COMPRESSIVE LASER RANGING WITH EMBEDDED SYSTEMS

by

Pushkar Pradeep Pandit

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 2015

DEDICATION

To my family.

ACKNOWLEDGEMENTS

I would like to express the deepest gratitude and appreciation to my mentor Zeb Barber, for his continued guidance and support over the past three years. A helpful, dedicated and hardworking genius like you is someone I will try to emulate throughout my career. I want to thank Randy Babbitt, Jason Dahl, Cal Harrington, Tia Sharpe, Ana Baselga, Krishna Rupavatharam, Russell Barbour, Sue Martin and Diane Harn for their help and advice throughout my time at Spectrum lab. I would like to acknowledge Dr. Joseph Shaw and Dr. Ross Snider for their guidance through graduate school. I would also like to acknowledge the Montana Board of Research and Commercialization Technology for funding this research. Finally, I would like to thank my friends and loved ones for their love and support from around the world.

TABLE OF CONTENTS

TABLE OF CONTENTS CONTINUED

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES CONTINUED

ABSTRACT

Compressive sensing is a signal processing technique that has recently come to the forefront due to its ability to work around the well-known Shannon-Nyquist-Whittaker sampling theorem by exploiting certain properties in real world signals. This thesis will explore the theory behind compressive sensing and demonstrate its implementation toward laser ranging, cumulatively known as compressive laser ranging. Experiments were set up using electronic and photonic devices combining the theory behind compressive sensing and laser ranging and successful results measuring distances to multiple targets were obtained. The experimental setup was also implemented on an FPGA in an effort to create a compact laser ranging system.

INTRODUCTION

Background

We live in the age of "big data" created by the digital infrastructure our world is now based upon. Some people may or may not have come across that term, but regardless, everyone is a part of it and is contributing to it. It refers to the sheer amount of data produced by our sensor systems that include anything from simple temperature sensors in homes to the multi-megapixel cameras everyone carries around in their pockets. A couple of decades ago the largest bottleneck for digital systems was the sensors itself. In general the signal processing and information storage infrastructure was far superior to data acquisition. However, recent advances in sensor hardware have exponentially increased the data produced and the price of owning such sensors has dropped drastically making it possible for everyone to own them. Soon, we are going to face a problem of having too much data to deal with, which might seem advantageous at first, but is most probably going to overwhelm the communication, signal processing and storage infrastructure. To put the idea in perspective let us look at some numbers from the past decade. In 2010, the amount of data generated was in the ballpark of 10^{20} bytes[1] and in the last 5 years we have produced over 90% of the data generated ever and this amount is expected to increase over tenfold in the next 10-15 years. In 2007, we reached a critical milestone where we produced more data than we had storage available in the world; and in 2011, we produced twice as much data as we had storage available and the rate is growing exponentially.[1, 2]

The data deluge has become particularly difficult to manage as we have reached the diffraction limit in lithography for microfabrication of storage devices. Even with innovative advances in these techniques to keep up with Moore's law, the rate of production of storage media is lagging severely to the rate of data acquisition and the gap is expanding. With this scenario it is quite clear that steps need to be taken such that information is not lost due to lack of storage. Loss of data might not seem like a critical problem when it concerns civilian infrastructure, but it can be a major problem for applications in military reconnaissance and large scientific experiments such as the ones performed at CERN and astronomical observatories.[3]

Let us consider the traditional process we have currently to acquire data and store it. We can see from Figure 1 that we have a process where we acquire the data through a sensor network, digitize it, compress it and then store it.

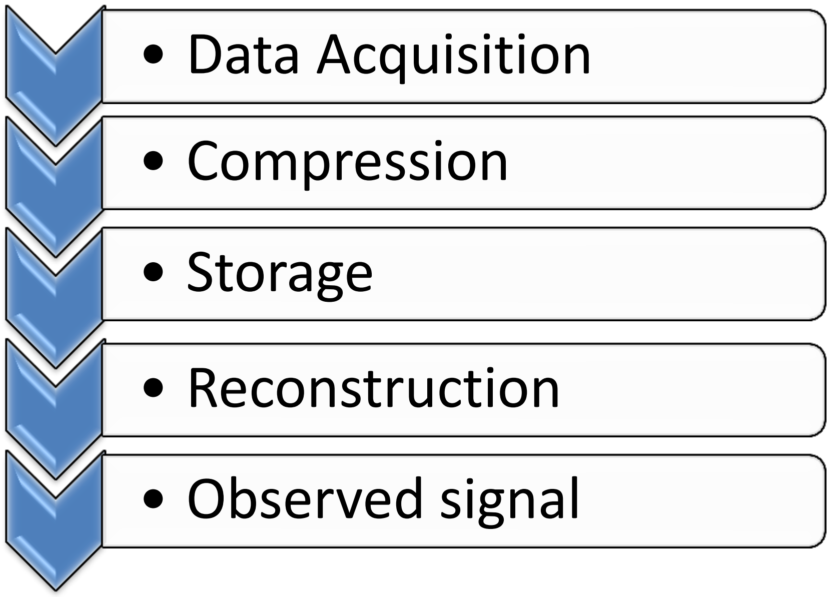

Figure 1. Conventional way of acquiring data, storing it and reconstructing it for use

When we want to view the data, we decompress the stored compressed data to reconstruct the original analog data. Compression is the step where we throw out a lot of the data (regardless of any information loss) and store only certain important coefficients in the data that will help reconstruct it using proper algorithms. For example, JPEG is an algorithm for the compression and reconstruction of digital images. Currently, compression of the acquired data is the best way for us to make our storage systems more efficient.

The process of first acquiring a huge amount of data and then compressing it seems quite wasteful. It would be more efficient if we could collect compressed data to begin with. That is the modus operandi on which this paper is based and we call it "compressive sampling" or "compressive sensing". For several decades, the collection of information, which we call "sampling," has been governed by the Shannon-Nyquist-Whittaker Sampling theorem. This theorem states that perfect reconstruction of a signal is possible only when the sampling frequency is greater than twice the maximum frequency of a bandlimited signal; furthermore, by the fundamental theorem of linear algebra, the number of collected samples (measurements) of a discrete finite-dimensional signal should be at least as large as its length (its dimension), to ensure perfect reconstruction. Almost every piece of technology today is designed with these constraints. Data sampled in this manner is eventually compressed and stored as we have discussed in the introduction.

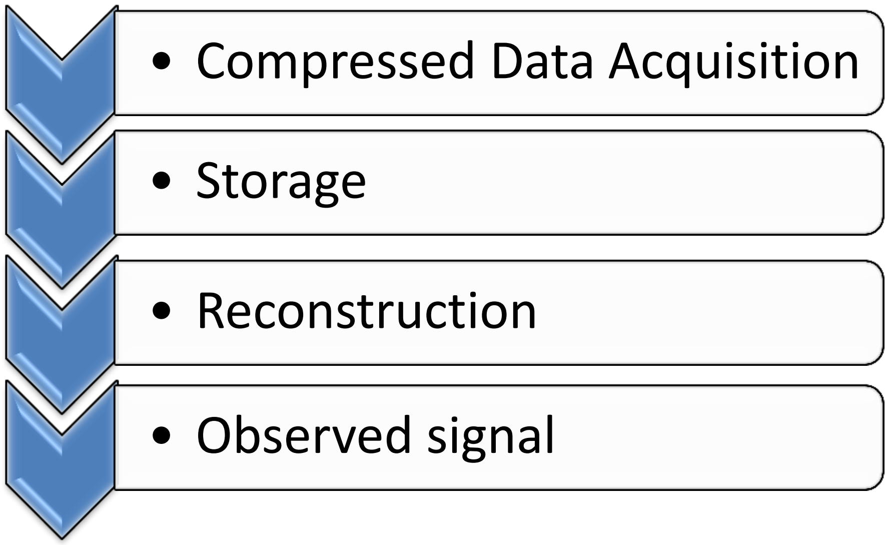

Figure 2. Compressive sensing (CS) paradigm. Data acquisition and compression are done simultaneously.

Compressed sensing (CS) is a new sensing paradigm where we try to combine the compression step and the sensing step for a more efficient system. Acquiring compressed data implies that we would be sampling data in violation of the Shannon-Nyquist-Whittaker sampling theorem since we would be taking fewer measurements than are required for perfect reconstruction. However, that idea is precisely what is explored in this thesis. We take compressed measurements to accurately reconstruct the object under certain critical assumptions that are trivial in practice. One of the first, well-publicized demonstrations of this technique is the single pixel camera work of the Digital Signal Processing group at Rice University led by Richard Baraniuk.[4] An illustration of the demonstration is shown in Figure 3.

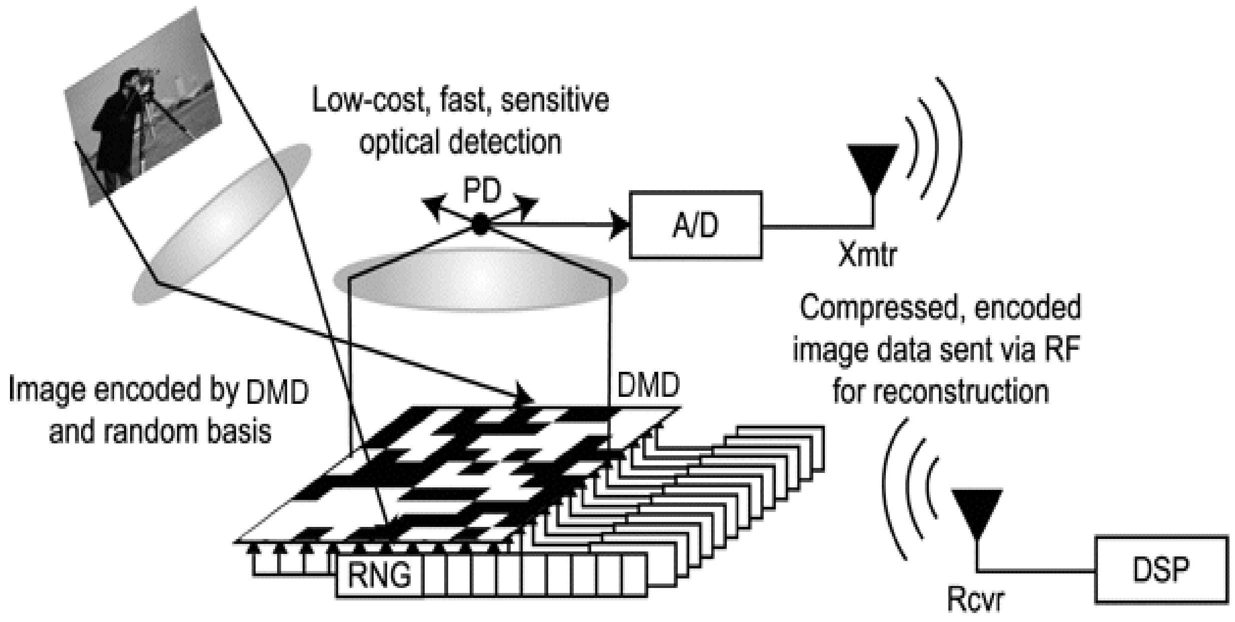
Low-cost, fast, sensitive optical detection
PD
A/D
Xmtr
Compressed, encoded image data sent via RF for reconstruction
Image encoded by DMD and random basis
DMD
RNG
Rcvr
DSP

Figure 3. Single Pixel Camera (Image courtesy of Rice Single-Pixel Camera Project, http://dsp.rice.edu/cscamera)[5]

The camera takes a number of measurements that are much less than the actual pixel density required by conventional cameras. A lens focuses the light from a scene onto a DMD (Digital Micromirror Device) which reflects it onto a detector through a lens. The DMD reflects light based on a pattern provided to it, in this case a random pattern. Random pixels on the DMD reflect light while the others do not. In this fashion a small number of measurements are taken with different random patterns on the DMD.

As an example, results from the Rice University single-pixel camera are shown in Figure 4. The size of the actual image and the number of measurements taken are shown. It can be seen that the image is reasonably well reconstructed using a fraction of the number of measurements required by the Shannon-Nyquist Sampling theorem. It is important to note that compressive sensing works under a necessary underlying assumption that the image is sparse in some basis. Sparsity is an important part of the compressive sensing paradigm and will be discussed in detail.

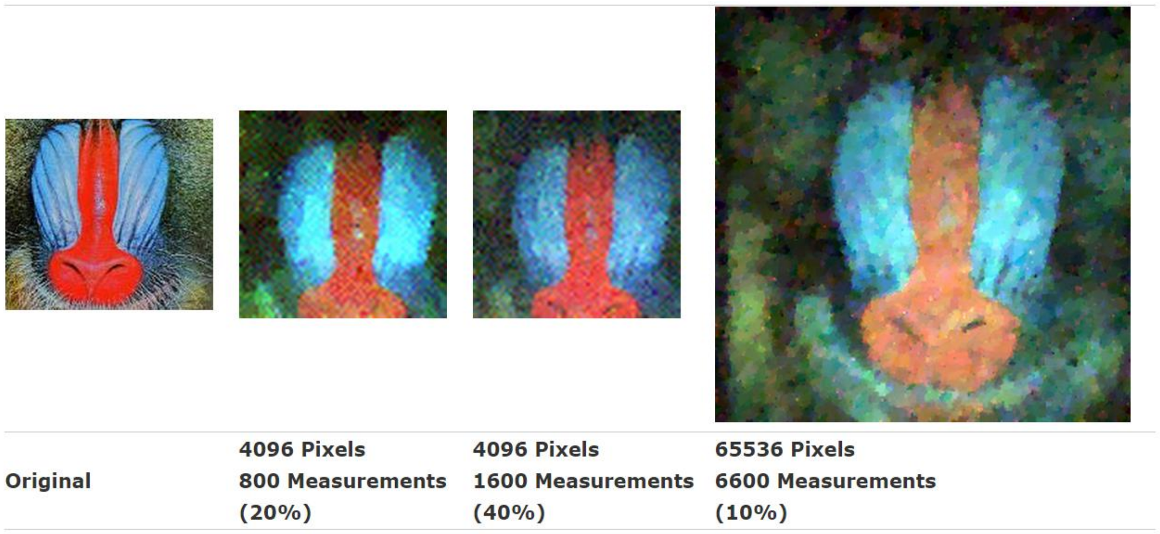

Figure 4. Results from single pixel camera for different number of measurements. (Image courtesy of Rice Single-Pixel Camera Project, http://dsp.rice.edu/cscamera)[4]

In this thesis we have taken this idea of compressive sensing and used it for range measurements. We demonstrate an idea of doing laser ranging in a novel way, much like the single-pixel camera demonstrated an innovative method of doing imaging.

Motivation

Laser ranging, also known as Laser Detection And Ranging (LADAR) or Light Detection And Ranging (LIDAR), is a way of measuring distances, in which a laser is used as an illumination source and the reflected light is analyzed to determine the distance or “range.” It is a similar process as radar, only using lasers instead of radio waves.

