CHANNEL ESTIMATION USING COMPRESSED SENSING VIA SDR's

By

NC Syed Muhammad Hamza

ASC Ahmad Rasul

ASC Farhan Ali

PC Zohair Hassan

Submitted to the Faculty of Electrical Engineering, Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfilment for the requirement of a B.E Degree in Electrical Telecommunication Engineering

In the name of ALLAH, the Most Beneficent, the Most Merciful

ABSTRACT

Channel Estimation using Compressed Sensing via SDR's

Channel estimation is applied at the receiving end to get the reaction of channel so that effects of the channel can be calculated. A noble signal processing technique called compressed sensing is used for channel estimation. We are exploiting the sparsity of the channel in time domain by selecting the pilot symbols or preambles randomly and constructing a random projection measurement matrix.

We are using development kit GNU Radio which is software based. We are building a Channel Estimator which is based on Compressed Sensing and implemented on a Universal Software Radio Peripheral 1 (USRP1). Results and plots for compressed channel sensing will be shown to prove the better and effective conclusions as compared to earlier used channel estimation.

The approach we are following improves the channel estimation accuracy by saving the bandwidth effectively. We are developing a practical compressed sensing-based channel estimator. CS can be effective when high quality results/resolution is required and also when we have hardware limitations. Our project demonstrates that how important is the usage of SDRs to fill the gap between a theoretical model and practical implementations, which as a result boosts us to continue following various algorithms.

JULY, 2018

CERTIFICATE

It is hereby certified that the contents and the form of the project entitled "Channel Estimation using Compresses Sensing via SDR's" submitted by the syndicate of

1) Syed Muhammad Hamza
2) Farhan Ali
3) Ahmad Rasul
4) Zohair Hassan

has been found satisfactory as per the requirement of the B.E. Degree in Electrical (Telecom) Engineering.

Name (Supervisor): **Lt. Col Faisal Akram**

Signature: ___________________

Date: __________________

DECLARATION

NO PORTION OF THE WORK PRESENTED IN THIS DISSERTATION HAS BEEN SUBMITTED IN SUPPORT OF ANOTHER AWARD OR QUALIFICATION EITHER AT THIS INSTITUTION OR ELSEWHERE.

DEDICATED TO

Allah Almighty,

Faculty & Supervisor for their Help,

And our parents for their support and blessings.

ACKNOWLEDGEMENTS

Without the will of ALLAH Almighty there is no success possible. We are thankful to ALLAH, who has guided us, given us the strength and gave us the ability to accomplish this task. Whatever we have achieved, we owe it to Him, in totality. We are also thankful to our parents and family and well-wishers for their admirable support and their critical reviews. We are indeed very much thankful to our respected supervisor Lt. Col Faisal Akram for his help and supervision throughout the duration of our project till completion. It is not common that one finds a supervisor who is always available and finds the time for listening to all the problems and roadblocks that crop up in the course of performing project. His technical skills and editorial advice was essential to the completion of this project and has taught us innumerable lessons and insights on the workings of academic research in general.

TABLE OF CONTENTS

1. Introduction.. (11)

1.1 Organization of document………………………………………………….. (11)

1.2 Overview……………………………………………………………………… (11)

1.3 Problem statement……………………………………………………………. (12)

1.4 Approach……………………………………………………………………… (12)

1.5 Salient features…………………….. ………………………………………… (13)

1.6 Objectives…………………………... (13)

2. Literature review……………………………………………………………… (14)

2.1 Channel estimation…………………………………………………………… (14)

2.2 Compressed sensing………………………………………………………….. (14)

2.3 GMSK modulation………………………….. (16)

3. Design and development…………………………………………………… (17)

3.1 Technical specification……….. (17)

3.1.1 Hardware components……………………………………………………... (17)

3.1.2 Software used…………….. (18)

3.2 Design summary…………………………………………………………... (19)

3.2.1 Transmission model…………………………………………………… (20)

3.2.2 Receiver model……………………………………………………… (20)

3.2.3 Overall flow diagram of project……………………………………….... (21)

3.3 Completion phase………………………………………………………………… (22)

3.3.1 CoSaMP…………………………………………………………………….. (22)

3.3.2 Implemented transmitter model……………………………………………..... (23)

3.3.3 Implemented receiver model with compressed sensing............................... (24)

3.3.4 Channel estimation of the received training sequence…………………… (27)

4. Analysis and testing……………………………………………………… (29)

4.1 Introduction……………………….………………………………………......... (29)

4.2 Conclusion and future work…………………………………………………….. (30)

5. References…………………………………………………………….......... (31)

Appendix A……………………………………………………………………..... (31)

Appendix B…………………………………………………………………….. (33)

Appendix C………………………………………………………………… (34)

LIST OF FIGURES

Fig 2.1: Compressed sensing …………………………………………………............ (15)

Fig 2.2: Compressed sensing flow graph……………………………………….. (15)

Fig 3.1: USRP Motherboard…………….. (18)

Fig 3.2: Arrangement of pilots ……………………………………………………. (19)

Fig 3.3: TX Flow diagram.....................……………………………………………. (20)

Fig 3.4: RX Flow diagram………………………………………………………… (21)

Fig 3.5: Flow diagram of project………………………………………………… (22)

Fig 3.6: CoSaMP…………………………………………………………………… (23)

Fig 3.7: USRP Transmitter Model……………………………………………….. (24)

Fig 3.8: Results…………………………………………………………………….. (28)

Fig 3.9: USRP Receiver Model with Compressed Sensing…………………........... (25)

Fig 3.10: Results………………………………………………………………..... (25)

Fig 3.11: Standalone file…………………………………………………….... (26)

Fig 3.12: Results…………………………………………………………………… (26)

Fig 3.13: Transmitter model of channel estimation………………………………. (27)

Fig 3.14: Receiver model of channel estimation………………………………..... (28)

Fig 3.15: Output of correlation estimator……………………………………………. (28)

Fig 4.1: BER curve……………………………………………………………… (29)

Chapter 1-Introduction

1.1 Organization of Document:

This document is divided into six main parts:

- First part introduces the project.
- Second part tells about background related to this field.
- Third part is about the literature review.
- Fourth part is about design, implementation and analysis of results.
- Fifth part enlightens future improvements that can be made.
- Final part is about references and bibliography.

1.2 Overview:

The classical technique used for channel estimation was complex as it was limited to frequency domain only. Higher transmission was not possible in classical channel estimation.

In communication systems, (CSI) is used where the channel characteristics are known. The already known information tells us that how a signal is propagated from transmitter side to the receiver side and displays the effects caused by the channel like reflection and dispersion etc. The method used is known as Channel estimation. The CSI causes the possibility of transformation of transmissions to the present conditions of the channel used, which causes an ease in gaining high data rates in real-environment communications.

1.3 Problem Statement:

We communicate wirelessly. Be it mobile phones or radio communication. The only worrying problem faced by the sector is lack of frequency spectrum. We take samples at the conventional Nyquist rate of twice the signal bandwidth which guarantees perfect recovery of the original signal but it's not bandwidth efficient.

Compressed sensing exploits the sparsity of a signal and we are able to completely reconstruct our signal at sub Nyquist rates. CS exploits redundancy to reduce the number of samples to describe a complete signal.

