 3. Example of the image reconstruction solution, using three different number of measures. (a) represent the original image sampled by the sensing matrix. (b-d) are the three reconstructions for 409, 819 and, 1024 respectively.

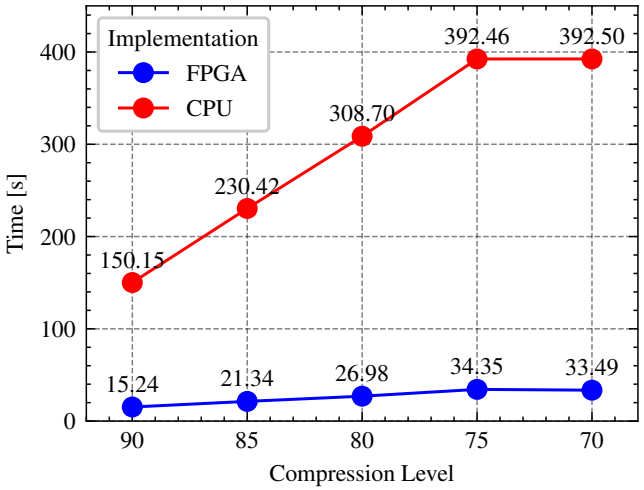


Fig. 4. Time consumption for the image reconstruction using five different levels of compression. We calculate the performance for the implementation using only the PS part of the SoC and using the FPGA to accelerate the implementation.

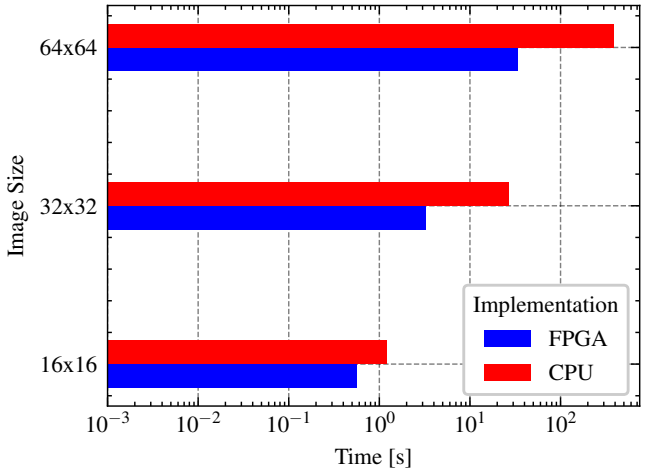


Fig. 5. Time consumption for the image reconstruction using three different images sizes and the same level of compression (70%). We calculate the performance for the implementation using only the PS part of the SoC and using the FPGA to accelerate the implementation.

[6] E. J. Candes, J. Romberg, l1-magic: Recovery of sparse signals via convex programming. URL: www.acm.caltech.edu/l1magic/downloads/l1magic.pdf, 2005.

[7] S. S. Chen, D. L. Donoho, and M A. Saunders. Atomic decomposition by basis pursuit. SIAM Journal on Scientific Computing, 20(1):33–61, 1998.

[8] T. Blumensath and M. E. Davies. "Iterative hard thresholding for compressed sensing," *Applied and Computational Harmonic Analysis*, vol. 27, no.3, pp. 265–274, 2009.

[9] Y. C. Pati, R. Rezaiifar and P. S. Krishnaprasad, "Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition," in Proc. of *27th Asilomar Conference on Signals, Systems and Computers*, 1993, pp. 40-44, vol.1, doi: 10.1109/ACSSC.1993.342465

[10] D. Needell and J. A. Tropp. "Cosamp: Iterative signal recovery from incomplete and inaccurate samples," Applied and Computational Harmonic Analysis, vol. 26, no. 3, pp. 301-321, 2009.

[11] Y. Kang, YP. Yao, ZH. Kang, L. Ma,TY. Zhang, "Performance analysis of compressive ghost imaging based on different signal reconstruction techniques," *JOSA A*. vol. 32, no. 6, pp. 1063-1067, Jun 2015, doi: 10.1364/JOSAA.32.001063. PMID: 26367039.

[12] C. Li, "An Efficient Algorithm For Total Variation Regularization with Applications to the Single Pixel Camera and Compressive Sensing," M.S. thesis, Rice Univ., Texas, Sep. 2009. Accessed on: Abr. 28, 2020. [Online]. Available: https://www.caam.rice.edu/~optimization/L1/TVAL3/CBL_master_thesis.pdf

[13] S. Becker, J. Bobin, and E. Candès, "NESTA: A Fast and Accurate First-order Method for Sparse Recovery,"*SIAM J. Imaging Sci*, vol. 4 no. 1, pp. 1-39, 2011.

[14] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges," in *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637-646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.

[15] D.W. Peaceman, and H.H. Rachford, "The Numerical Solution of Parabolic and Elliptic Differential Equations." *Journal of the Society for Industrial and Applied Mathematics*, vol. 3, no. 1, pp. 28–41, Mar. 1955. Accessed on: Oct. 05, 2020. [Online]. Available www.jstor.org/stable/2098834.

[16] J. Barzilai and J.M. Borwein, "Two-point step size gradient methods", IMA Journal of Numerical Analysis, vol. 8, no. 1, pp. 141–148, Jan., 1988.

[17] H. Zhang and W.W. Hager, "A nonmonotone line search technique and its application to unconstrained optimization", *SIAM Journal on Optimization* vol. 14, no. 4, pp. 1043–1056, 2004.