Conventional pulsed laser ranging systems consist of a laser that emits a pulse that reflects from a target and intervening air molecules and airborne particles and is then collected on a detector. The time delay between transmission of the light pulse and reception of the light scattered from the target is measured accurately to calculate the time it takes the light to travel to the target and back. Using this travel time with the speed of light, and the index of refraction of the medium through which the light travels, the range to an object or a scene can be calculated. The system is shown in Figure 5.

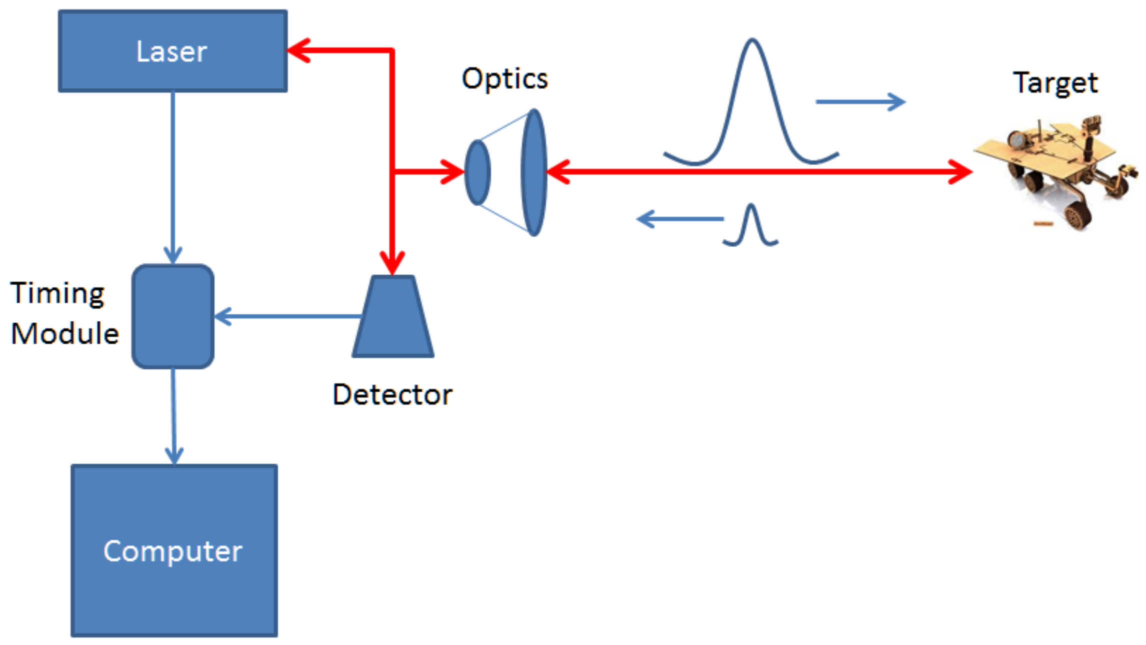

Figure 5. A block diagram illustrating a conventional ladar system.

Laser ranging systems can be found in differing complexity in several applications. A recent application has been in self-driving cars that use a ladar scanner for 3D rendering of the surrounding for navigation and obstacle detection.[6] Some of the most robust ladar systems can be found on airplanes and satellites that do terrain mapping as well as in aerosol detection and atmospheric study.[7–9] Ladars are also being used for long range chemical detection such as explosives, narcotics etc.[10–13] There are numerous applications for ladar and they will continue to grow in the future.

In this thesis I present a novel laser ranging system developed at Spectrum Lab, Montana State University. This new approach, known as Compressive Laser Ranging (CLR) uses the principles of compressive sensing for laser ranging and offers significant advantages over some aspects involved in conventional laser ranging such as high bandwidth analog waveform generation and detection.[14]

Outline

Chapter 2 of the thesis starts off by discussing the compressive sensing paradigm and some important assumptions such as sparsity that make the process possible. Next, I discuss reconstruction of the compressively sampled data and the algorithms used for this project: OMP (Orthogonal Matching Pursuit) and CoSaMP (Compressive Sampling Matching Pursuit). Chapter 3 provides a brief theory of laser ranging and discusses the application of the compressive sensing paradigm to laser ranging which we call compressive laser ranging (CLR). In this chapter the experimental setup is described, along with some results and discussion. Chapters 4 and 5 discuss the implementation of experimental setup on an FPGA to create a compact laser ranging system before wrapping up with the conclusions.

COMPRESSIVE SENSING

Background

The foundations of compressive sensing have been around for over a century. Basics of compressive sensing such as sparse approximations and harmonic analyses have been studied as far back as 1907 in a paper by Schmidt,[15] which explored Singular Value Decomposition (SVD), which happens to be an integral algorithm for the compressive sensing recovery process. However, the real advent of the compressive sensing era was heralded in the last decade by a couple of important papers, one by David Donoho[16] and the other by Emmanuel Candes, Justin Romberg, and Terence Tao[17], which not only provided algorithms but also demonstrated them with applications. Since then the field of compressive sensing has gathered steam and has found applications from biomedical sensing[18] to seismology[19,20] and astronomy[3].

The Compressive Sensing Problem

In the signal processing world, data frequently are digitized and collected linearly and are used to reconstruct the intended signal from discrete measurements. It is fairly common that the system can be modelled with a system of linear equations. This is true with the general compressive sensing problem, which is described by a simple linear equation that relates the measurement vector *y* to the signal of interest *x* through a measurement matrix *A*:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

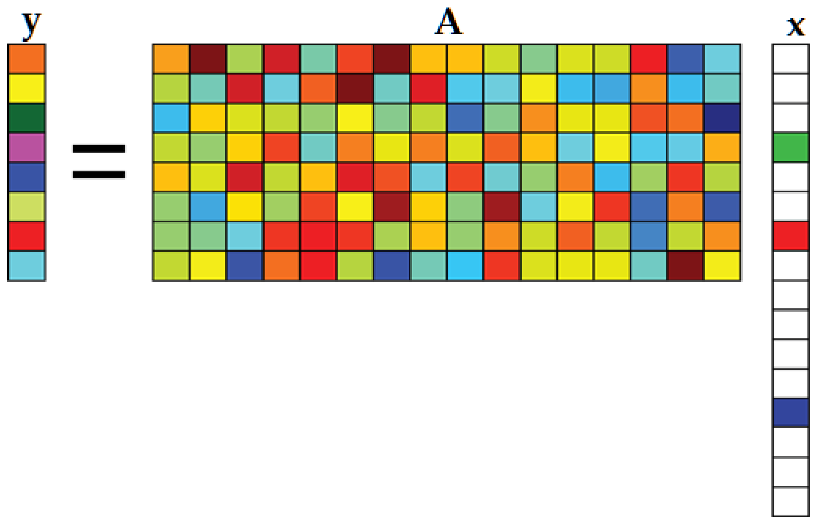

Figure 6. Compressive sensing problem. The measurement vector y is the product of the sensing matrix A and the sparse signal x.

By the laws of linear algebra, the measurement matrix A has dimensions mxn, where m is the number of measurements in y and n is the number of elements in x, as indicated in Figure 6. In the simplest case this is not a major problem, as long as A is a reasonable matrix. Since we are doing compressive sensing, in our case $m << n$. As soon as we apply this condition, the system becomes highly underdetermined, with infinitely many solutions and no obvious reconstruction. However, an important assumption we have been mentioning is sparsity. As long as x is a sparse vector, that is to say, it only has a limited number k of non-zero elements with $k \ll n$, reconstruction of the original signal x from the measurement vector y and the measurement matrix A is possible. The measurement matrix A also needs to be designed in such a manner that it can capture all the information about the signal x. The matrix A can be designed to conform to the conditions that are required for the process to succeed, although there are certain conditions that need to be fulfilled for it to be optimal. Because assuming anything about an unknown signal can be a dangerous practice, assuming that a signal is sparse in sensing might seem like a long shot; however, we will see in the next section it is not a terrible assumption to make.[21]

Sparsity

A signal is said to be sparse if majority of its components are zero. A mathematical definition can be expressed using the concept of,

the support of a vector x, which is the index set of its nonzero entries given by,

$$supp(x) := \{j \in [N] : x_j \neq 0\}.$$

The vector x is called s-sparse if at most s of its entries are nonzero, i.e.,

$$||x||_0 := card(supp(x)) \leq s.$$

Here the operator $card$ is the cardinality of the support of x, i.e., it counts the non-zero indices of x which are defined by $supp$. $||x||_0$ is the 0 norm of the signal x, where the norm in general is defined by

$$||x||_p := \left(\sum |x|^p\right)^{\frac{1}{p}}.$$

In reality, many real world signals can be approximated well by sparse signals in some basis. This sparse approximation of compressibility of real-world signals is the reason compression algorithms such as JPEG, MPEG etc. are able to function accurately. In the case of JPEG, the sparse basis is the discrete cosine basis and compression is achieved by storing only the largest discrete cosine coefficients. The remaining coefficients are set to zero giving rise to the compression. An example of a compressed image is shown in Figure 7 where only 10% of the total coefficients are stored and the remaining 90% are set to zero, effectively discarding them.

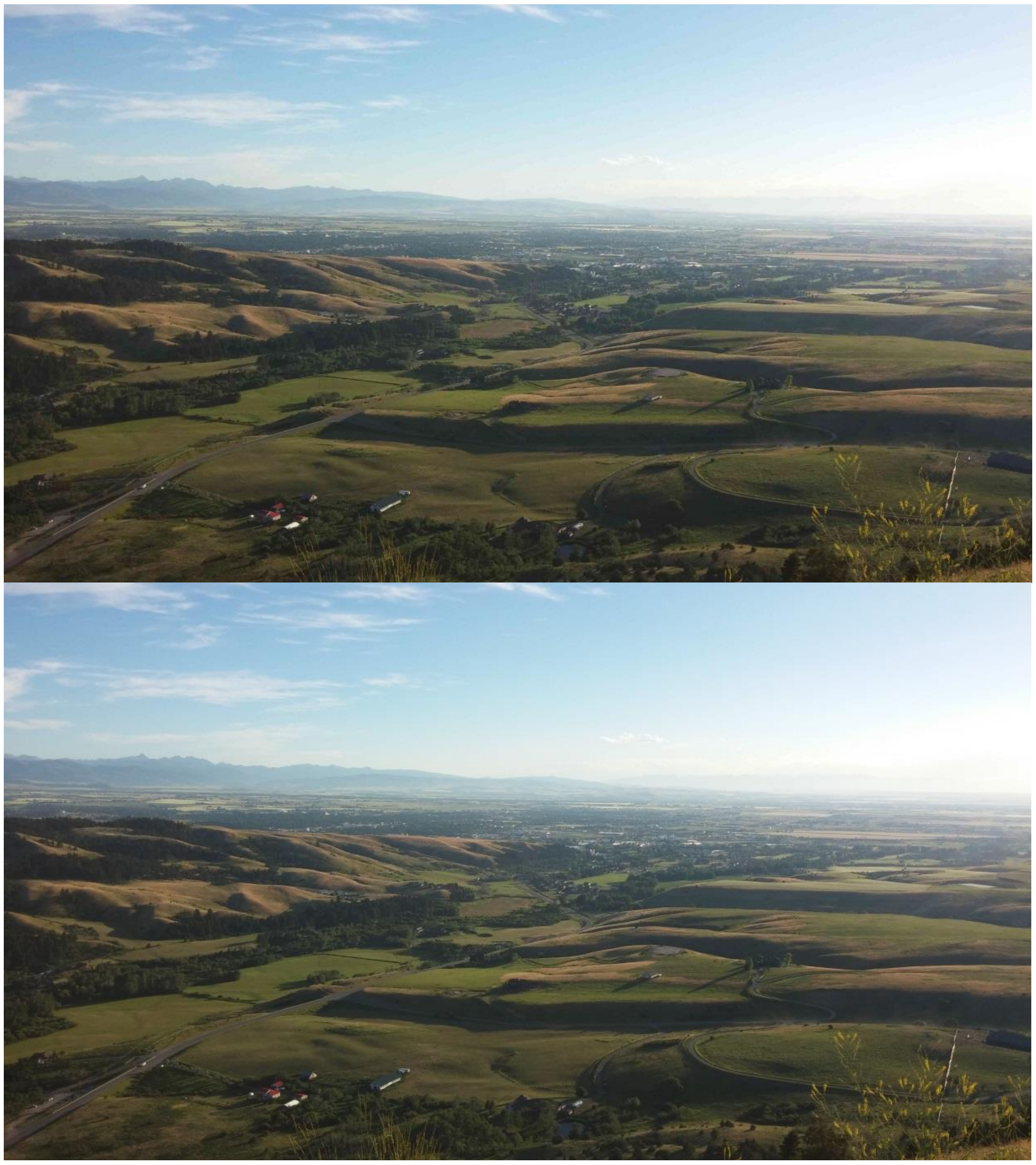

Figure 7. Original image (top) and reconstructed image (bottom) with 90% of the coefficients set to zero, effectively reducing the image size by 90%.

Thus the picture in Figure 7 is quite sparse and so are most other pictures and sets of data that we come across, some even more sparse where we can set more than 99% of the coefficients to 0. In such cases, there is no substantial loss of quality in the compressed version compared to the original image. A simple example of a non-intuitive sparse signal can be seen in Figure 8 where a signal is sparse in the Fourier domain but does not appear to be so in the time domain.

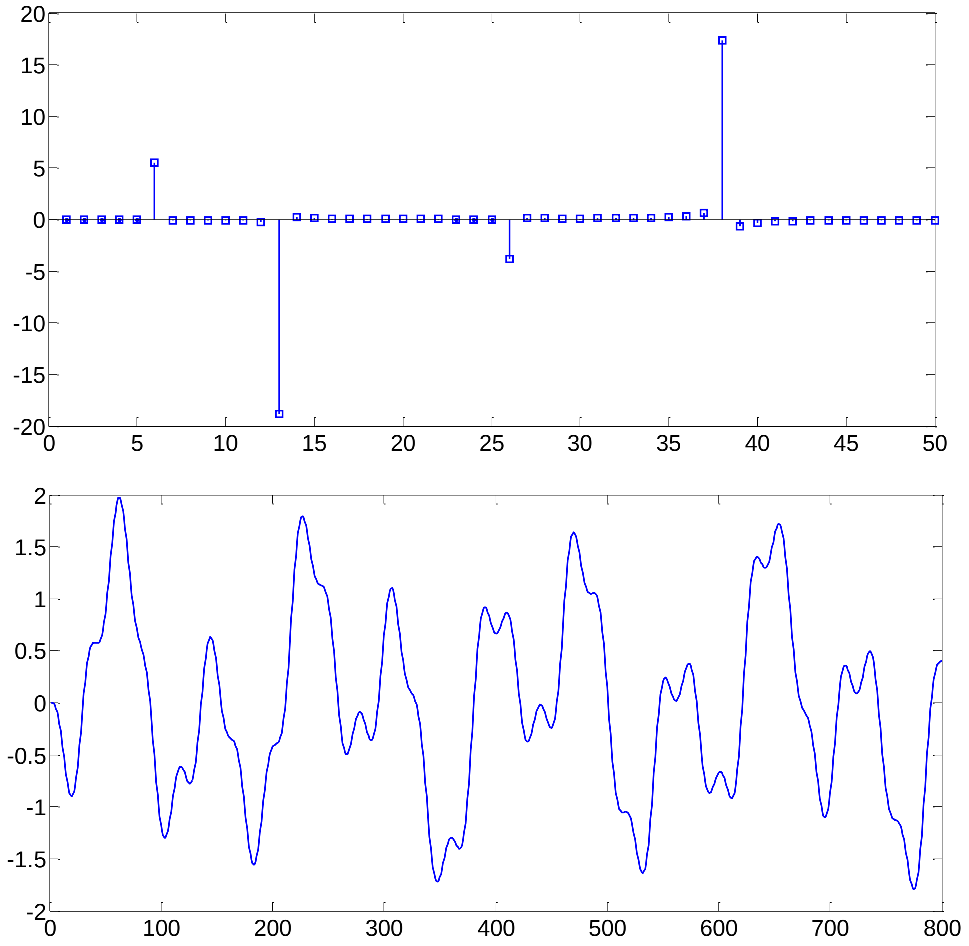

Figure 8. Simple example of a non-intuitive sparse signal. A Fourier 4-sparse vector in the figure on the top and its time domain signal in the bottom figure which would require at least 60 measurements according to Shannon-Nyquist-Whittaker Sampling theorem.