This technique modifies the channel estimation results to accurate level by conserving the bandwidth, hence overall throughput is increased.

1.4 Approach:

In this project, the problem of useful technique of channel estimation is analysed and a solution is proposed.

We plan to design a good and appreciable Compressed Sensing (CS) structure to obtain the CS estimates and to acknowledge the sparse channel. We will be using software skills i.e. Matlab, C++, Python to build the customized blocks in GNU radio.

The increased demand of software radio systems made us to aim on use such a design model. We are looking towards the specifications that the hardware platform, USRP1 offers and the radio software development kit, GNU Radio in a LINUX based environment.

1.5 Salient Features:

- Accurate
- Bandwidth efficient
- Less Margin of Error
- Use of Image Processing Algorithms
- Use of USRP

1.6 Objectives:

To implement CS technique and resultantly design a communication model is one of the core objectives. To achieve this goal, we will be designing our project on GNU Radio in Ubuntu LINUX environment and using USRP1 kit.

Chapter 2-Literature Review

The literature available for this project is explained below:

2.1 Channel Estimation

In real environment communication systems, the signal which we transmit is reflected and reaches the receiver end of the model as dispersed and shifted form of the original signal. To cater these effects, channel estimation is applied at the receiving end of our communication environment. [1]

By combining the pilots channel estimation is implemented. In this technique pilots are arranged in such a manner that it is already known to the transmitter side and fed to the receiver end. Due to high demand of data rate channel estimation should be preferably used.

2.2 Compressed Sensing:

Compressed sensing is a technique that uses less information to retrieve the required signal. Compressed sensing (CS) demonstrates a sampling framework based on the rate of information of a signal, thereby minimizing redundancy during sampling. [2] This means that a signal can be recovered from far fewer samples than generally required. As the channel estimates hn is of the nature having sparsity, we can reconstruct it by building a vigilantly created measurement matrix and by the use of optimization methods like l1 minimization. As our aim is to take more from the less in a communication based

environment, CS is a bright method. [3] The equation of measurement matrix in compressed sensing is shown in Fig 2.1

$$y = \phi * hn \ldots\ldots\ldots\ldots\ldots\ldots\ (1)$$

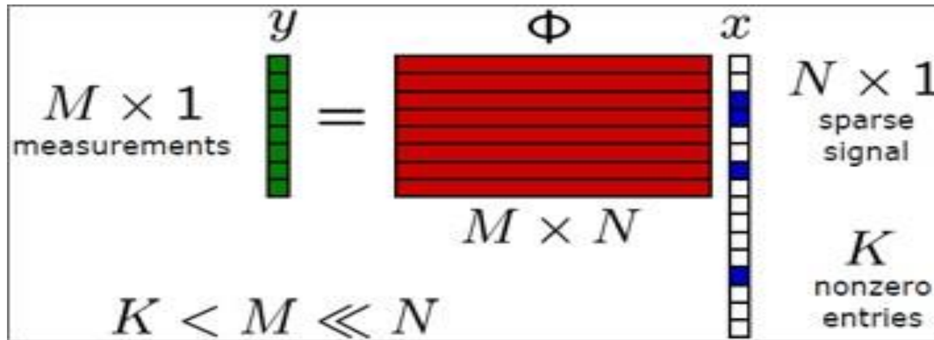

Fig 2.1 Compressed sensing

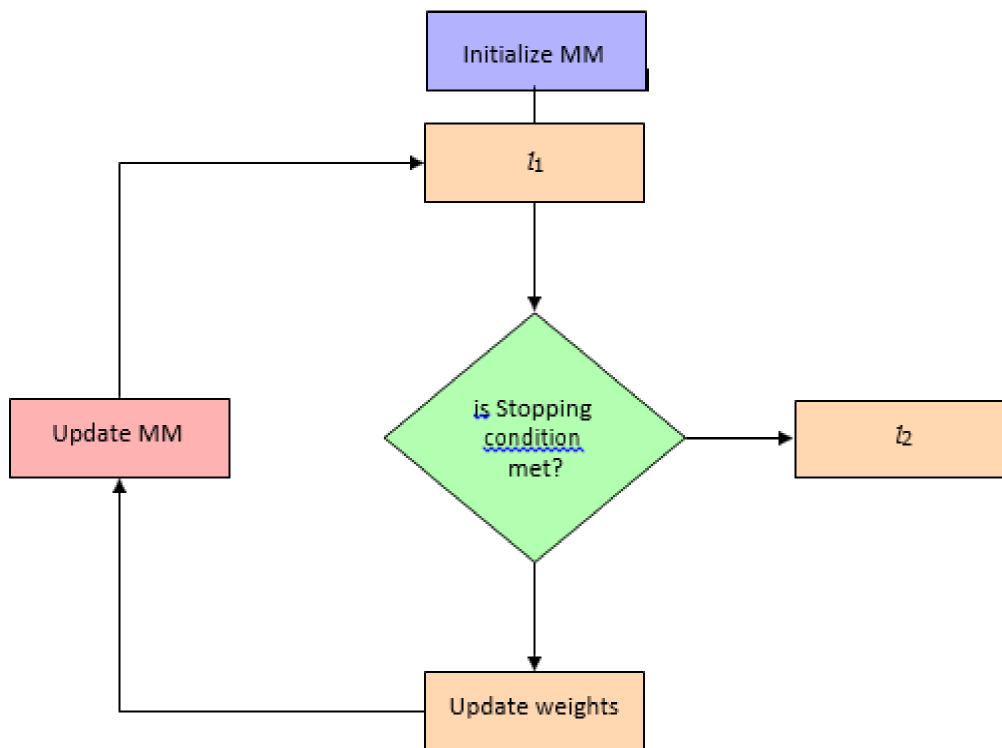

Fig 2.2 Compressed Sensing Flowchart

2.3 GMSK Modulation:

Gaussian Minimum Shift Keying, GMSK, is a form of frequency modulation that is used in wireless communication based systems.

GMSK is the most widely associated with the 2G GSM mobile communications system where it proved to be an effective form of modulation. It was one of the reasons that GSM cell phones had a long battery life in view of the high efficiency that could be obtained from the RF power amplifiers.

Chapter 3-Design and Development

3.1 Technical Specification:

3.1.1 Hardware Components:

- USRP1
- Laptops
- Antennas
- Power cables
- Ethernet Gigabit cable

3.1.1.1 USRP:

The USRP1 stands for Universal Software Radio Peripheral hardware (USRP) which has the potential to process RF. It's hardware includes 128 MS/s dual DAC, 64 MS/s dual ADC and USB 2.0 connection. USRP works at frequency of DC to 6 GHz. The USRP1 platform can support two RF daughterboard at the same time.

3.1.1.2 Antenna:

The main daughter board and antenna to be used in the current setup is RFX 400 with range from 400 MHz to 500 MHz with a bandwidth of 40 MHz .The maximum gain for this daughterboard is 45 db.

Fig 3.1   USRP Motherboard

3.1.2 <u>Softwares Used:</u>

- GNU RADIO
- MATLAB
- Python

- **<u>GNU Radio:</u>**

The software we are currently using is GNU Radio which is well matched with the USRP design. The GNU Radio has blocks which is able to apply software radios. So basically all the low pass working is implemented in the GNU Radio. We are using the latest version of GNU Radio for our setup. GNU Radio uses Python as a coding language to build connectors and customized blocks are made in C++. [5]

- **<u>MATLAB:</u>**

MATLAB software is used to plot the functions we use for our project design.

- **<u>Python:</u>**

Python is used as a high-level language which is widely used as programming.

3.2 <u>Design Summary:</u>

The hardware (H/W) platform to implement our project is the Universal Software Radio Peripheral manufactured by Ettus Research (Organization). It is used to perform experiments wirelessly using Radio Frequency.

To locate the starting position of the symbol, already familiar symbols are used and many models of synchronization are present in GNU Radio. The Schmidl and Cox (SC) design is used to construct the pilot symbols in the present USRP 1 experiment. The pilots familiar symbols used to synchronize and starting channel estimation. Arrangement of preambles is shown in Fig (3.2)

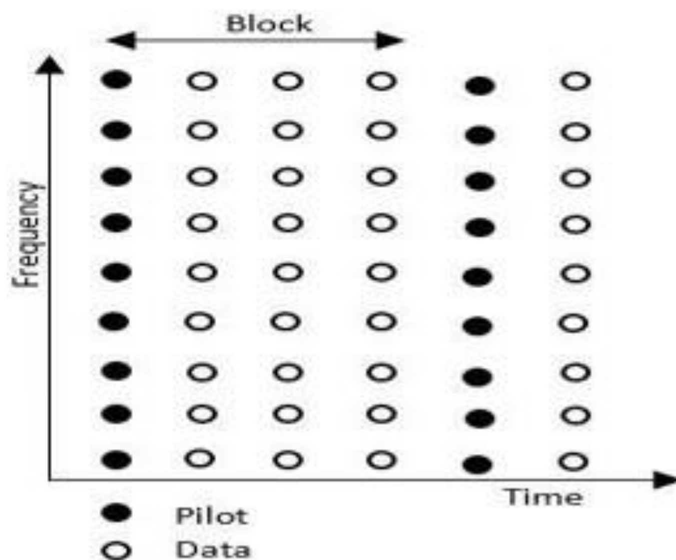

Fig 3.2: Arrangement of Pilots

3.2.1 Transmission Model:

The transmitter section consists of a vector source through which input as a vector is fed to the packet encoder block in GNU radio. The value of the parameter payload is set to x (bytes). This means the encoder takes in those bytes at a time, and these bytes get wrapped into a packet with a header, access code, and preamble, and the packet is sent on to the next block. The output of the packet encoder is fed to the GMSK modulation block. This block expects the input to be packed bytes (e.g. A in binary). It then breaks each byte apart into seperate bits and that is modulated and displays the complex (baseband data).The output of this block is transmitted wirelessly through USRP connected with the laptop.

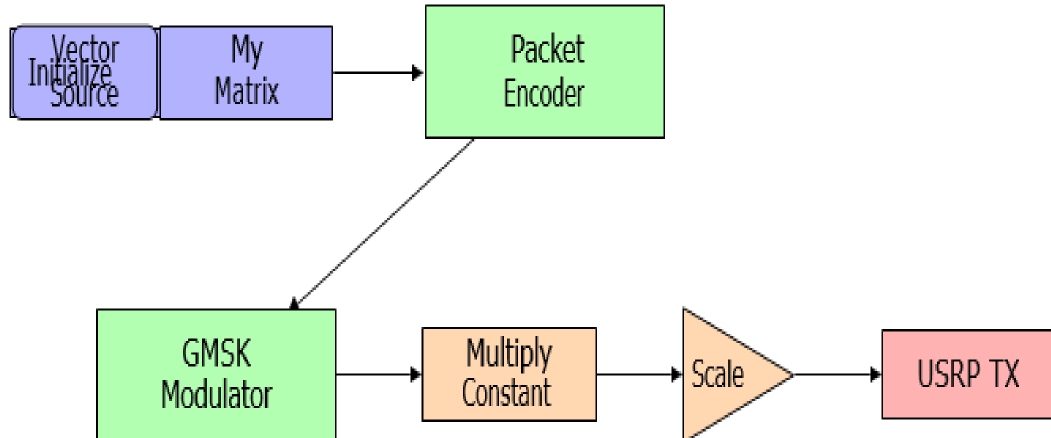

Fig 3.3: TX Flow Diagram

3.2.2 Receiver Model:

At the receiver, after USRP, the GMSK demodulator block is connected. This block shows the numerical complex values which is then demodulated to produce bytes that can be any

of these i.e. 0x00 or 0x01.The demodulator block is connected with the packet decoder block which unwraps the packet by removing the header and access code.

The flow diagram for the receiver is shown in Fig 3.4.

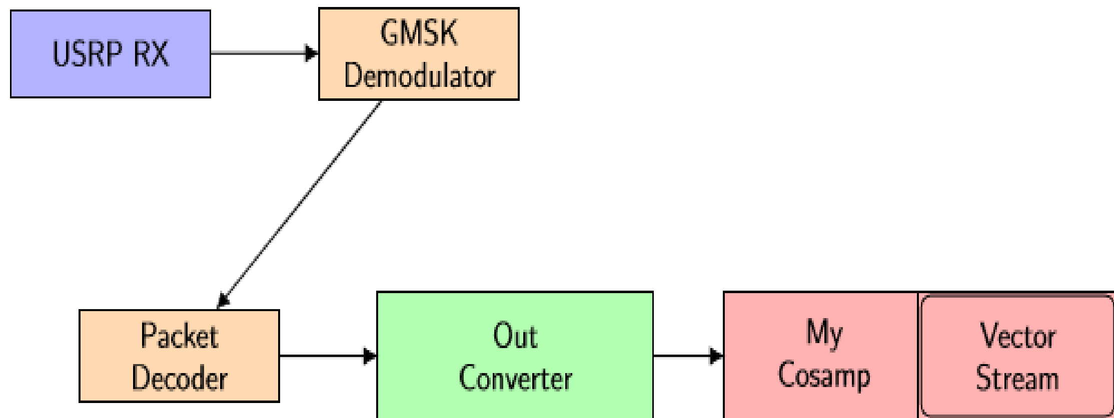

Fig 3.4: RX Flow Diagram

3.2.3 <u>Overall Flow Diagram of Project:</u>

An initial vector source was define with the name Fyp source and it is multiplied with the sensing matrix which reduces its dimensions.

The matrix is passed through the packet encoder which wraps the header and access code in the form of packets which is also the requirement for modulation. The GMSK modulation is used for the modulation of the data to be transmitted. In this modulation technique the phase shift of the carrier occurs .The modulated data is transmitted wirelessly through USRP [5] and is passed through a channel which introduces its effects on the transmitted data.

At the receiver end the received data file is demodulated and then passed through the packet decoder which unwraps the header and access code.

After passing through the packet decoder block, CoSaMP block is designed and used. CS is the technique which we use to reconstruct signals which are of high order dimensions. This reconstruction is done from a small number of measurements by utilizing the sparse properties of the signal. CS is based upon the basic fact that we can demonstrate large number of signals using only a few non-zero coefficients using a dictionary or needed basis.