Recovery Algorithms

Recovery of the sparse vector from an underdetermined linear system for compressive sensing has been broadly divided into three categories, namely; optimization methods, hard thresholding-based methods and greedy methods. There are several algorithms in each category used for reconstruction usually based on the type of application in question. For compressive laser ranging the algorithms used were greedy methods as they are easy to implement. The algorithms are extremely fast compared to optimization algorithms and are a better fit to the laser ranging problem than the thresholding algorithms. We will briefly describe optimization and thresholding methods and then we will look at the greedy algorithms used for this project in detail.

Optimization Methods

The optimization method used for compressive sensing is a convex optimization problem and is NP-hard in general; that is, the problems cannot be solved in polynomial time. An optimization problem is given by

$$minimize\ F_0(x)\ subject\ to\ F_i(x)\ \leq\ b_i, i\ \epsilon\ [N]$$

where the function F_0 is the objective function and the functions $F_{1},....,F_n$ are constraint functions. Based on the constraint functions, this problem is typically a convex optimization problem. However, in our case, since the constraints are linear functions, the problem is a linear one and can be approximated by

$$minimize\ ||z||q\ subject\ to\ Az\ =\ y$$

For q = 1, this becomes a convex relaxation problem and is given by

$$minimize\ ||z||_1\ subject\ to\ Az\ =\ y$$

This is the most commonly used optimization method called the l_1-minimization algorithm also known as basis pursuit. In fact, most of the proofs related to successful recovery of sparse vectors in compressive sensing are based on this method.[22]

Hard Thresholding Based Methods

Hard thresholding uses an operator H_s that sets all but the s-largest components greater than γ to zero and is given by,

$$H_s(z) = \begin{cases} z_i & |\gamma| \\ 0 & otherwise \end{cases}$$

The remaining components of the signal are left untouched. Here H_s is the best sparse approximation to the vector z, where $||z - Hs||_1$ is at its minimum. This process is done iteratively and is similar to the greedy algorithms that will be discussed further. The most commonly used thresholding algorithm is the Iterative thresholding algorithm.[23,24]

Greedy Methods

Greedy algorithms solve the reconstruction problem by iteration. The algorithm looks for columns in the measurement matrix in greedy fashion, which correlate with the measurement vector iteratively. Each iteration minimizes the least-square error and subtracts the residual from the measurement vector for the next iteration. The process continues until the exact columns corresponding to the measurement vector are identified. Matching pursuit algorithms are the most widely used greedy algorithms for sparse approximation. For this project the OMP (Orthogonal Matching Pursuit) and CoSaMP

(Compressive Sampling Matching Pursuit) were used which are derived from the Matching Pursuit algorithm developed by Mallat and Zhang in 1993.[25]

Orthogonal Matching Pursuit. Orthogonal Matching Pursuit (OMP) was developed my Mallat and Zhang in 1993 and demonstrated for compressive sensing by Gilbert and Tropp in 2007.[25,26] OMP iteratively constructs an approximation by calculating a locally optimal solution at each step. The locally optimal solutions add up to form the final globally optimal solution that is an approximation of the original signal x. The basic premise is to find columns in A that most closely correspond to the measurement vector y. Since x is sparse, the measurement y is formed from the linear combination of the columns in A that correspond to the non-zero elements in x. The algorithms take the measurement vector y, the measurement matrix A and the sparsity level s as the inputs. The measurement vector y is considered as a residual and is updated every iteration. The residual is updated by subtracting the contribution of the largest column from A at each iteration to give the largest columns in A that correspond to the non-zero elements in the vector x. The algorithm is described in Table 1.

Table 1 OMP Algorithm[26,27]

```
OMP($A, y, s$)

Input: Measurement matrix $A$, measurement vector $y$, sparsity level $s$
Output: $s$-sparse approximation a of the signal

$a^0 \leftarrow 0$
$v \leftarrow y$
$k \leftarrow 0$

$repeat$
$k \leftarrow k + 1$
$z \leftarrow A * v$
$\Omega \leftarrow argmax(z)$
$T \leftarrow \Omega \cup supp(a^{k-1})$
$b|_T \leftarrow A_T^{\dagger} y$

$a^k \leftarrow b_s$
$v \leftarrow u - Aa^k$
```

The algorithm begins by initializing the approximation a to 0, the residual v to the measurement vector y and the stopping condition k to 0. The algorithm then goes into a loop to iteratively approximate the signal until the stopping condition is met, which for this project was the sparsity s of the signal. An iteration starts off by taking the inner product between the measurement matrix A and the residual v, and finding the largest correlation between the two, Ω. The new value is combined with the vector containing the values from earlier iterations in the support vector T. A least-square approximation of the signal is then calculated using the columns with indices corresponding to the support vector T and is stored in b which is then assigned to the indices of the non-zero values in the signal approximation a. Finally, the residual from the current iteration is subtracted from the total residual before the iteration is run again.

The recovered approximation to the signal x is based on its sparsity and the number of measurements taken. As the sparsity increases, so do the amount of measurements required for an accurate reconstruction, a characteristic seen in every recovery algorithm. The number of measurements required for accurate reconstruction is given by

$$m = \; slog(n),$$

where s is the sparsity and n is the length of the unknown signal.

Compressive Sampling Matching Pursuit. Compressive Sampling Matching Pursuit (CoSaMP) is based on the preceding algorithm OMP and was developed by Needell and Tropp in 2009.[27] Other variations of OMP include algorithms like Regularized Orthogonal Matching Pursuit (ROMP),[28] and Stagewise Orthogonal Matching Pursuit (StOMP).[29] CoSaMP follows steps similar to OMP, except that instead of finding a single column with the largest contribution it finds a vector of columns with the largest contribution and then eliminates all but the largest column to build up the approximation, then subtracts the contribution from the residual. CoSaMP provides better performance than OMP in the presence of noise and works with a variety of sampling schemes. The algorithm for CoSaMP is given in Table 2.

Table 2 CoSaMP Algorithm[27]

```
CoSaMP($A, y, s$)

Input: Measurement matrix $A$, measurement vector $y$, sparsity level $s$
Output: $s$-sparse approximation $x$ of the signal

$a^0 \leftarrow 0$
$v \leftarrow y$
$k \leftarrow 0$
$repeat$
$k \leftarrow k + 1$
$z \leftarrow A' * v$
$\Omega \leftarrow supp(z_{2s})$
$T \leftarrow \Omega \cup supp(a^{k-1})$
$b|_T \leftarrow A_T^{\dagger} y$
$b|_{T^c} \leftarrow 0$
$a^k \leftarrow b_s$
$v \leftarrow u - Aa^k$
```

The algorithm for CoSaMP is quite similar to the OMP algorithm. The algorithm starts off by initializing the values of the signal approximation, the residual and the stopping condition. The inner product of the measurement matrix and the residual is calculated for each iteration. The next step differs from OMP, where instead of searching for the largest correlation between the measurement matrix and the residual, the algorithm searches for $2s$ largest components in the inner product. The largest components are merged with the approximation from the previous iteration and a least squares approximation of the signals is calculated and the most relevant k values are saved in the approximation. The signal approximation is updated and the weight of these values is subtracted from the residual before going to the next iteration.

CoSaMP provides better bounds on its results because it searches for $2s$ components instead of one largest component, improving the bounds on Restricted Isometry Constant (RIC, seen in the next section) and making it more robust than OMP.

CoSaMP in general provides better guarantees than OMP in the presence of noisy samples. CoSaMP was the primary algorithm used for the compressive laser ranging experiment since it provided better guarantees for reconstruction. CoSaMP was also successfully implemented on a soft core processor and the details are found in Chapter 5.

Conditions for Signal Recovery

Reconstruction of a signal with fewer measurements than required by Nyquist sampling comes with certain conditions. The biggest condition was discussed earlier in the section explaining sparsity. For accurate reconstruction of the signals some properties related to the sparsity and the measurement matrix need to be considered. The two most important properties that make the process more reliable and easier are the restricted isometry property and the incoherence of the measurement matrix.

Restricted Isometry Property

An important consideration for robust compressive sampling is the ability of the measurement matrix to preserve the important pieces of information from the signal of interest in the inner product. This property is verified by applying something known as the Restricted Isometry Property (RIP) of the measurement matrix. RIP is defined in terms of the Restricted Isometry Constant (RIC) for every sparse vector x, denoted by $\boldsymbol{\delta}$ such that,

$$(1-\delta)||x||_2^2 \leq ||\boldsymbol{A}x||_2^2 \leq (1+\delta)||x||_2^2$$

Literally speaking, this inequality is NP-hard in nature and cannot be computed easily and the RIC is unbounded. However, for certain random matrices it has been shown that the RIC can remain bounded.[16, 17] RIC is generally in the range of 0 to 1 for random matrices. The closer the RIC is to 0 the better the property of the matrix to hold the information about the sparse vector x in the inner product. RIP ensures that all subsets of s columns from the measurement matrix are orthogonal and that the sparse signal is not in the null space of the matrix i.e. it does not have a trivial solution where the inner product Ax is 0.[32,33]

Coherence

When fewer measurements than required by the Nyquist sampling theorem are being taken, it is worthwhile to put a bound on the minimum number of measurements required to accurately reconstruct the desired signal. The number of measurements required for robust sampling and the ability to reconstruct from those measurements depends on the coherence of the measurement matrix. Coherence is a property that measures the maximum correlation between the elements of two matrices. The two matrices can be the measurement matrix and the matrix representing the basis in which the signal is sparse. It is important for the two matrices to have low coherence for compressive sampling to work. If the measurement matrix is given by φ and the representation basis matrix is given by ∂ then the coherence is given by,

$$\mu\,(\varphi, \partial) = \sqrt{\mathrm{n}} \max_{1 \le k, j \le n} |\langle \varphi_k, \partial_j \rangle|$$

Here we take the inner product of the vectors represented by φ_k and ∂_j to find the correlation between the two matrices. The representation bases are common signal representations such as the pixel basis, Fourier basis, wavelet basis, discrete cosine basis etc. It is worthwhile to have measurement matrices that can have low coherence with any representation basis. This property is fulfilled by random matrices such as Gaussian, Bernoulli and pseudorandom binary matrices. [34,35] In the next chapter, we will use particular convolutions of pseudorandom binary matrices for compressive laser ranging as they are incoherent with any basis and provide the best results for compressive sampling in an unknown basis,[36] which in our case will be the range domain.

COMPRESSIVE LASER RANGING

Laser Ranging and its Drawbacks

Compressive laser ranging (CLR) by itself is a novel method of doing laser ranging that has not been tried before. Although a clear application for CLR is still forthcoming, the method does offer some advantages compared to conventional laser ranging paradigms such as chirped and pulsed laser ranging. CLR offers an advantage over chirped ranging by mitigating the need for high bandwidth analog waveform generation and storage. The range resolution of chirped lidar is based on the bandwidth of the chirp and is given by $c/2B$ where B is the bandwidth of the chirp and c is the speed of light. CLR offers the same resolution using binary waveforms, which are easy to generate using a single seed and don’t need storage. For pulsed ranging, to obtain sub-centimeter resolution, the bandwidth of the pulse needs to be in the GHz range, which means that the detectors and analog-to-digital converters need to respond with GHz bandwidth as well, corresponding to nanosecond time scales. In contrast, CLR uses detectors with bandwidths in the MHz range to obtain sub-centimeter resolution, since the measurements are not made in the time domain, but rather as an inner product that we will explore in detail in the next few sections. Pulsed ranging also has high peak powers that can be several megawatts and may be detrimental depending on the target being ranged. In CLR the power is distributed over random binary patterns with peak powers in the milliwatt range.

The primary advantage of CLR over conventional ranging is that CLR pushes the required computational power from the detector to the processing system and reduces the amount of data to be transmitted from the sensor to the processing system. This setup can be especially useful in harsh environments, such as in space or on deep sea probes. The compressed measurement of data on a low-bandwidth detector saves energy. The sensor system can then be made more compact and robust without the need for digital compression or high bandwidth transmission links.

Compressive Ranging Approach

The innovation in Compressive Laser Ranging (CLR) is the recasting of time-of-flight measurements from a convolution measurement to an inner product measurement, allowing compressive sensing techniques to be directly applied to inner product measurements in the generally sparse time-of-flight basis.

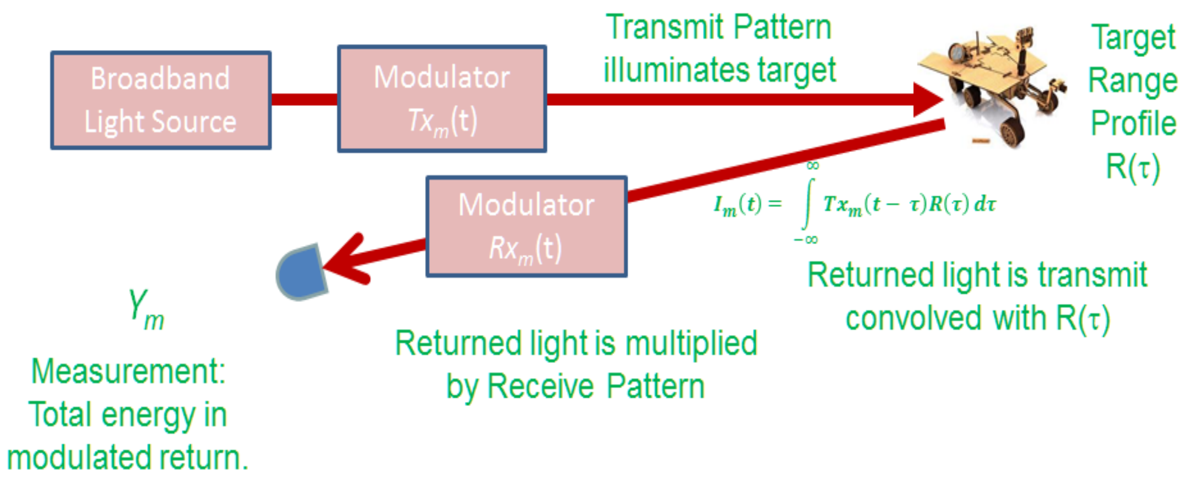
Broadband Light Source
Modulator $Tx_m(t)$
Transmit Pattern illuminates target
Target Range Profile R(τ)
Modulator $Rx_m(t)$
$I_m(t) = \int_{-\infty}^{\infty} Tx_m(t-\tau)R(\tau)\,d\tau$
Returned light is transmit convolved with R(τ)
Y_m
Measurement: Total energy in modulated return.
Returned light is multiplied by Receive Pattern

Figure 9. Compressive laser ranging paradigm

In the approach presented in Figure 9 the signal is a range profile ***R(τ),*** which represents the reflectivity of a 1D scene of targets as a function of round-trip delay τ. The transmitted signal and the received signal are temporally modulated and collected using a an integrating detector, which allows for ***R(τ)*** to be determined without the need for a time-domain signal. The transmitted signal is optically modulated by a signal *$Tx_m(t)$* and the return signal is optically modulated by a signal *$Rx_m(t)$* before being measured by a detector that only measures the total power. This process is repeated with different Tx and Rx modulations to produce *M* measurements (*m* = 1,…, *M*). To directly apply CS basis pursuit algorithms to reconstruct the range profile, we have to represent these measurements with a sampling matrix. The description and physical explanation for this sampling matrix is given below. We begin with a temporally modulated transmitted signal *$Tx_m(t)$.* This transmitted signal reflects off the targets with range profile ***R(τ),*** forming a convolution product given as

$$I_m(t) = \int_{-\infty}^{\infty} Tx_m(t - \tau)R(\tau)\,d\tau.$$

$I_m(t)$ represents the return signal before the receive modulator, which is then temporally modulated by $Rx_m(t)$. After modulation the signal is then temporally integrated with a detector. Each of the integrated measurements in the measurement vector ***Y*** is given by:

$$Y_m = \int_{-\infty}^{\infty} Rx_m(t)I_m(t)\,dt$$

Substituting the value for I_m and rearranging the order of integration gives an inner product measurement in the delay domain,

$$Y_m = \int_{-\infty}^{\infty} R(\tau) D_m(\tau)\, dt,$$

where $D_m(\tau)$ is a function representing the correlation product of the transmit and receive modulations,

$$D_m(\tau) = \int_{-\infty}^{\infty} Tx_m(t) Rx_m(t+\tau)\, dt$$

If we discretize the measurement problem, the CLR sensing function and the range profile can be represented as a mixing matrix *D* of dimensions *M* x *N* as $D_{mn} = D_m(n\delta\tau + \tau_H)$ and a sparse signal vector $R_n = R(n\delta\tau + \tau_R)$ where τ_R is the average round trip delay to the targets and τ_H is the delay required to accurately overlap the Rxm(t) signal with the return signal since inherent delays are caused by fiber and electronics in the system. BTR is the bandwidth of the signals Txm(t) and Rxm(t), which defines $\delta\tau = \frac{1}{B_{TR}}$. The defined problem therefore is to solve for $\boldsymbol{R}$ in $\boldsymbol{Y} = \boldsymbol{DR}$. CS recovery algorithms such as CoSaMP and OMP can then be used to reconstruct the range profile. The CLR range resolution is given by *dR = c/(2BTR)* where *c* is the speed of light. The bandwidth of the detector and analog-to-digital converters required in this case is then *BTR/N* which can be orders of magnitude lower than that required for high-resolution, incoherent ranging systems given by *c/(2dR)*. The range window for CLR is approximately the sum of the widths of $Tx_m(t)$ and $Rx_m(t)$ given by $N\delta\tau$ and is weighted by their correlation with a full width at half maximum (FWHM) value of about $N\delta\tau/2$.

Experimental Setup

The compressive laser ranging approach was experimentally demonstrated using the setup depicted in Figure 10. Successful results were obtained with setups that had two different detectors. The initial setup included a Geiger-mode avalanche photodiode was used to verify the theory and obtain results for ranging multiple targets; a second setup with a new detector was used to make the system more compact using a linear-mode avalanche photodiode. Both setups will be discussed in detail in this section.

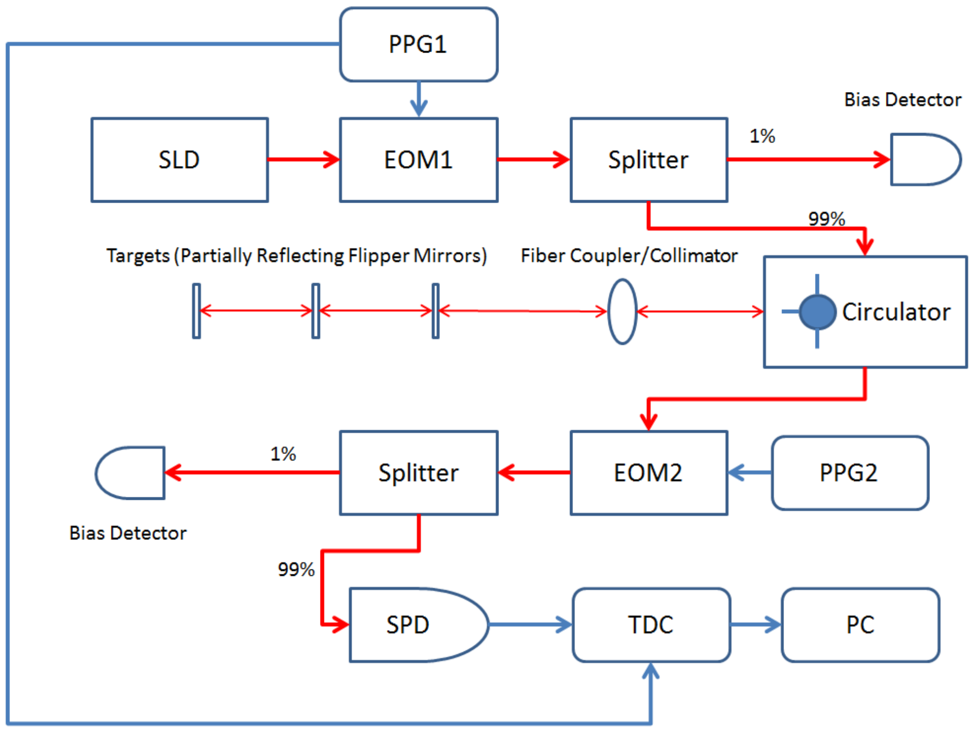

Figure 10. Compressive laser ranging setup with pulse pattern generators

The first experimental results were demonstrated using a fiber-coupled GM-APD (Geiger Mode Avalanche Photodiode Detector) and an incoherent illumination source, which is a 1550nm SLD (super luminescent diode) with a FWHM bandwidth of about 50 nm. The GM-APD used for this experiment was a highly sensitive single-photon detector with one of the best efficiencies, along with a low dark count rate for 1550nm light. It had some drawbacks, however: the design was bulky because of a large heat sink that was required to cool the detector to -50° C. Also, the APDs had a 1μs dead time after a detection event, which potentially could have caused irregularities in the collected data. The detector was used at an efficiency of 10% and a dark count rate of ~1 kHz, but the efficiency could be increased to 20% for a corresponding increase in dark count rate if required. The super luminescent diode (SLD) was used as a broadband laser to reduce the errors in the measurement caused by speckle in previous versions of the experiment, which prevented the resolution of multiple targets.[14]

Electro-optic modulators (EOMs) are frequently used in fiber-optic telecom systems to modulate data bits onto laser light. The same idea was used here, where Mach-Zehnder Electro-optic Modulators were used to modulate the intensity of light in the form of bit patterns generated by pulse pattern generators (PPGs). The EOMs were rated for 10 Gbps data rates and had to be temperature controlled to minimize errors in the bits. PPGs are instruments from the telecom industry that are used for signal integrity testing. In this case they were used to generate pseudorandom binary bit patterns that formed the measurement matrix of the compressive laser ranging setup. The measurement process is discussed below.

The light from the diode is modulated by a null biased electro-optic intensity modulator, EOM1, for which the transmit patterns $\boldsymbol{Tx_m(t)}$ are generated by the pulse pattern generator PPG1. The 1% output from a 99:1 splitter after EOM1 is used to monitor and adjust the null bias of EOM1. The 99% output of the splitter is sent through a circulator to a fiber collimator/coupler, which creates a free-space beam. The light hits targets, which are partially reflecting flipper mirrors, and returns to the circulator via the fiber coupler/collimator. The output from the circulator is passed to a second null-biased electro-optic intensity modulator EOM2 for which the return patterns $\boldsymbol{Rx_m(t)}$ are generated by the second pulse pattern generator PPG2. The light from EOM2 is again passed through the 99% end of a 99:1 splitter where the light from EOM2 is massively attenuated (80 dB) and collected by the GM-APD. The detector output is then sampled by the Time-to-Digital Converter (TDC) for time digitization. The 1% is again used to monitor and adjust the null bias of EOM2. The TDC uses a trigger from PPG1 to synchronize the received photon counts with the beginning of the waveforms transmitted to EOM1 and EOM2. The digitized data are sent to a computer for further processing. Photon counts from multiple waveform pairs being sent to EOM1 and EOM2 are processed to obtain the measurements $\boldsymbol{Y}$. A section of the sample data obtained from the APDs is shown in Figure 11.

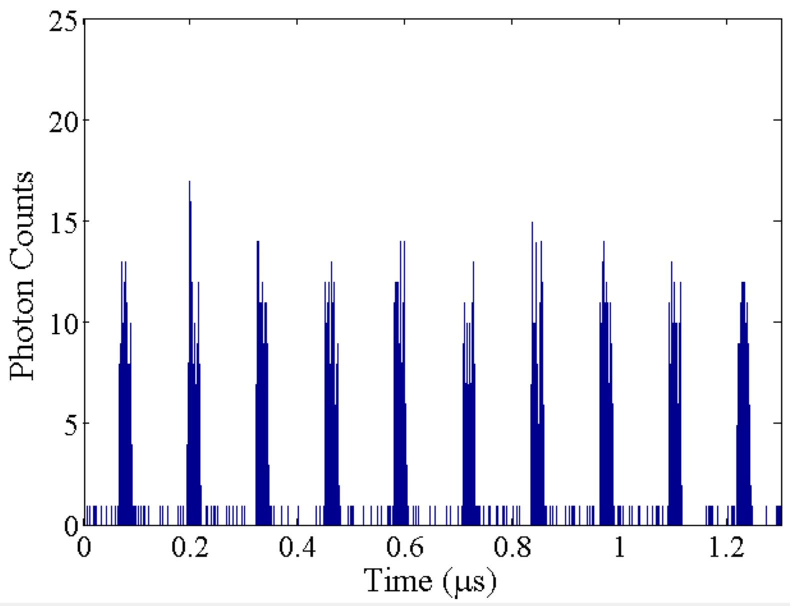

Figure 11. Time vs photon counts data obtained from the APDs.

In Figure 11 each peak corresponds to an overlap between the $\boldsymbol{Tx_m(t)}$ and $\boldsymbol{Rx_m(t)}$ waveforms after hitting the target. The patterns are separated by time which for simplicity was two times the length of each pattern. This separation ensured that measurements did not interfere with each other. The number of counts depends on the delay $\boldsymbol{\tau}$ and forms the measurement vector $\boldsymbol{Y}$ after some processing**.** For 30 measurements, 60 waveforms are required to be transmitted to normalize the correlation between the two patterns that overlap and to correct for any errors that occur as the light travels through the system. Two binary-bit patterns would produce a correlation with a nominally Gaussian shape. In reality, the correlation of the two patterns needs to be centered about zero for the ensemble of the measurements to form a zero-mean Gaussian measurement vector. To that effect, transmit waveforms $\boldsymbol{Tx_m(t)}$ were generated in sets of 2 with 30 sets, each containing the same waveform twice. The receive waveforms $\boldsymbol{Rx_m(t)}$ were also transmitted in 30 sets of 2; however, the second waveform in each set was the bit complement of the first waveform in that set. An illustration of the same is shown in Figure 12.

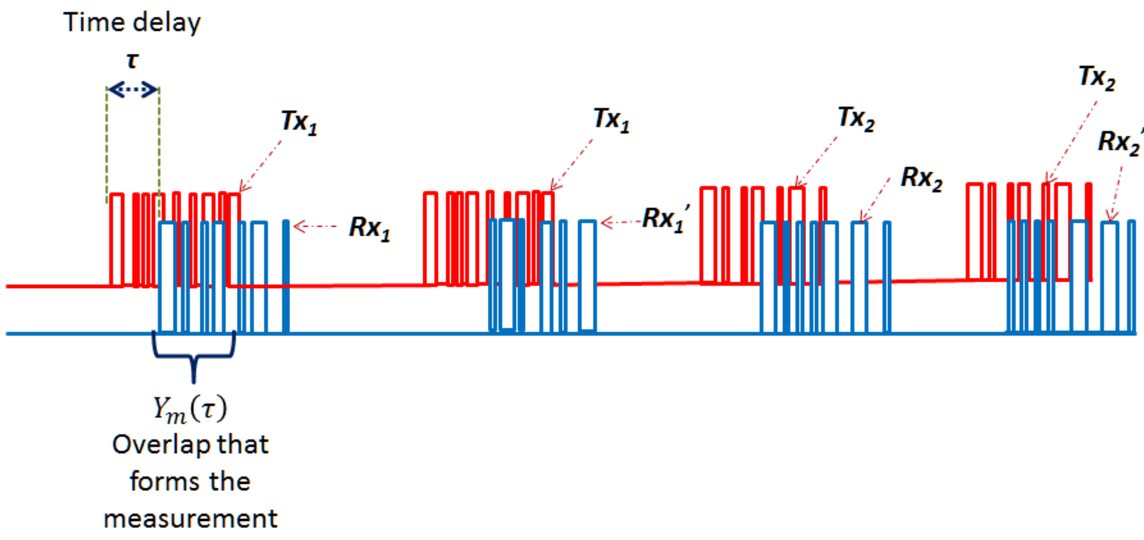

Figure 12. Timing diagram of the system that forms the measurement vector ***Y***.

The resulting initial measurement vector consisted of 60 measurements with each odd measurement representing the correlation of the transmit and receive waveforms and the next measurement containing its complement. For the final measurement vector ***Y*,** the complement measurement was subtracted from the initial correlation measurement, thus normalizing and background subtracting the measurement vector and reducing errors in the system.