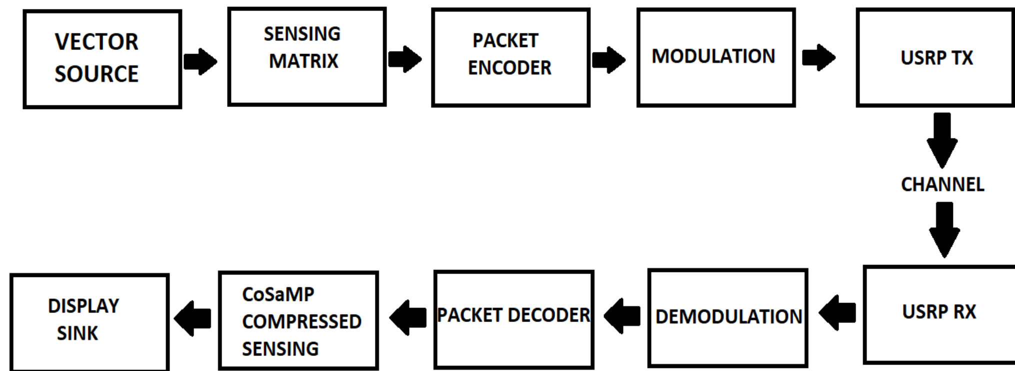

Fig 3.5: Flow diagram of project

3.3 Completion Phase:

3.3.1 CoSaMP:

The complex portion to reconstruct signal is by tracing the positions of those signal parts with the large or big values in the aimed signal. CoSaMP works by using a technique as used in restricted isometric property. Let's we have that the sampling matrix has constant δs1. For an s-sparse signal x, the vector y=Φ∗Φx can act as a proxy. The algorithm initiates the idea to iteratively approximate the target signal. As the algorithm keeps going on, the samples are constantly made up to date. This procedure is made to repeat again and again until we get energy which can be extracted from the signal.

Algorithm 1 CoSaMP Recovery Algorithm

CoSaMP (Φ, **u**, *s*)
Input: *Sampling matrix* Φ, *noisy sample vector* **u**, *sparsity level* s
Output: *An* $s-$ *sparse approximation* **a** *of the target signal*

$\mathbf{a}^0 \leftarrow 0$ ⊳ Trivial initial approximation
$\mathbf{v} \leftarrow \mathbf{u}$ ⊳ Current samples=Input samples
$k \leftarrow 0$

repeat
$k \leftarrow k+1$
$\mathbf{y} \leftarrow \Phi * \mathbf{v}$ ⊳ Form signal proxy
$\Omega \leftarrow \text{supp}(\mathbf{y}_{2s})$ ⊳ Identify large components
$T \leftarrow \Omega \cup \text{supp}(\mathbf{a}^{k-1})$ ⊳ Merge supports

$\mathbf{b}|_T \leftarrow \Phi_T^{+}\mathbf{u}$ ⊳ Signal estimation by least squares
$\mathbf{b}|_{T^c} \leftarrow 0$

$\mathbf{a}^k \leftarrow \mathbf{b}_s$ ⊳ Prune to obtain next approxiamtion
$\mathbf{v} \leftarrow \mathbf{u} - \Phi\mathbf{a}^k$ ⊳ Update current samples
until *halting criterion true*

Fig 3.6: CosAmP

3.3.2 Implemented Transmitter Model

We made a transmitter side using Gaussian Minimum Shift Keying (GMSK) modulation with a vector source acting as Input (I/P).

The vector had to be sparse in nature for perfect recovery to be achieved. As mentioned earlier in the theory of compressed sensing and its details. The following was the transmitter used, made and simulated in GNU Radio:

The implemented transmitter model in GNU is shown in Fig 3.7 and the calculated result is shown in Fig 3.8

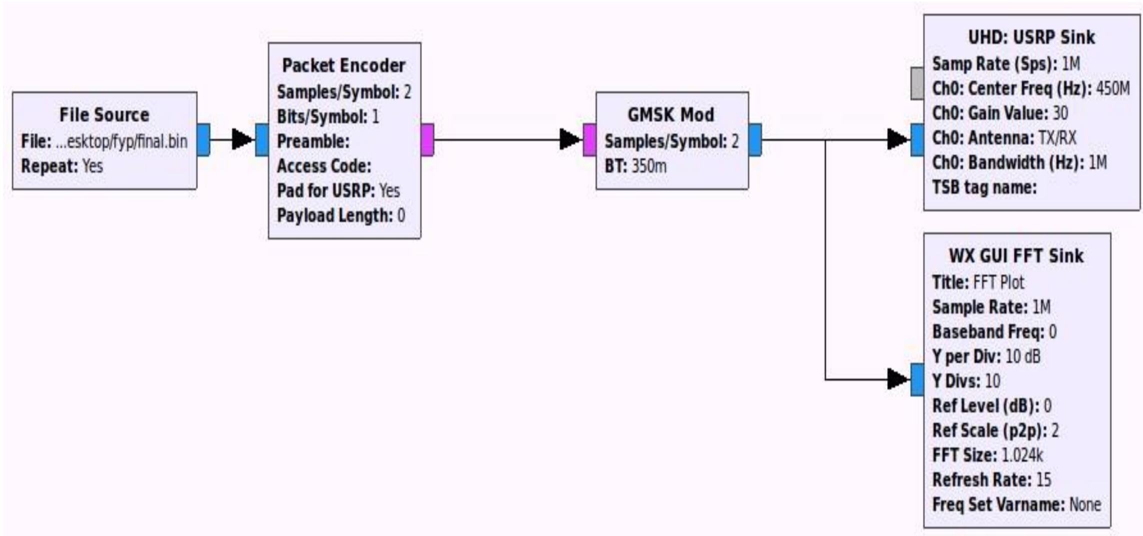

Fig 3.7 USRP Transmitter Model

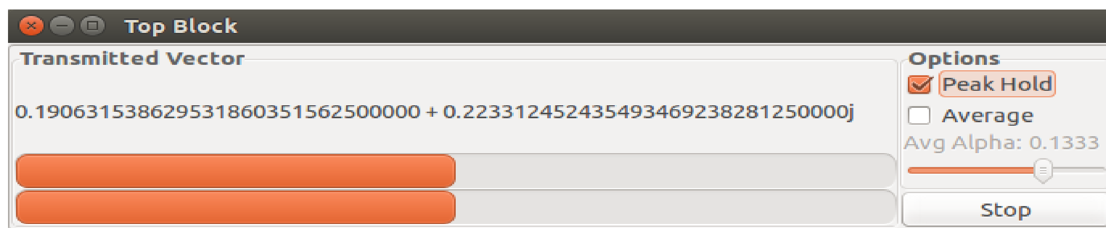

Fig 3.8 Results

3.3.3 Implemented Receiver Model with Compressed Sensing:

The receiver side was also made and simulated on GNU Radio simulation software. The Rx had to be made carefully according to the parameters of the transmitter. One of the problems faced in making the CoSaMP block was the compatibility with GNU Radio. The block was coded in C++ and worked on data type double, while GNU is configured to work on float data. This issue was resolved by converting the data with our very own customized converter blocks. The "Out Converter" converted float to double, while the "My Converter" block successfully converted the recovered data from CoSaMP back to float

for GNU understanding and compatibility. The implemented receiver model is shown in Fig 3.9 and calculated result is shown in Fig 3.10