The second setup consisted of a similar system with a room-temperature, linear APD that operated at 1550nm. This experiment was performed in an effort to make the system compact since the new APD was the size of a Quarter and required much less power and space, and counteracted the dead-time issue related to the GM-APD. The APD put out a voltage as a function of photons received. This detector was not fiber coupled, so a 50 mm focal-length lens was used with a fiber coupler to collect onto the detector the light from the fiber connected to the second EOM. This detector had a built-in amplifier and had to be supplied with a 5V power supply along with a 50V DC power supply for the bias control. Compared to the GM-APD, however, this detector required a relatively high signal-to-noise ratio. Because the voltage output of the APD was collected using an Analog-to-Digital (ADC) card connected to the computer. This APD was able to achieve results similar to the GM-APD in experiments using mirrors, in which all the targets were successfully recovered. A further experiment was attempted where a Spectralon target instead of the mirrors was used to test results with less-cooperative ranging targets. Initial measurements with the linear APD failed to provide sufficient back-scattered photons (more in the next section) to exceed the detector NEP. An erbium doped fiber amplifier (EDFA) was used next to increase the transmitter power from ~15mW to ~960 mW. The setup with the EDFA initially failed because of a miscalculation of the delay it introduced into the system. However, by the time this issue was fixed, the linear-mode APD ceased functioning, possibly from a voltage surge on one of its bias voltage inputs. Based on the results from the mirrors and photon budget calculations it can be inferred that the setup with non-cooperative targets would have been successful had the linear mode APD not died.

Photon Budget and SNR Considerations

Single photon counting typically follows a Poisson distribution for which the signal-to-noise ratio (SNR) is given by

$$SNR = \frac{N}{\sqrt{N}},$$

where *N* is the number of counts. Since photon counting is inherently a shot-noise limited mode of operation, the dominant traceable source of noise was shot noise. Given that the average number of counts observed for the resolution of the targets was about 60,000 over a period of 100ms with a dark count rate of 1 kHz. The SNR of the GM-APD setup was thus calculated to be about 245 where the laser output power was 15mW. The power transmission coefficient of the system was calculated from the insertion losses incurred at the EOMs, fiber splitters, fiber circulators, fiber couplers along with the power loss due to coupling inefficiency from free space. The calculations were done for a target average distance of 100 cm away from the transmitter. The power transfer coefficient from the laser to free space was calculated to be 0.0485. The coefficient from coupling back into a fiber to the GM-APD was calculated to be 0.1217. With three partially reflecting mirrors (5% reflectivity) the power to the GM-APDs can be calculated to be 13 μW. The power was further attenuated using 25 dB attenuators to avoid saturating the GM-APD, which also acted as a simulation for performance of the system in low-light conditions. With the GM-APDs, it was found that the system provided sufficient SNR to resolve 3 targets successfully.

The linear-mode APD proved to be more challenging for SNR calculations. The same transfer coefficients calculated for the GM-APD setup still apply in this setup, except for the extra loss incurred during the ranging of a diffuse (Spectralon) target 100 cm away from the transmitter. The linear-mode APD had a specified noise-equivalent-power (NEP) of 88 fW/$\sqrt{Hz}$ with a bandwidth of 50 MHz. The power loss due to free space coupling on the detector was difficult to calculate; however, it was approximated to be about -10 dB. With these values and the power transfer coefficients calculated in the earlier section the optical power on the detector was calculated to be approximately 7 μW, which would generate a SNR of 2135. However, this SNR would be significantly reduced by the additional noise that would be contributed by the high-bandwidth analog amplifiers on the APD and the electronics required to collect the data. The amplifier on the APD provided a gain of 340 KV/W, which provided an output voltage of 450 mV. The signal was observed on an oscilloscope and the noise level from electronics was observed to be about 1.5 mV. From these observations, the SNR was approximated to be ~300 for mirror targets using the linear mode APD.

For the diffuse target the power collected was significantly reduced by scattering of light from the target into angles other than the backscatter direction. Using Lambert's cosine law and an approximation of the size of the input aperture it was calculated that 0.16% of the incident power was returned to the detector. The reflectance of the Spectralon target was specified to be 0.99. Using the losses for the mirror targets and the losses for diffuse target scattering, the power incident on the detector was calculated to be 4.488 nW, which gives an SNR of 7.2. This SNR was not sufficient to successfully range targets since with a gain of 340 KV/W the output voltage was observed at 1.5 mV, which was giving an actual SNR of less than 1. To improve the SNR an EDFA was added right before transmitting the light to the targets. The EDFA would have improved the SNR by ~20 dB, putting it in the realm of the SNR for the mirror targets. Had the APD persisted, this SNR would have been sufficient to resolve the Spectralon target using the linear-mode APD.

Results and Discussion

Pattern lengths of 128 bits were chosen for the setup with both 1.5 cm and 3 cm resolution, corresponding to pattern bit rates of 10 Gbps and 5 Gbps, respectively. The number of measurements, M, was 30, which is much less than $N = 255$ for 128-bit pseudorandom pattern. It is however greater than $Slog(2N)$, which is ~19 for S = 3, the number of targets. The data collected from the time-to-digital converter (TDC) is processed so that the number of photon counts corresponding to each of the 30 patterns yields the measurement vector Y to be used with CS recovery algorithms. For the data below the total number of photon counts for all 30 measurements was approximately 60,000. A typical measurement matrix $\boldsymbol{Y}$ is plotted in Figure 13.

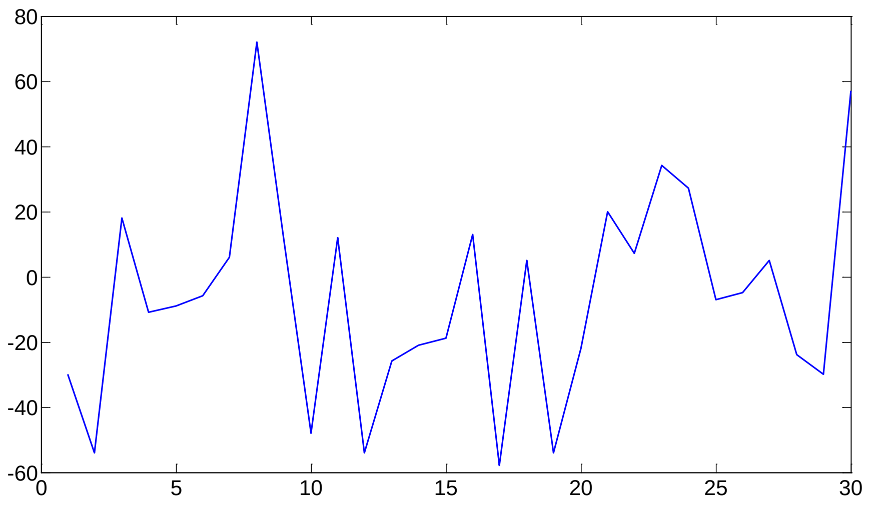

Figure 13. Plot of measured vector *Y* with 30 measurements.

The measurements for the scene were processed with a CoSaMP algorithm, which yielded the results for 3 cm and 1.5 cm resolution. The range profiles for waveforms at 10 Gbps, which yield a resolution of 1.5 cm are shown in Figure 14. To simulate a changing scene two targets were ranged with one being stationary as a reference range source. For the higher resolution system the number of measurements was increased from 30 to 60, which is still much smaller than the scene of length 255. Compressive laser ranging is able to achieve high resolution range profiles with multiple targets without the need for high-bandwidth detectors or ADCs. The GM-APD has high timing resolution of ~200 ps, which means the 10 Gbps ranging presented here represents a factor of two increase in range resolution, however this resolution could be retained for much slower detectors. Due to the availability of relatively cheap and compact electronics (discussed in the next chapter), which can produce multi- gigabit pseudorandom binary waveforms, it is possible to create a compact and efficient high resolution ranging system. By increasing the data rate and bit length it is possible to create a higher resolution and a larger window for ranging.

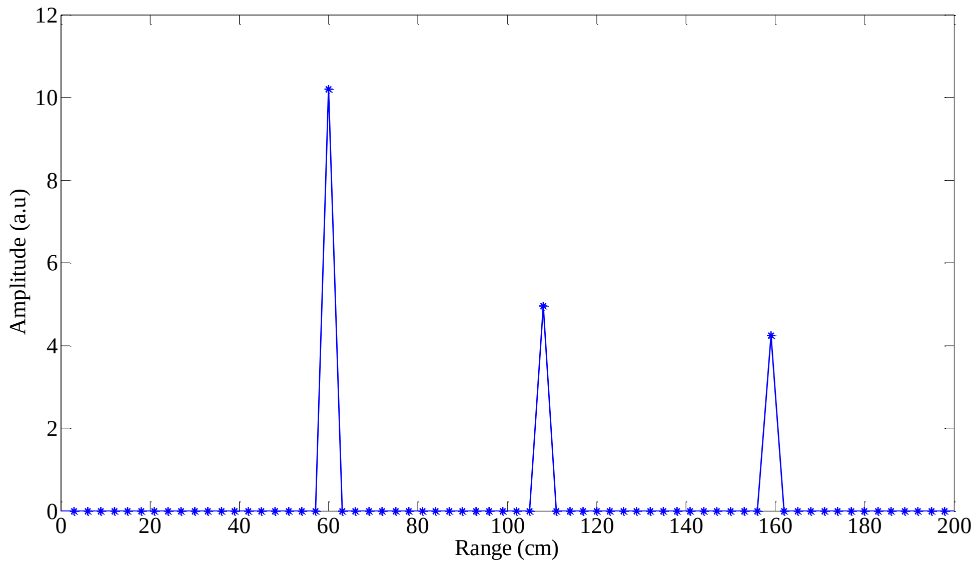

Figure 14. Compressive laser ranging with multiple targets. In this case 3 targets are found with a distance of 48cm between each other.

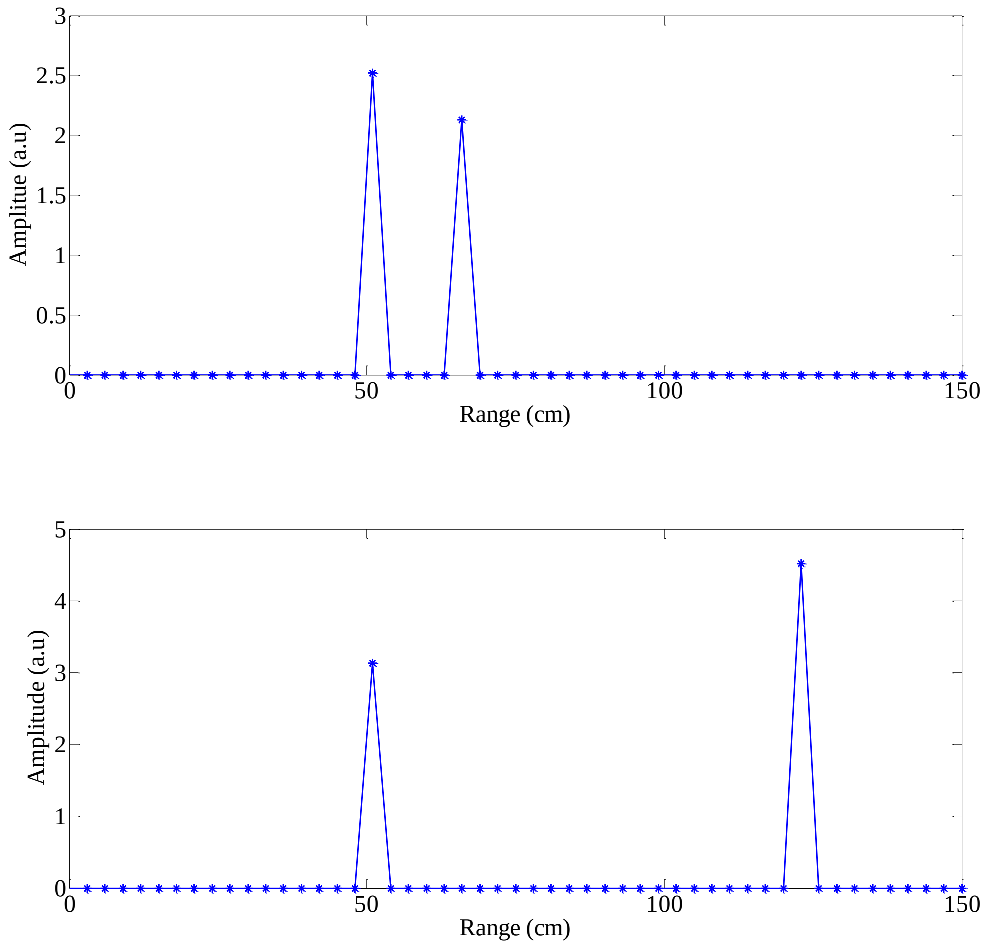

Figure 15. Compressive laser ranging for a single reference target and a moving target

IMPLEMENTATION OF SIGNAL GENERATION ON FPGA

Motivation

Signal generation for the CLR system done using large pulse pattern generators is a big detriment to making the CLR system compact and portable. For generation of transmit and receive waveforms, two PPGs had to be used and each weighed about 40 kilograms with a volume of about 0.2 m^3. To make the system as compact as possible, a solution was needed where the signal generation could be implemented in a small package. An excellent option is Field Programmable Gate Arrays (FPGAs), which are small chips that have the capability to produce multi-gigabit waveforms on several different channels, making it possible to generate the transmit and receive waveforms on one circuit board weighing less than a kilogram and had a volume of less than 0.002 m^3.

Introduction to FPGA

A Field Programmable Gate Array (FPGA) is an array of logic gates that can be programmed to connect the gates in a manner that can emulate the behavior of most electronic circuits. An FPGA can be programmed repeatedly in the field, hence the name “field programmable,” which is what makes them popular. FPGAs are arranged as a series of configurable programmable logic blocks and configurable interconnects between those blocks, as depicted in Figure 16. The two major companies in the field of FPGAs are Altera and Xilinx. Hardware description languages such as VHDL and Verilog are used to program these devices.

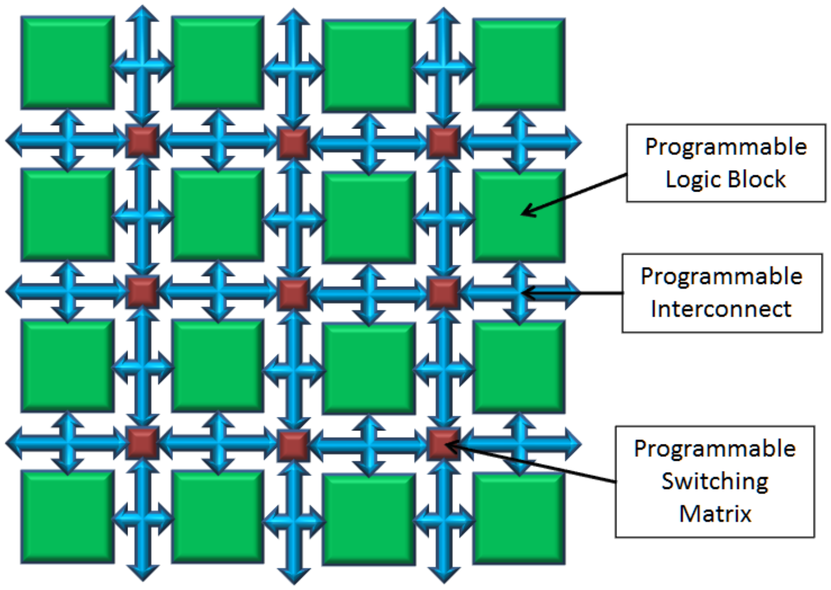

Figure 16. Architecture of an FPGA[37]

Depending on the manufacturer the logic blocks of the FPGA have varying complexity and are implemented in a different fashion. A typical Xilinx FPGA logic block has look up tables (LUTs) that perform all the logic functions. To perform these logic functions the logic blocks contain multiplexers, flip-flops, registers etc. A typical Xilinx logic block is shown in Figure 17.

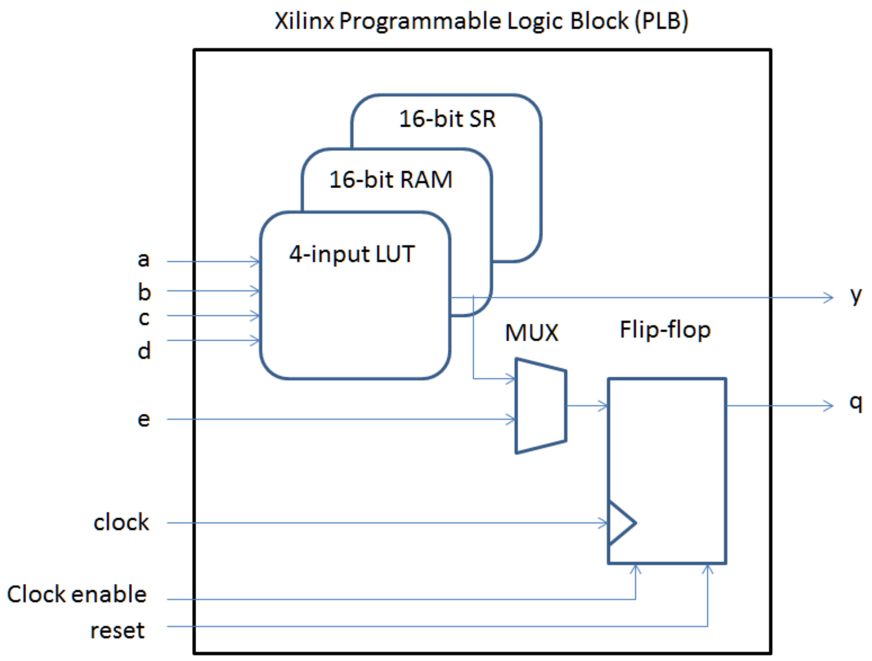

Figure 17. Typical design of a programmable logic block manufactured by Xilinx inc.[37]

Owing to the programmability of these logic blocks in hardware using languages such as VHDL and Verilog, large parallel processing architectures can be implemented. FPGAs are mostly being used for applications which involve high-speed digital signal processing (DSP), embedded microcontroller systems and physical layer communication chips. For this project the high-speed DSP and physical layer communication chips option was used to implement serial transceiver links using multi-gigabit transceivers at speeds of 10 Gb/s. A soft core processor implemented using the FPGA was used to implement the recovery algorithm discussed earlier.

Multi-gigabit Transceivers

Multi-Gigabit Transceivers (MGTs) are used for data transfer over high-speed communication links both on and off the chip. The name transceiver represents the functionality of the device since it acts as both a transmitter and a receiver. With the current technology speeds of over 28 Gb/s can be achieved over a single link. For this project, to modulate the light using an EOM transceivers connected to SMA ports with functionality up to 12.5 Gb/s were used. Use of the FPGA along with the MGTs makes the system extremely compact and modular. The schematic of a typical transceiver is shown in Figure 18.

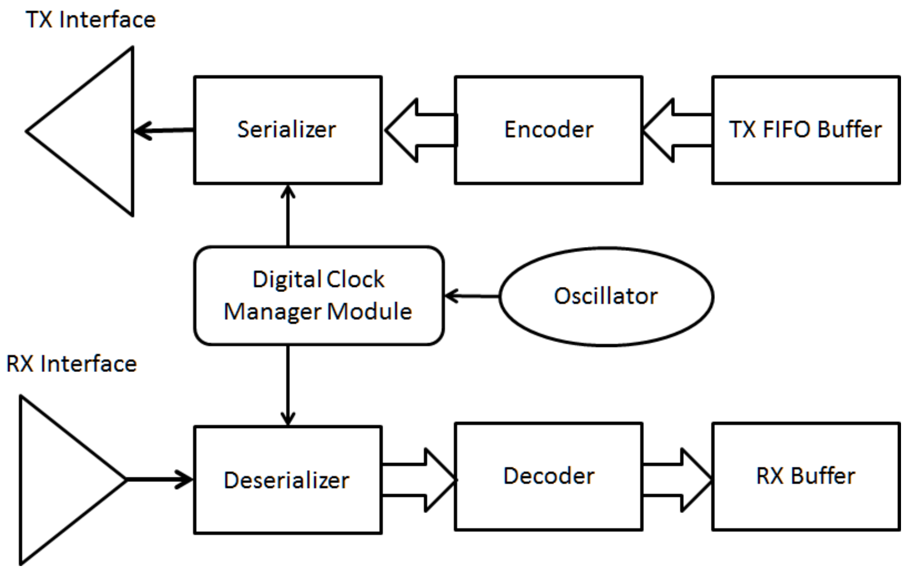

Figure 18. Schematic of a transceiver[38]

The important components of a transceiver that make multi-gigabit rates possible are the clock, clock manager and the serializer/deserializer. The oscillator creates the clock and the clock manager performs clock division using a phase-locked loop (PLL), which signal is sent to the serializer/deserializer. The top part of Figure 18 is the transmitter part of the transceiver where the transmitter first-in first-out buffer (FIFO) holds the parallel data received from the FPGA. Parallel data can be encoded as desired before being sent to the serializer, which uses the clock from the clock manager to generate serial data at rates that are decided by the clock division and the parallel data being sent to it. The serial data is connected to a transmitter line interface, which connects it to the outside world. The interface used for CLR was an SMA port, since the EOMs take in SMA input. However, there are several different options that are more popular, such as the Ethernet port, SFP/SFP+ modules, etc., which are used for network and optical communications. The receiver works in a similar fashion when high speed data are received and deserialized using the clock manager before being decoded. The decoded data are then verified for alignment using clock correction and channel bonding modules before being stored in the receiver buffer from which the FPGA can collect the data.

MGTs much like the design of the FPGA itself are implemented differently by the various manufacturers. The FPGA manufacturers provide users with Intellectual Property (IP) which are readymade pieces of code for operating these transceivers without having to code the process seen in Figure 18 in hardware. These IPs provide several different options of operating the transceivers, along with readymade encoders and decoders for schemes used commonly for high-speed serial data. The data rates at which the transceivers operate are dictated by the clock and the clock manager. Most devices are designed to operate at clock rates of 10-600 MHz. MGTs were used to emulate the behavior of the PPGs seen in the experimental setup. MGTs have made many communication systems more compact, cheaper, and less power consumptive, much like they made the CLR system capable of fitting in a small box instead of several optical tables.

State Machine for PRBS Signal Generation

MGTs and the IP provided by Xilinx were used to implement a state machine to generate known patterns on the transceiver links at rates ranging from 2.5 Gb/s to 10 Gb/s. Depending on the speed grade, Xilinx transceivers are represented by GTX, GTH and GTZ, which represent speeds of 10 Gb/s, 13.1 Gb/s and 28 Gb/s respectively. The Kintex – 7 board used for this project had 32 GTX transceivers, one of which was connected to SMA ports. Four other transceivers were sent to the FMC extension card that can be connected to SMA ports. The GTX transceiver can take inputs of 20, 32, 40, 64 and 80 bits. Since the patterns used in the design were 128 bits, an input parameter of 64 bits was chosen for simplicity as depicted in Figure 19. The patterns had to be zero padded for discrete measurements. For easy visualization the waveform of the patterns was stored in a data file in hexadecimal format and the FPGA was programmed to read the data file and store the waveform in a block RAM that was connected to the transceiver using a pattern generator module. A GTX transceiver module and a pattern generator module are the major components that were coded using VHDL. In Figure 19 it can be seen that the transceiver provided a clock to the state machine implemented by the file CLR_PATTERN_GEN.vhd to return the relevant data from the BRAM to the transceivers.

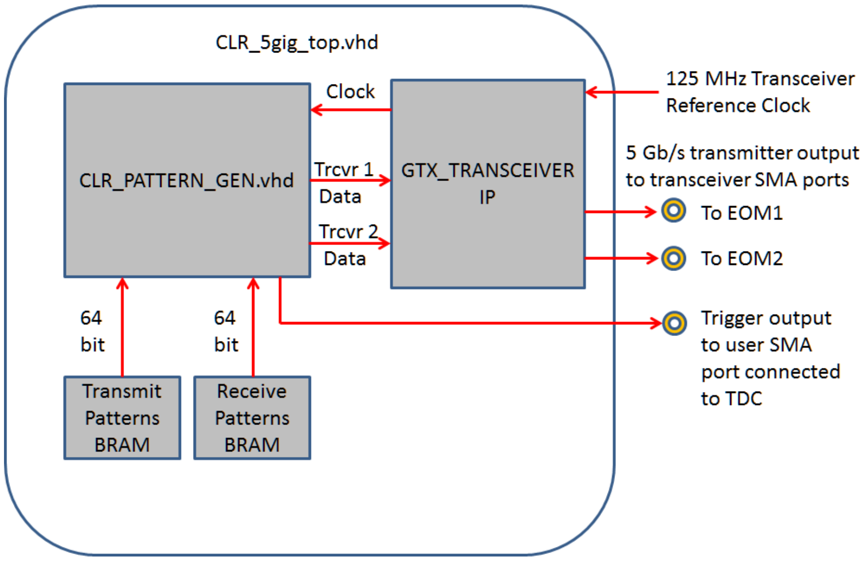

Figure 19. Block diagram of the setup in the FPGA

A low-jitter 125 MHz clock source native to the transceivers on the KC705 board was used by two transceivers to generate waveforms at two SMA ports for the two EOMs. A single LVCMOS pulse was generated at a separate SMA port every time the transceiver received data from the first address in the block RAM. This pulse could be directly detected by the TDC and was used to synchronize the data collection and processing. A picture of the trigger and the first few patterns of the two waveforms generated by the FPGA captured using a 12.5 GHz scope is shown in Figure 20. Certain delays are present between the waveforms to compensate for delays through the system.

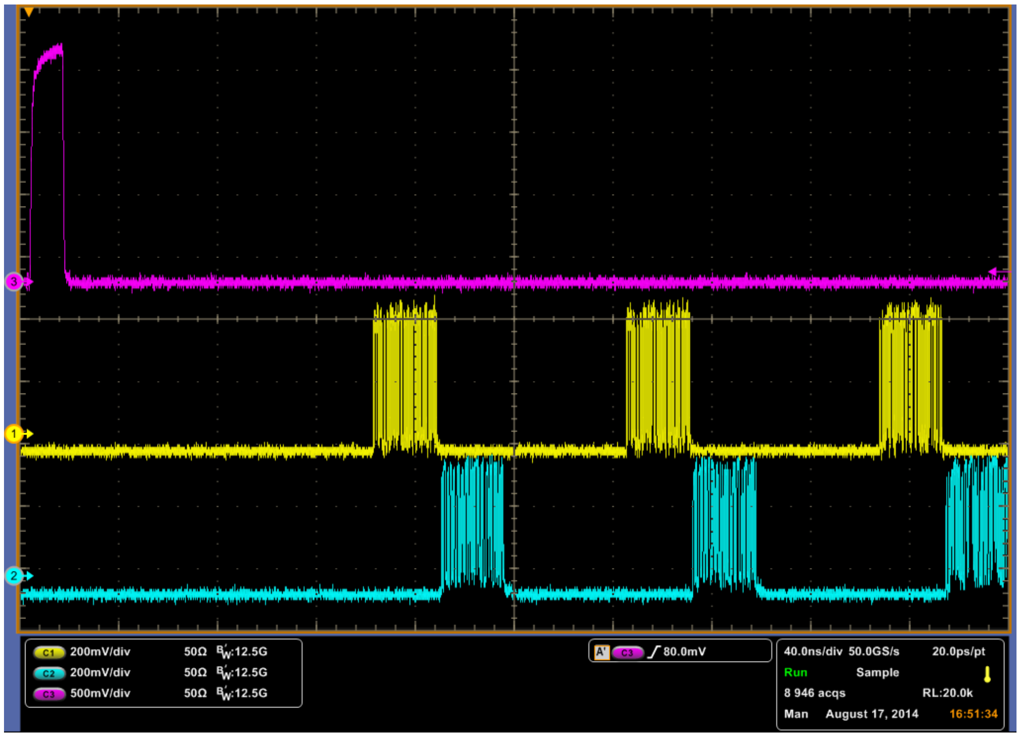
C1 200mV/div 50Ω B_W:12.5G
C2 200mV/div 50Ω B_W:12.5G
C3 500mV/div 50Ω B_W:12.5G
A' C3 80.0mV
40.0ns/div 50.0GS/s 20.0ps/pt
Run Sample
8 946 acqs RL:20.0k
Man August 17, 2014 16:51:34

Figure 20. Tx and Rx in yellow and blue along with the trigger in magenta generated by the FPGA for CLR.

Results obtained with the patterns generated from the FPGA are displayed in Figure 21. The results were verified with the experimental setup using the PPGs.

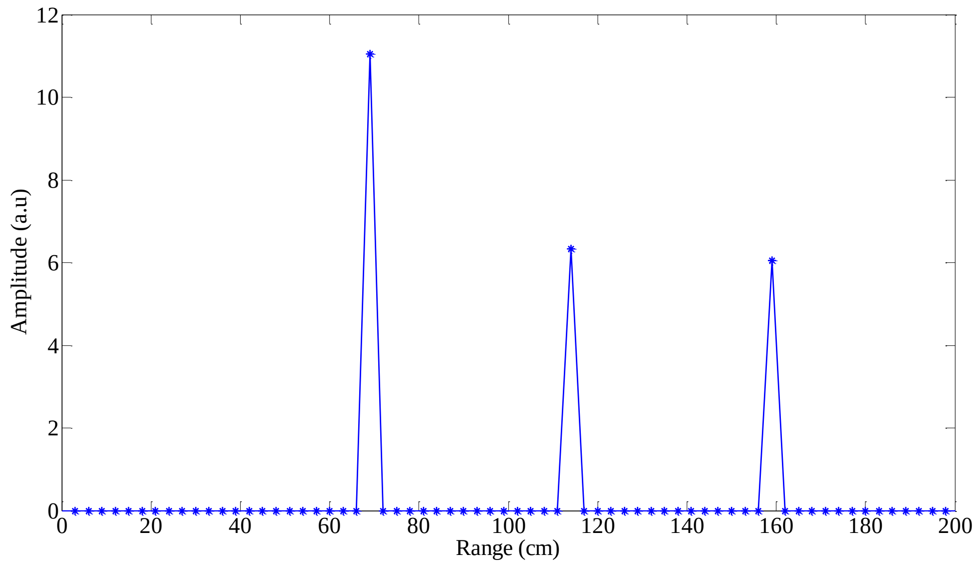

Figure 21. Compressive laser ranging results using patterns generated by FPGAs

Successful implementation of CLR required synchronized patterns with preset delay. Electronic systems generally use comma characters to synchronize the signals on the receiving end of the signal. However, for CLR, the waveforms needed to be synchronized at the transmitting end, which required extra coding and knowledge not provided intuitively with the Xilinx system. The problem became compounded when transceivers were used on the evaluation board as well as on the FMC (used for extra SMA ports), since the delay in transmission became less consistent. Without predefined modules to implement systems such as PCIE/Ethernet, implementation of multiple GTX transceivers in hardware required extra modules of frame generators to be programmed for all of the SMA transceivers to work simultaneously. One of the basic problems was the way the synthesis software converted from hexadecimal to binary, causing individual 4-bit words and the entire pattern to transmit in a state that was flipped left to right.