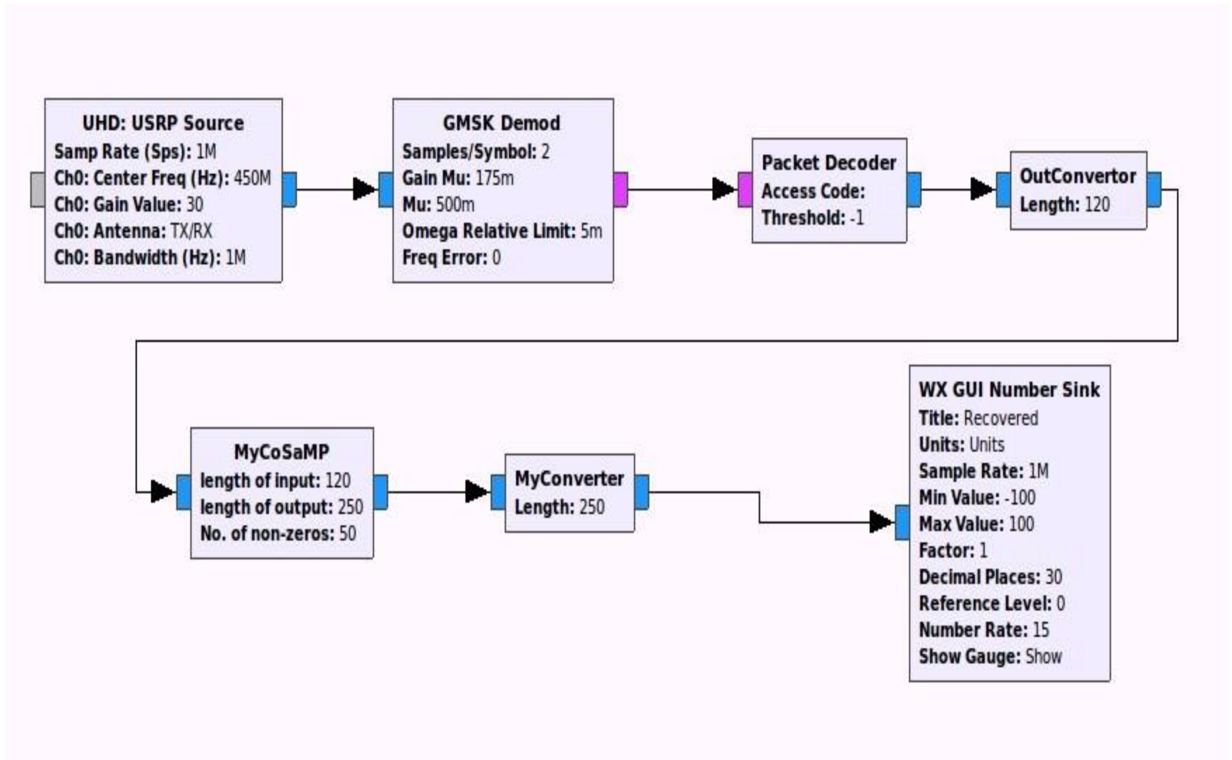

Fig 3.9 USPR Receiver Model with CoSaMP

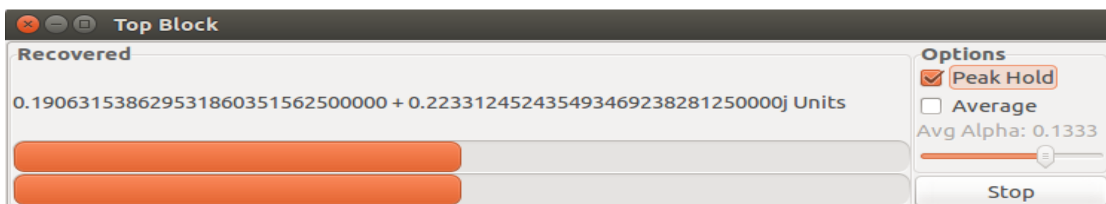

Fig 3.10 Results

The signal can be recovered from far fewer samples than generally required. As the channel estimates hn is of the nature having sparsity, we can reconstruct it by building a vigilantly created measurement matrix and by the use of optimization methods like l1 minimization. An algorithm which is called as CoSaMP assures the same results as an ideal optimization technique.

The standalone file of our communication model and its result is shown in Fig 3.11 and Fig 3.12.