Predefined software could not be used to solve this problem since the receiving end of the transceiver was not used; therefore, VHDL functions had to be coded to correct the patterns. The VHDL code for the implementation of the state machine can be found in Appendix B.

SIGNAL RECOVERY USING A SOFT-CORE PROCESSOR

Implementation of Recovery Algorithms in C

Implementation of the compressive ranging setup on the FPGA is mainly divided into two parts; generating multi-gigabit patterns and processing the range using recovery algorithms. The first part of the implementation was demonstrated in Chapter 4, where ranging results were successfully obtained from patterns and a trigger generated by the FPGA. The second part of the implementation was demonstrated using a soft-core Nios-II processor on an Altera Stratix V FPGA board. The board is a transceiver signal integrity development kit, which has plenty of hardwired transceiver outputs to SMA ports. Altera provides IP for the soft-core processor and the transceivers that were used for the setup.

FPGAs gave us the option of doing computation in hardware as well as in software. Processing done in hardware is generally harder to implement since real-time troubleshooting is difficult and the code is not very modular since hardware redesign takes time. Although hardware computations are much faster than software calculations, instantaneous calculations are not a requirement for the processing part of the system. The decision was made to go with software computation and the code for the processing was written in C, making the implementation of certain numerical algorithms relatively simple compared to hardware languages such as Verilog or VHDL. Nios-II is a 32-bit RISC processor, which has the option to do multiplication in hardware, making some of the computations faster. It provides three options with increasing computational power.

The most computationally efficient processor was used, since most of the FPGA capacity was used for the processing. However, only 6 MB of memory was provided as on-chip memory with the FPGA, so the code had to be compact and memory efficient. The memory needed to accommodate both the patterns to be used as well as the essential matrices required in the recovery algorithms.

To provide memory-efficient computation, the C code was coded from scratch, without importing any libraries for complex linear algebra calculations. The code was made modular towards the processing side with all the variables and matrix dimensions kept variable and entered at the beginning of the processing step, since it is generally inconvenient to write a program that can deal with a matrix of arbitrary dimensions. In C, the use of pointers makes this process relatively easy and all the matrices in this demonstration were described using double pointers to keep the code consistent as well as modular. The algorithm implemented in this setup was the CoSaMP algorithm as described in chapter 2, although the program was designed such that any similar algorithms, such as OMP, STOMP and ROMP, could be implemented easily with minimal restructuring of the code.

The biggest challenge of the program was the implementation of the Moore-Penrose pseudoinverse of a matrix that is central to solving an underdetermined linear system. The common method used to implement the function is done using singular value decomposition (SVD), which factorizes the target matrix into three matrices of the form $\mathbf{M = U\Sigma V^*}$**,** where V^* is the transpose of the matrix V. U and V are transposed and multiplied in a certain order with the inverse matrix Σ^+ to obtain the pseudoinverse. Σ is a diagonal matrix and the inverse is obtained trivially by taking a reciprocal of the diagonal values. The Moore-Penrose pseudoinverse of the matrix M is given by $\mathbf{M^{+} = V\Sigma^{+}U^{*}}$. Functions for multiplying and transposing these matrices also were required as were functions for allocating and de-allocating memory used for these matrices. To keep the code memory efficient, memory allocated for matrices was freed as soon as the computation on that matrix was finished. Some of the computations also involved matrix and vector multiplication. The program was divided into three major code files with one containing the main body, one containing the function for the pseudoinverse and one containing all the functions for matrix calculations. The file containing the matrix operations contained the functions for matrix and vector multiplications of all forms and sizes, along with the functions for matrix and vector memory allocations and sorting of vectors and matrices. The code containing the pseudoinverse function was implemented separately, which included the complex function for SVD. The code for the SVD was based on the algorithm and code provided in the book Numerical Recipes in C by Teukolsky, Vetterling and Flannery.[39] The main file of the program was the implementation of the CoSaMP algorithm which asked for the known matrix from the user, the measurements and the dimensions of the known matrix. The three major files along with the board support package for the FPGA were the software part of the implementation of the recovery algorithm.

The hardware for the Nios-II processor was designed and implemented with 3MB of memory using the IP provided by Altera in the Quartus-II FPGA programming software provided with the FPGA board. The processor communicates with the computer through a USB-Blaster cable. The Eclipse Integrated Development Environment (IDE) package was used to create the board support package for the FPGA and compile the C code. The program went through several iterations of debugging and troubleshooting before successfully processing the accurate range. The results obtained from this program were compared with results obtained from the same algorithm in use with Matlab for the compressive ranging setup. The C code for the CoSaMP algorithm can be found in Appendix C.

Computational Constraints

All the computations performed in the implementation of the recovery algorithms were single precision floating-point calculations since the measurement matrix consisted of floating point values. Floating point operations are known as FLOPS and are used as the metric for measuring the time it takes for a system to go through a certain algorithm. For the recovery algorithm the computational resources were mostly concentrated towards the multiplication of matrices and the calculation of the Singular Value Decomposition of the known measurement matrix.

The known measurement matrix used for this project was an $m \times n$ matrix with $m = 30$ and $n = 255$. The measurement vector containing the collected data is an $m \times 1$ vector. The amount of memory required to store the known matrix and the measurement vector is ~245 kB. The number of flops required for a matrix-vector multiply is approximated to be $2mn$. The number of flops for a multiply of a 255x30 matrix and a 30x1 vector is 15300. The number of flops required will increase linearly as the number of measurements increases to hundreds and the range window dependent on n becomes thousands of bins long. The second most computationally expensive step was the least-square solution in the CoSaMP algorithm which required a Singular Value Decomposition to calculate the pseudoinverse of the matrix along with the multiplication of the matrix with the support vector. The computational cost was reduced due to the reduction of the columns of the known measurement matrix to twice the sparsity level of the signal, which was six in most cases. The computation included the implementation of the SVD algorithm and a multiplication of the three output matrices from the algorithm to compute the pseudoinverse. For the most common cases the values of *m* and *n* were 6 and 255, respectively. The number of flops required to implement the SVD algorithm is given by $2mn^2 + 11n^3$ and was calculated to be ~180 Mflops for the matrices used for this project and is the most expensive step compared to other steps.[40] The computational cost of the SVD algorithm is high but it provides a numerically stable solution that can work for rank deficient matrices as well, making it robust.

The total computational cost of the algorithm can be approximated to be ~190 Mflops. The NIOS-II can perform 1.13 instructions per clock cycle. For a 50 MHz clock rate the NIOS-II can reconstruct the sparse signal in ~3.5 seconds. For the compressive laser ranging setup, since the measurement vector needs to be fully collected before it can be used in the algorithm. The transmission of patterns and collection and sorting of data to form the measurement vector requires less than 100 ms. Thus the speed of the compressive laser ranging system is highly dependent on the reconstruction algorithm and can be improved by using a faster clock on the processor or using a board with a dedicated processor.

CONCLUSION

Compressive laser ranging was successfully demonstrated and implemented on an FPGA board. Single and multiple targets were resolved with fewer measurements than would have been required for Nyquist sampling. Multi-gigabit pattern generation for compressive sensing was implemented on an FPGA, along with the complex recovery algorithms required for the recovery of sparse signals. The ultimate goal of this project was to fit the entire compressive laser ranging setup inside a small box and create a complete 3D imaging system, as shown in Figure 22, for low transmit powers.

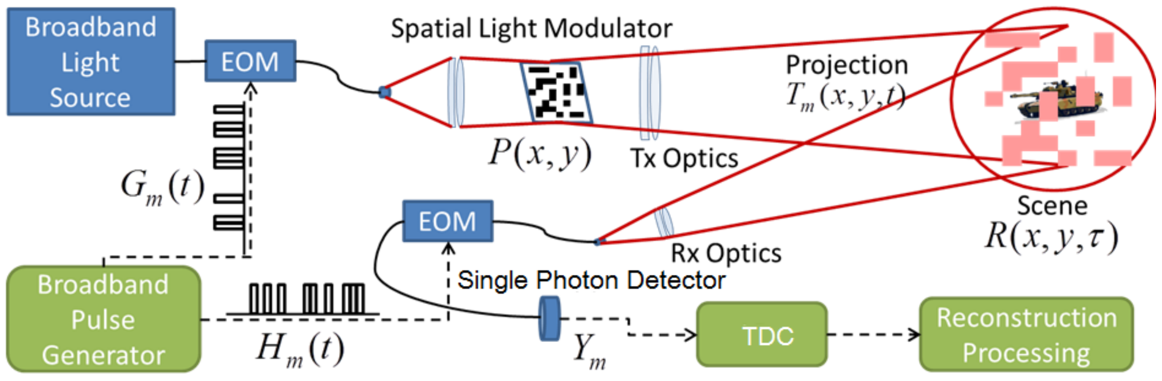

Figure 22. Compressively sensed 3D imaging system.

With the work reported in this thesis for a ranging system, a small compressive 3D imaging system could be made possible once it is combined with a compressive sensing system for 2D imaging. With further work, all the signal generation and processing could be programmed on one FPGA whereas currently the pattern generation and the processing are done on different FPGA boards. By consolidating resources on one board and porting the designs to work on the same board, all the computation could be done on the same board. FPGA boards come with SFP/SFP+ ports that use lasers to transmit data as binary bits, a system that is similar to the one used for producing the patterns used in CLR. Using them might eliminate the need for EOMs, thereby making the system more compact.

REFERENCES CITED

1. Gantz, J. & Reinsel, D. The Digital Universe Decade - Are You Ready? *IDC White Pap.* (2010). at <http://www.emc.com/collateral/analyst-reports/idc-digital-universe-are-you-ready.pdf>

2. Baraniuk, R. G. More Is Less: Signal Processing and the Data Deluge. *Science* **331,** 717–719 (2011).

3. Li, F., Cornwell, T. J. & de Hoog, F. The application of compressive sampling to radio astronomy: I. Deconvolution. *Astron. Astrophys.* **528,** A31 (2011).

4. Compressive Imaging: A New Single-Pixel Camera | Rice DSP. at <http://dsp.rice.edu/cscamera>

5. Baraniuk, R. G. Single-pixel imaging via compressive sampling. *IEEE Signal Process. Mag.* (2008). at <http://dsp.rice.edu/sites/dsp.rice.edu/files/publications/journal-article/2008/s-pixel-imaging-ieee-2008.pdf>

6. Levinson, J. *et al.* Towards fully autonomous driving: Systems and algorithms. in *2011 IEEE Intelligent Vehicles Symposium (IV)* 163–168 (2011). doi:10.1109/IVS.2011.5940562

7. McManamon, P. F. Review of ladar: a historic, yet emerging, sensor technology with rich phenomenology. *Opt. Eng.* **51,** 060901–1 (2012).

8. *Topographic Laser Ranging and Scanning: Principles and Processing*. (CRC Press, 2008).

9. *Lidar*. **102,** (Springer-Verlag, 2005).

10. Shaw, J. A. *et al.* Polarization lidar measurements of honey bees in flight for locating land mines. *Opt. Express* **13,** 5853 (2005).

11. Bauer, C., Burgmeier, J., Bohling, C., Schade, W. & Holl, G. in *Stand-Off Detection of Suicide Bombers and Mobile Subjects* (eds. Schubert, H. & Rimski-Korsakov, A.) 127–133 (Springer Netherlands, 2006). at <http://link.springer.com/chapter/10.1007/1-4020-5159-X_15>

12. Bobrovnikov, S. M. & Gorlov, E. V. Lidar method for remote detection of vapors of explosives in the atmosphere. *Atmospheric Ocean. Opt.* **24,** 235–241 (2011).

13. Wallin, S., Pettersson, A., Östmark, H. & Hobro, A. Laser-based standoff detection of explosives: a critical review. *Anal. Bioanal. Chem.* **395,** 259–274 (2009).

14. Babbitt, W. R., Barber, Z. W. & Renner, C. Compressive laser ranging. *Opt. Lett.* **36,** 4794–4796 (2011).

15. Poole, D. *Linear Algebra: A Modern Introduction*. (Cengage Learning, 2014).

16. Donoho, D. L., Drori, I., Tsaig, Y. & Starck, J. L. *Sparse solution of underdetermined linear equations by stagewise orthogonal matching pursuit*. (Citeseer, 2006).

17. Candes, E. J. & Tao, T. Near-Optimal Signal Recovery From Random Projections: Universal Encoding Strategies? *IEEE Trans Inf Theor* **52,** 5406–5425 (2006).

18. Li, S., Xu, L. D. & Wang, X. A Continuous Biomedical Signal Acquisition System Based on Compressed Sensing in Body Sensor Networks. *IEEE Trans. Ind. Inform.* **9,** 1764–1771 (2013).

19. Herrmann, F. J., Friedlander, M. P. & Yilmaz, O. Fighting the curse of dimensionality: compressive sensing in exploration seismology. *Signal Process. Mag. IEEE* **29,** 88–100 (2012).

20. Wang, Y., Cao, J. & Yang, C. Recovery of seismic wavefields based on compressive sensing by an l1-norm constrained trust region method and the piecewise random subsampling. *Geophys. J. Int.* **187,** 199–213 (2011).

21. Baraniuk, R. Compressive sensing. *IEEE Signal Process. Mag.* **24,** (2007).

22. Foucart, S. & Rauhut, H. *A Mathematical Introduction to Compressive Sensing*. (Springer New York, 2013). at <http://link.springer.com/10.1007/978-0-8176-4948-7>

23. Blumensath, T. & Davies, M. E. Iterative hard thresholding for compressed sensing. *Appl. Comput. Harmon. Anal.* **27,** 265–274 (2009).

24. Blumensath, T. & Davies, M. E. Iterative thresholding for sparse approximations. *J. Fourier Anal. Appl.* **14,** 629–654 (2008).

25. Mallat, S. G. & Zhang, Z. Matching pursuits with time-frequency dictionaries. *IEEE Trans. Signal Process.* **41,** 3397–3415 (1993).

26. Tropp, J. A. & Gilbert, A. C. Signal recovery from random measurements via orthogonal matching pursuit: The Gaussian case. (2007). at <http://authors.library.caltech.edu/27144/>

27. Needell, D. & Tropp, J. A. Cosamp: iterative signal recovery from incomplete and inaccurate samples. *Commun. ACM* **53,** 93–100 (2010).

28. Needell, D. & Vershynin, R. Uniform Uncertainty Principle and Signal Recovery via Regularized Orthogonal Matching Pursuit. *Found. Comput. Math.* **9,** 317–334 (2009).

29. Donoho, D. L., Tsaig, Y., Drori, I. & Starck, J.-L. Sparse Solution of Underdetermined Systems of Linear Equations by Stagewise Orthogonal Matching Pursuit. *IEEE Trans. Inf. Theory* **58,** 1094–1121 (2012).

30. Baraniuk, R., Davenport, M., DeVore, R. & Wakin, M. A simple proof of the restricted isometry property for random matrices. *Constr. Approx.* **28,** 253–263 (2008).

31. Bah, B. & Tanner, J. Improved bounds on restricted isometry constants for Gaussian matrices. *SIAM J. Matrix Anal. Appl.* **31,** 2882–2898 (2010).

32. Candes, E. J. The restricted isometry property and its implications for compressed sensing. *Comptes Rendus Math.* **346,** 589–592 (2008).

33. Davenport, M. A. & Wakin, M. B. Analysis of Orthogonal Matching Pursuit Using the Restricted Isometry Property. *IEEE Trans. Inf. Theory* **56,** 4395–4401 (2010).

34. Qaisar, S., Bilal, R. M., Iqbal, W., Naureen, M. & Lee, S. Compressive sensing: From theory to applications, a survey. *J. Commun. Netw.* **15,** 443–456 (2013).

35. Candes, E. J. & Wakin, M. B. An Introduction To Compressive Sampling. *IEEE Signal Process. Mag.* **25,** 21–30 (2008).

36. Romberg, J. Compressive sampling via random convolution. *PAMM* **7,** 2010011–2010012 (2007).

37. Maxfield, C. *The design warrior's guide to FPGAs devices, tools and flows*. (Newnes : Elsevier, 2004). at <http://www.books24x7.com/marc.asp?bookid=37283>

38. Athavale, A. & Christensen, C. *High Speed Serial I/O Made Simple*. (2005).

39. *Numerical recipes in C: the art of scientific computing*. (Cambridge University Press, 1992).

40. Trefethen, L. N. & III, D. B. *Numerical Linear Algebra*. (SIAM, 1997).

APPENDICES

APPENDIX A

COSAMP IN MATLAB

perform_CoSaMP.m

```
function a = perform_CoSaMP(D,u,s)

% a = s-sparse approximation of the target signal
% D = mixing matrix size m X N
% u = noisy samples size m x 1 (column vector)
% s = sparsity level

%%
[m,N] = size(D);
a = zeros(N,1);
v = u;
k = 0;
T = [];
%%
for k = 1:s;
    y = abs(D'*v);
    for kk = 1:2*s
        O(kk) = find(abs(y) == max(abs(y)),1,'first');
        y(O(kk)) = 0;
    end
    T = union(O,find(a));
    b = pinv(D(:,T))*u;
    bs = sort(abs(b),'descend');
    b = b.*ismember(abs(b),bs(1:s));

    a(T) = b;
    a_all(:,kk) = a;
    v = u-D*a
end

a = a_all;
```

APPENDIX B

CLR PATTERNS STATE MACHINE IN VHDL

CLR_PATTERN_GEN.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use std.textio.all;
use ieee.std_logic_textio.all;

--*****************************Entity Declaration************************

entity CLR_PATTERN_GEN is
generic
(
   WORDS_IN_BRAM : integer    :=   512
);
port
(
   -- User Interface
   TX_DATA_OUT            : out   std_logic_vector(79 downto 0);
   TX_DATA_OUT2           : out   std_logic_vector(79 downto 0);
   TXCTRL_OUT             : out   std_logic_vector(7 downto 0);
   TXCTRL_OUT2            : out   std_logic_vector(7 downto 0);
   TRIG                   : out   std_logic;
   -- System Interface
   USER_CLK           : in    std_logic;
   SYSTEM_RESET       : in    std_logic
);
end CLR_PATTERN_GEN;

architecture RTL of CLR_FRAME_GEN is
   attribute DowngradeIPIdentifiedWarnings: string;
   attribute DowngradeIPIdentifiedWarnings of RTL : architecture is "yes";
--**************************Parameter Declarations********************

   constant DLY : time := 1 ns;

--************************** Wire Declarations*************************

   signal  tx_ctrl_i             :   std_logic_vector(7 downto 0);
   signal  tx_data_bram_i        :   std_logic_vector(63 downto 0);
   signal  tx_ctrl_i2            :   std_logic_vector(7 downto 0);
   signal  tx_data_bram_i2       :   std_logic_vector(63 downto 0);
   signal  tx_data_ram_r         :   std_logic_vector(79 downto 0);
   signal  tx_data_ram_r2        :   std_logic_vector(79 downto 0);
```

```
    signal  tied_to_ground_vec_i    :   std_logic_vector(31 downto 0);
    signal  tied_to_ground_i       :   std_logic;
    signal  tied_to_vcc_i          :   std_logic;
    signal  tied_to_vcc_vec_i      :   std_logic_vector(15 downto 0);

--*********************Internal signalister Declarations********************

    signal  read_counter_i         :   unsigned(8 downto 0);
    signal  read_counter_conv      :   std_logic_vector(8 downto 0);
    signal  system_reset_r         :   std_logic;
    signal  system_reset_r2        :   std_logic;
    attribute keep: string;
    attribute ASYNC_REG                     : string;
    attribute ASYNC_REG of system_reset_r     : signal is "TRUE";
    attribute ASYNC_REG of system_reset_r2     : signal is "TRUE";
    attribute keep of system_reset_r:   signal is "true";
    attribute keep of system_reset_r2:   signal is "true";

--**************************User Defined Attribute*************************

    type RomType is array(0 to 511) of std_logic_vector(79 downto 0);

    function reverse_any_vector (a: in std_logic_vector)
       return std_logic_vector is
       variable result: std_logic_vector(a'REVERSE_RANGE);
    begin
       for i in a'RANGE loop
          result(i) := a(i);
       end loop;
       return result;
    end;

    impure function InitRomFromFile (RomFileName : in string) return RomType is

        FILE RomFile : text open read_mode is RomFileName;
        variable RomFileLine : line;
        variable ROM : RomType;
    begin
        for i in RomType'range loop
          readline (RomFile, RomFileLine);
          hread (RomFileLine, ROM(i));
        end loop;
        return ROM;
    end function;
```

```
   signal ROM  : RomType := InitRomFromFile("tr_dat.dat");
   signal ROM1 : RomType := InitRomFromFile("rx_dat.dat");

--************************Main Body of Code***************************
begin

   tied_to_ground_vec_i    <=   (others=>'0');
   tied_to_ground_i        <=   '0';
   tied_to_vcc_i           <=   '1';

   --___________ synchronizing the async reset for ease of timing simulation ________
   process( USER_CLK )
   begin
      if(USER_CLK'event and USER_CLK = '1') then
         system_reset_r <= SYSTEM_RESET after DLY;
         system_reset_r2 <= system_reset_r after DLY;
      end if;
   end process;

   --_______________________ Counter to read from BRAM ____________________

   process( USER_CLK )
   begin
      if(USER_CLK'event and USER_CLK = '1') then
         if((system_reset_r2='1') or (read_counter_i = (WORDS_IN_BRAM-1)))then
            read_counter_i <= (others => '0') after DLY;
            TRIG <= tied_to_vcc_i;
         else
            read_counter_i <= read_counter_i + 1 after DLY;
            TRIG <= tied_to_ground_i;
         end if;
      end if;
   end process;

   -- Assign TX_DATA_OUT to BRAM output

   process( USER_CLK )
   begin
      if(USER_CLK'event and USER_CLK = '1') then
         if(system_reset_r2='1') then
            TX_DATA_OUT      <= (others => '0') after DLY;
            TX_DATA_OUT2     <= (others => '0');
         else
```

```
            TX_DATA_OUT      <= (tx_data_bram_i & tx_data_ram_r(15 downto 0))
after DLY;
            TX_DATA_OUT2     <= (tx_data_bram_i2 & tx_data_ram_r2(15 downto 0));
         end if;
      end if;
   end process;

   -- Assign TXCTRL_OUT to BRAM output
   process( USER_CLK )
   begin
      if(USER_CLK'event and USER_CLK = '1') then
         if(system_reset_r2='1') then
            TXCTRL_OUT    <= (others => '0') after DLY;
            TXCTRL_OUT2   <= (others => '0');
         else
            TXCTRL_OUT    <= tx_ctrl_i  after DLY;
            TXCTRL_OUT2   <= tx_ctrl_i2;
         end if;
      end if;
   end process;

   --____________________________ BRAM Inference Logic_________________________

   tx_data_bram_i      <= reverse_any_vector(tx_data_ram_r(79 downto 16));
   tx_data_bram_i2     <= reverse_any_vector(tx_data_ram_r2(79 downto 16));
   tx_ctrl_i           <= tx_data_ram_r(15 downto 8);
   tx_ctrl_i2          <= tx_data_ram_r2(15 downto 8);

   read_counter_conv   <= std_logic_vector(read_counter_i);

   process (USER_CLK)
   begin
     if(USER_CLK'event and USER_CLK='1') then
       tx_data_ram_r  <= ROM(conv_integer(read_counter_conv)) after DLY;
       tx_data_ram_r2 <= ROM1(conv_integer(read_counter_conv));
     end if;
   end process;
end RTL;
```

APPENDIX C

COSAMP IN C ON NIOS-II

perform_CoSaMP.c

```
/*
 * perform_CoSaMP.c
 *
 *  Created on: Jan 4, 2015
 *      Author: Pushkar Pandit
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "matrix_operations.h"

void pinv(double **D, int m, int n, double **Dinv);

double *perform_CoSaMP(double **D, int m, int n, double *u, int s)
{
	int i,j,k,p, size_t;
	int *O, *T;
	double *a, *v, *y, *b, *Dxa;
	double max;

	a = dvector(1,n);
	v = dvector(1,m);
	T = ivector(1,2*s*s);


	for (i = 1; i <= m; i++){
		v[i] = u[i];
	}

	for (i = 1; i <= n; i++){
		a[i] = 0;
	}

	for (i = 1; i <= 2*s*s; i++){
		T[i] = n+1;
	}

	size_t = 2*s;
```

```
for(k = 1; k <= s; k++){

        double **Dt,   // D transpose for y
        **Dinv,        // Inverse of Ds
        **Ds;  // Subsection of D with pertinent columns

        Dt = dmatrix( 1, n, 1, m);      // Allocate memory for Dt
        y = dvector(1, n);                      // Allocate memory for y
        O = ivector(1,2*s);                     // Allocate memory for O

        printf("\nv before = \n");
        for (i = 1; i <= m; i++) {
                printf("\t%f ", v[i]);
                printf("\n");
        }
        dmtranspose(D, m, n, Dt);      // Transpose the matrix D
        dmvmult(Dt, n, m, v, m, y); // Calculate y = (D'*v);

        printf("\ny before pruning = \n");
        for (i = 1; i <= n; i++) {
                printf("\t%f ", y[i]);
                printf("\n");
        }
        i = 1;                                                  //
        for (j = 1; j <= 2*s; j++){                     //
                max = fabs(y[1]);                               //
                for (p = 1; p <= n; p++){        //
                        if (fabs(y[p]) >= max){ //
                                O[i] = p;          // Find the 2*s largest values in y
                                max = fabs(y[p]);       //
                        }                                               //
                }                                                       //
                y[O[i]] = 0;                            //
                i++;                                            //
        }                                                               //
        printf("\nY = \n");
        for (i = 1; i <= n; i++) {
                printf("\t%f ", fabs(y[i]));
                printf("\n");
        }
        free_dvector(y,1,n);    // Free the memory allocated for y

        size_t = 2*s; // Size of the vector T which holds the largest components of
A
```

```
                    //////////////////////////////////////////
                    for (i = 1; i <= 2*s; i++){                              //
                              T[i] = O[i];                                             //
                    }
//

//
                    for (i = 1; i <= n; i++){                                //
                              if(a[i] != 0){                                           //
                                        T[size_t + 1] = i;                             //
                                        size_t++;                   // Calculate T = union(O, find(a));
                                        a[i] = 0;                                                //
                              }
//
                    }
//

//
                    for(i = 1; i <= size_t; i++){                  //
                              for(j = i+1; j <= size_t; j++){       //
                                        if(T[i] == T[j]){                              //
                                                  T[j] = n+1;                                    //
                                        }
//
                              }
//
                    }
//
                    qsort(T + 1, 2*s*s, sizeof(int), cmpfunc);//
                    //////////////////////////////////////////

                    size_t = 0;
                    for(i = 1; i <= 2*s*s; i++){              // Calculate the size of T after the
union
                              if(T[i] < n+1){                               // for future operations
                                        size_t++;
                              }
                    }

                    Dinv = dmatrix( 1, size_t, 1, m);         // Allocate memory for inverse of Ds
                    Ds = dmatrix(1, m, 1, size_t);             // Allocate memory for Ds
                    b = dvector(1, size_t);                              // Allocate memory for b
which is the vector for least squares approx.
```

```
        for (i = 1; i <= m; i++){
                for(j = 1; j <= size_t; j++){
                        Ds[i][j] = D[i][T[j]];          // Get the relevant columns
from D in to Ds
                }
        }

        pinv(Ds, m, size_t, Dinv);                      // Invert matrix Ds
        dmvmult(Dinv, size_t, m, u, m, b);    // Multiply the inverted matrix and
the measurements for
        // the least squares approximation
        printf("\nb = \n");
        for (i = 1; i <= size_t; i++) {
                printf("\t%f ", b[i]);
                printf("\n");
        }

        for (j = 1; j <= s; j++){
                max = fabs(b[1]);
                for (p = 1; p <= size_t; p++){  // Find the s largest approximations
and update a
                        if (fabs(b[p]) >= max){
                                O[i] = p;
                                max = fabs(b[p]);
                        }
                }
                a[T[O[i]]] = fabs(b[O[i]]);
                b[O[i]] = 0;
                i++;
        }

        printf("\na = \n");
        for (i = 1; i <= n; i++) {
                printf("\t%f ", a[i]);
                printf("\n");
        }

        Dxa = dvector(1,m);

        dmvmult(D, m, n, a, n, Dxa);

        for (i = 1; i <= m; i++){
                v[i] = u[i] - Dxa[i];
```

```
		}
		printf("\nv = \n");
		for (i = 1; i <= m; i++) {
			printf("\t%f ", v[i]);
			printf("\n");
		}
		free_dvector(b, 1, size_t);
	}
	free_dvector(Dxa, 1, m);

	return a;
}
```