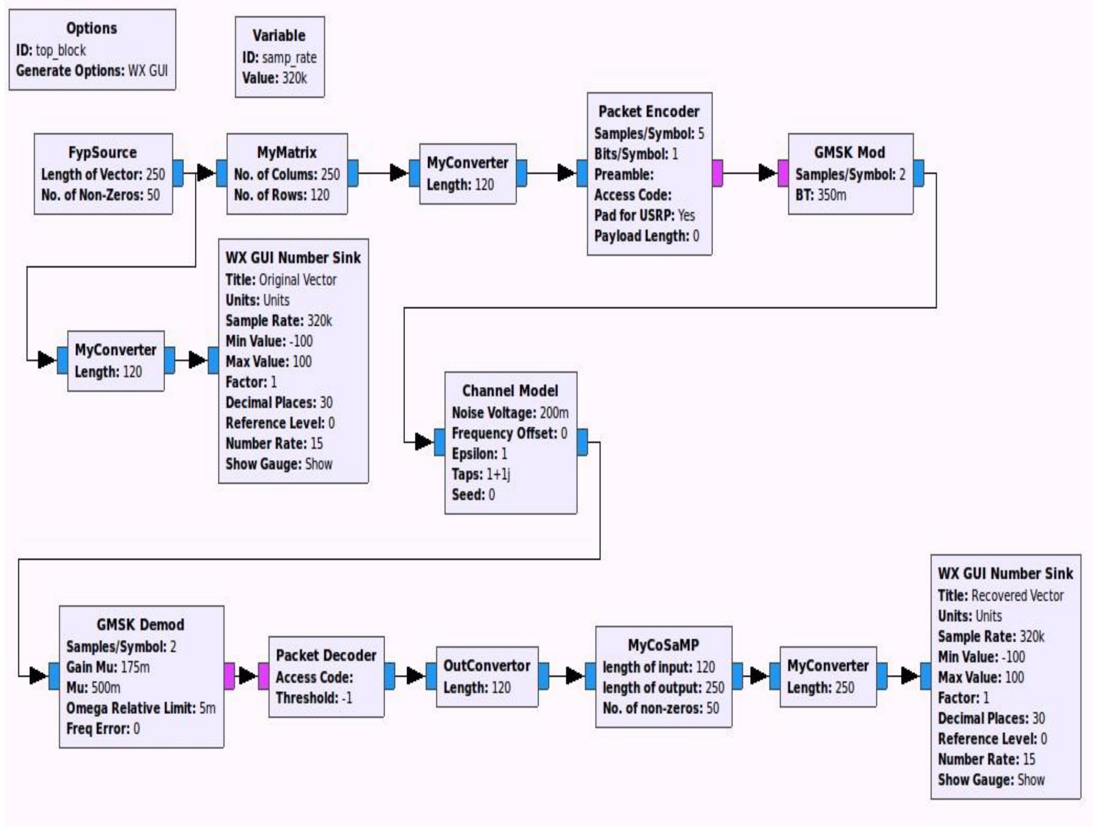

Fig 3.11 Standalone file

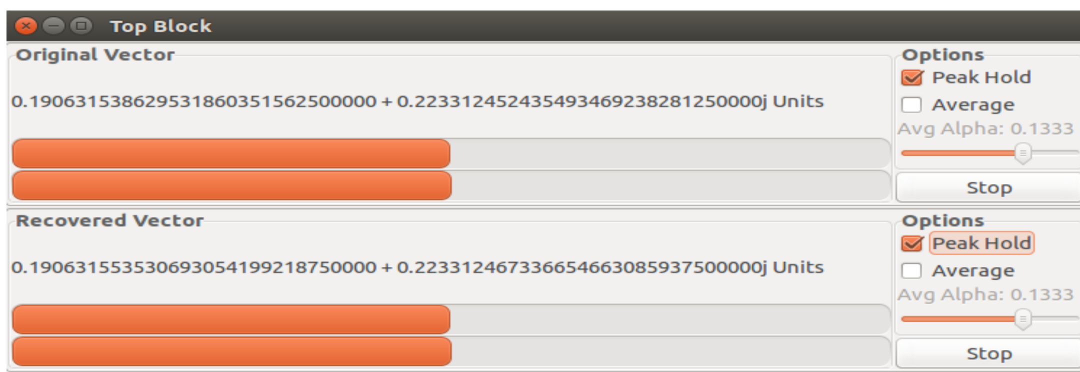

Fig 3.12 Results

3.3.4 Channel Estimation of the Received Training Sequence

A: Transmitter Model of the Channel Estimation:

At the transmitter end the training sequence is encoded and modulated for transmission and is connected to USRP for the transmission of sequence.

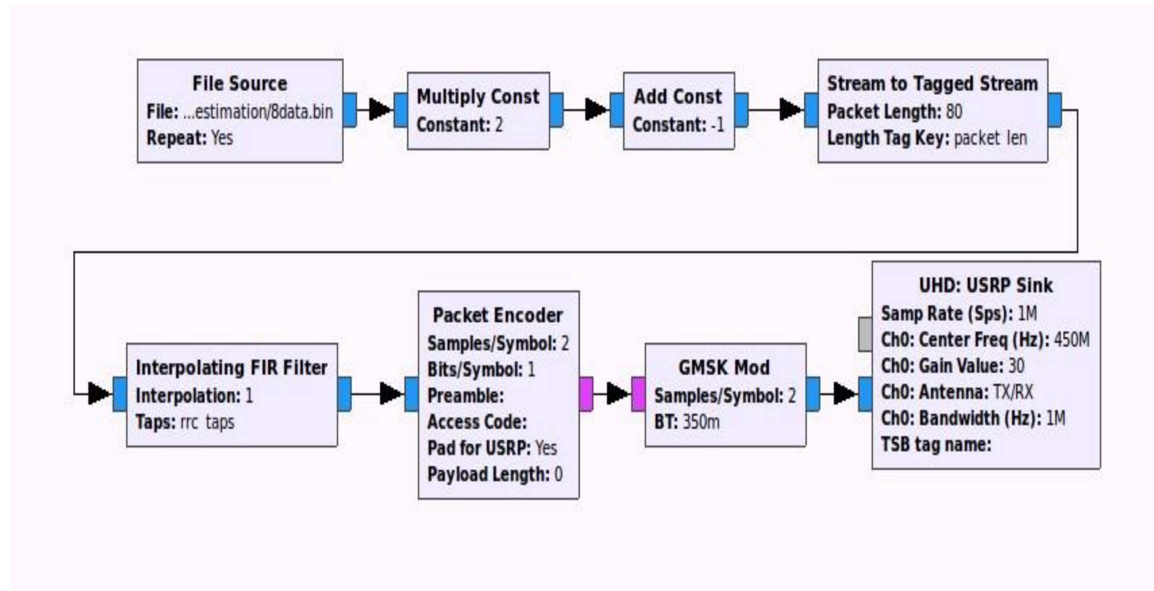

Fig 3.13 Transmitter model of channel estimation

B: Receiver Model of the Channel Estimation

The training sequence passes through the channel before reaching at the receiver end. This training sequence is then demodulated and decoded and is connected to the correlation estimator block This block auto-correlates the original and the received training sequence and outputs the result of multipaths in the form of taps. The output of correlation estimator is shown in Fig 3.14.

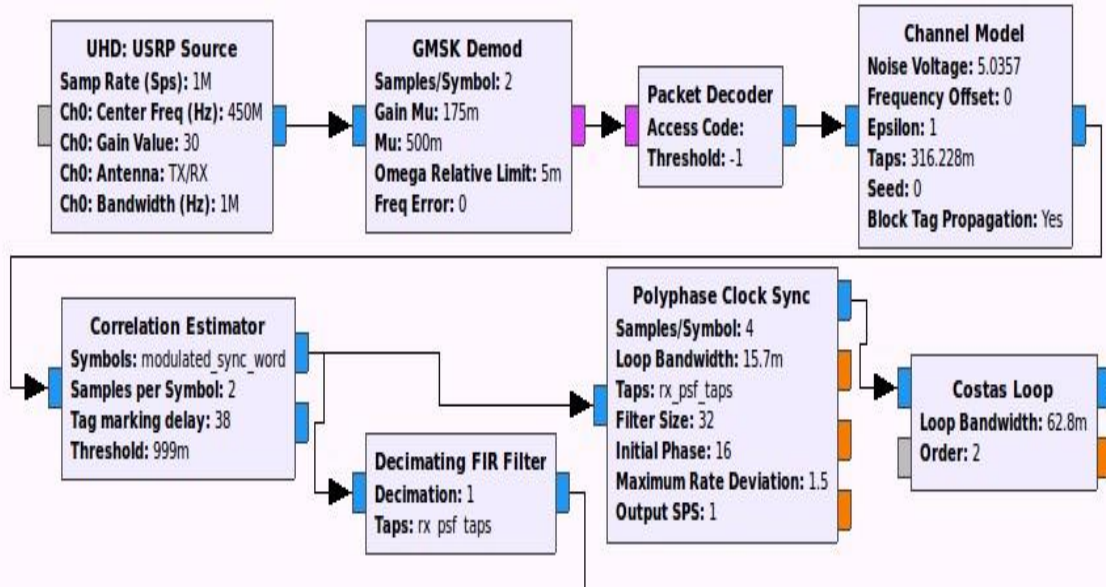

Fig 3.14 Receiver model of channel estimation

C: <u>Results of Receiver Model:</u>

This block searches for a sync word by correlation and uses the results of the correlation to get a time and phase offset estimate. These estimates are passed downstream as stream tags for use by follow-on synchronization blocks. The output of the correlation estimator is shown in the form of taps which actually represents the multipath effects of the channel.

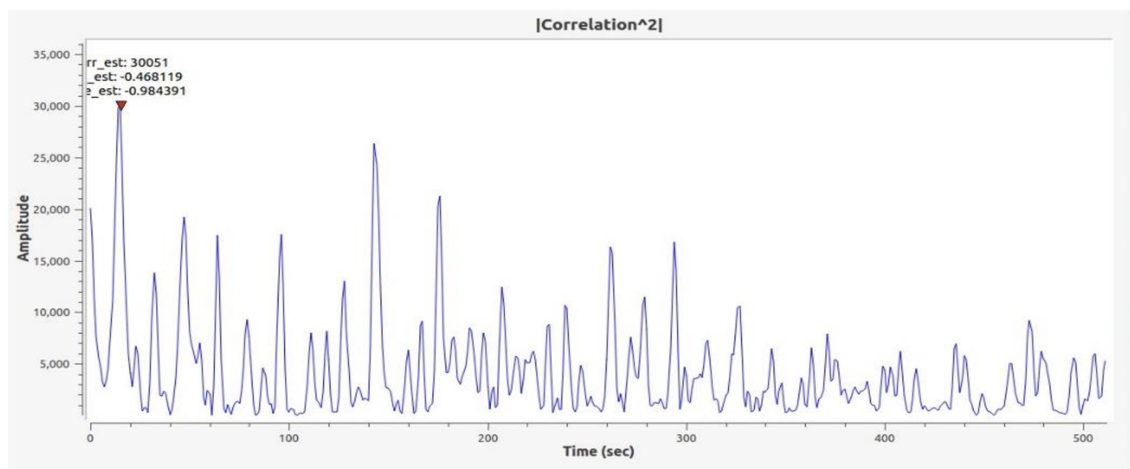

Fig 3.15 Output of correlation estimator

CHAPTER 4-Analysis and Testing

4.1 Introduction:

This chapter describes the simulation result for testing the BER of GMSK which is implemented on USRP1 and software used is GNU Radio. The result is discussed and possible future enhancements for the project are described.

Bit Error Rate (BER) is actually the number of error bits received divided by the total number of bits transferred. We can estimate the BER by calculating the probability that a bit will be correctly transferred due to interference.

One of the challenges faced in digital communications systems is the need for Tx to Rx performance measurements. The measure of that performance is called bit-error rate (BER), which calculates the reliability of the entire communication system from "bits received" to "bits transmitted".

Our BER is as follows:

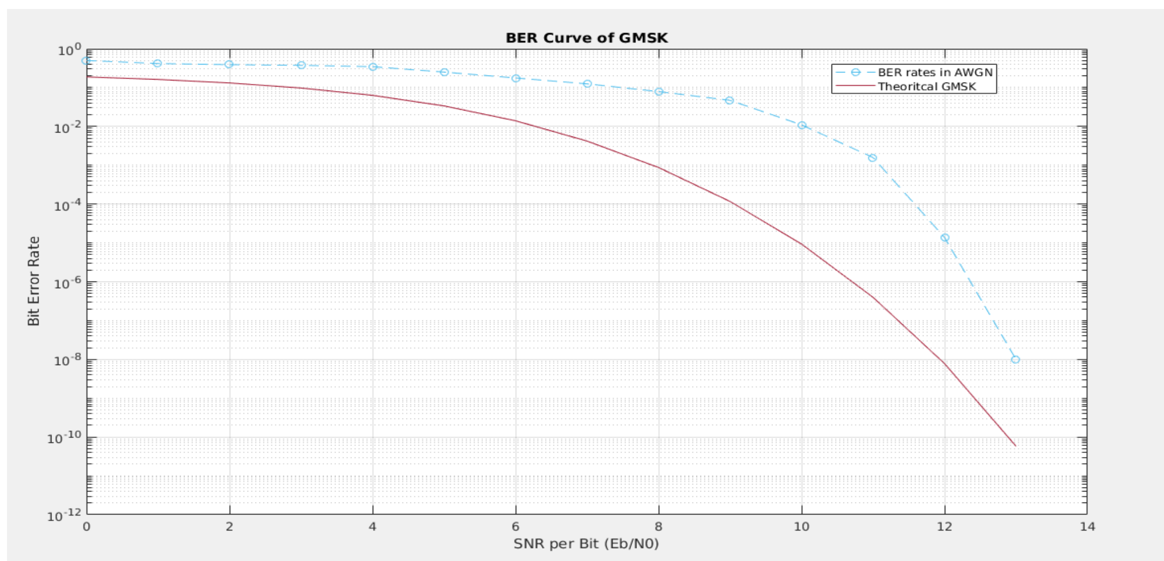

Fig 4.1 BER curve

4.2 Conclusion and Future Enhancements:

Every project faces many difficulties till the completion stage. This research/project also had some challenges. Major of the challenges were resolved. Simulations are easier to build whereas translating it to a real scenario needs a lot of updating and re-design.

One of the difficult challenge was building the C++ modules in the GNU Radio with Python to SWIG and C++ interfaces. We built a stand-alone signal processing block in our project for the first time. So it took us quite difficulty and a lot of time to grasp an understanding the underlying issues while designing a signal processing model.

APPENDIX A

CHANNEL ESTIMATION USING COMPRESSED SENSING VIA SDR's

Extended Title: Channel estimation using compressed sensing via SDR's.

Brief Description of The Project / Thesis with Salient Specifications:

Investigating a proof of concept for channel estimation and implementation on software defined platform. CSI will be fed back to the transmitter with limited resources.

Channel estimation is a major technique used in CDMA, LTE, WiMAX.

Scope of Work:

Major contender in 5th Generation (5G) standard implementation.

Academic Objectives:

Improving coding skills in C++, Linux and Matlab.

Study wireless communication using channel estimation and channel state information (CSI) techniques.

Understanding compressed sensing Improving skills in Digital Communication

Application / End Goal Objectives:

Achievement of higher throughput using compressed sensing technique.

Previous Work Done on The Subject:

Extensive Research in Wireless Communication. Research in progress on compressed sensing.

Material Resources Required:

USRP

No of Students Required: 4 **Group Members:** Syed Muhammad Hamza (Syndicate leader) Ahmad Rasul Farhan Ali Zohair Hassan **Special Skills Required:** Linux Python GNU Radio

Appendix B:

Bibliography

[1] D. Tse and P.Viswanath, Fundamentals of Wireless Communications, Cambridge University Press, May 2005.

[2]https://repository.up.ac.za/bitstream/handle/2263/61643/Laue Demystifying 2017.pdf

[3]http://wireless.egr.uh.edu/Tina.pdf

[4]S. Weinstein and P. Ebert, ”Data Transmission by Frequency-Division Multiplexing Using the Discrete Fourier Transform”, IEEE Transactions on Communication Technology, vol. 19, no. 5, pp. 628-634, Oct. 1971

[5] GNU Radio Homepage, http://gnuradio.org/, Dec. 2012.

[6] https://www.ettus.com/

[7] T. M. Schmidl and D. C. Cox, Robust Frequency and Timing Synchronization for OFDM, IEEE Trans. Communications, vol. 45, no. 12, 1997

[8]https://www.sciencedirect.com/science/article/pii/S1063520308000638

[9] Goldsmith, “Wireless Communications, “Cambridge University Press, 2005

[10]A. Dowler, A. Nix and J.McGeehan,“Data-derived Iterative Channel Estimation with Channel
Tracking for a Mobile fourth generation wide area OFDM system,” IEEE Global Telecommu
nications Conference, vol.2, pp. 804 - 808, Dec. 2000

APPENDIX – C

Code of MyCosamp Block:

```
#ifdef HAVE_CONFIG_H

#include "config.h"

#endif

#include <gnuradio/io_signature.h>

#include "MyCoSaMP_impl.h"

#include<stdlib.h>

#include<stdio.h>

#include <armadillo>

using namespace arma;

using namespace std;

namespace gr {

  namespace MCos {

    MyCoSaMP::sptr

    MyCoSaMP::make(int lin,int lout,int nx)
    {
      return gnuradio::get_initial_sptr

        (new MyCoSaMP_impl(lin, lout, nx));
    }

    /*
     * The private constructor
     */
    MyCoSaMP_impl::MyCoSaMP_impl(int lin,int lout,int nx)

      : gr::block("MyCoSaMP",
```

```
          gr::io_signature::make(1,1, sizeof(cx_double)),

          gr::io_signature::make(1,1, sizeof(cx_double))),
     my_lin(lin),

     my_lout(lout),

     my_nx(nx)
    {
     set_relative_rate (1.0 * my_lout / my_lin);

     set_output_multiple (my_lout);
    }

    /*
     * Our virtual destructor.
     */
    MyCoSaMP_impl::~MyCoSaMP_impl()
    {
    }

    void
    MyCoSaMP_impl::forecast (int noutput_items, gr_vector_int
&ninput_items_required)
    {
     assert (noutput_items % d_M == 0);

     // find the number of input items required to generate the requested number of output
items

     int ninput_items =  my_lin * noutput_items / my_lout;

     // set all streams' input requirements
     unsigned ninputs = ninput_items_required.size ();

     for (unsigned i = 0; i < ninputs; i++)
     {

      ninput_items_required[i] = ninput_items;

     }

    }

    void
    MyCoSaMP_impl::CoSaMP1(cx_vec z, cx_mat Phi, int s, int max_iter, cx_vec& a)
```

```
{
 int  d;

 cx_mat v,y,tmp;

 uvec ix,Omega,T;

 double err ;

 int it, stop ;

 cx_vec b ;

 d = size(Phi, 1) ;

 err = pow(10, (-5)) ;

 a = zeros<cx_vec>(d) ;

 v = z ;

 it = 0 ;

 stop = 0 ;

 while (!stop)
 {
  cout<<"transpose is"<<trans(Phi)<<endl;

  y = trans(Phi)*v ;

  cout<<"y is"<<y<<endl;

  //Sort index used for location wise sorting

  ix=sort_index( abs(y),"descend" );

  //only using first 2*s entries of y where s is the sparsity of the vetor

  Omega = ix(span(0, (2*s-1))) ;

  ix=sort_index( abs(a),"descend" );

  //Union is implemented with the use of intersection as direct union function was not
available
```

```
uvec HH=intersect(Omega,ix(span(0, (s-1))));

uvec YY = join_cols(Omega,ix(span(0, (s-1))));

uvec ZZ=sort(YY);

int dd=(YY.n_elem-HH.n_elem);

uvec SS(dd);

int jj=0;

for (int ii=1; ii<YY.n_elem ; ii++)
{

 if (ZZ(ii) != ZZ(ii-1))

 {

  SS(jj)=ZZ(ii);

  jj++;

 }

}

SS((dd-1))=ZZ(0);

T=sort(SS);

b = zeros<cx_vec>(d) ;

//solve function is used for least square

b((T)) = solve(Phi.cols(T) , z);

ix=sort_index( abs(b),"descend" );

a = zeros<cx_vec>(d) ;

a(ix(span(0, (s-1)))) = b(ix(span(0, (s-1)))) ;
```

```
 v = z-Phi*a ;

 it = it+1 ;

 if ((it>max_iter || norm(v)<=err*norm(z)))
 {

  stop = 1 ;

 }

}

}

int
MyCoSaMP_impl::general_work (int noutput_items,
                    gr_vector_int &ninput_items,
                    gr_vector_const_void_star &input_items,
                    gr_vector_void_star &output_items)
{
 const cx_double *in = (const cx_double *) input_items[0];

 cx_double *out = (cx_double *) output_items[0];

 cx_vec fd;

 fd = zeros<cx_vec>(my_lin);

 cx_vec x;

 x = zeros<cx_vec>(my_lout);

 for(int i=0;i<my_lin;i++)

 fd(i)=in[i];

 int k;

 k = my_nx;

 cx_mat xs;

 int my_length=my_lout;
```

```
xs = zeros<cx_vec>(my_length);

uvec p;

int SZ=my_length;

uword perm[SZ];

for (int i = 0; i < SZ; i++)
{

 perm[i] = i;

}

// Random permutation the order

for (int i = 0; i < SZ; i++)
{

 int j, t;

 j = rand() % (SZ-i) + i;

 t = perm[j];

 perm[j] = perm[i];

 perm[i] = t; // Swap i and j

}

p=zeros<uvec>(my_length);

for(int i=0;i<SZ;i++)
{

 p(i)=perm[i];

}

//Using seed for generating random vector

arma_rng::set_seed(8);
```

```
xs(p(span(0,  (k-1)))) = 1*1.0/sqrt(2*k)*(randn(k, 1)+cx_double(0, 1)*randn(k, 1));

//For normalization of the vector

xs = xs*diagmat(1/(sqrt(arma::sum(xs%arma::conj(xs))))) ;

cx_vec s=zeros<cx_vec>(my_lout);

s=xs;

cx_mat E;

int M=my_lin,N=my_lout;

arma_rng::set_seed(8);

E = 1*1.0/sqrt(2*M)*(randn(M, N)+cx_double(0, 1)*randn(M, N)) ;

int das=6*(my_nx+1);

CoSaMP1(fd, E, my_nx,das,x);

//Comparing two vetors by taking the norm

cout<<"norm is"<<'\t'<<norm(x-s)<<endl;

for(int i=0;i<my_lout;i++)
{

 out[i]=x(i);

}

 return my_lout;
}

  } /* namespace MCos */

} /* namespace gr */
```

Code of MyMatrix Block:

```
#ifdef HAVE_CONFIG_H

#include "config.h"

#endif

#include <gnuradio/io_signature.h>

#include "MyMatrix_impl.h"

#include<stdlib.h>

#include<stdio.h>

#include <armadillo>

using namespace arma;

using namespace std;

namespace gr {

  namespace matrix {

    MyMatrix::sptr

    MyMatrix::make(int lin,int lout)

    {

      return gnuradio::get_initial_sptr

        (new MyMatrix_impl(lin, lout));
    }

    /*
     * The private constructor
```

```
 */
MyMatrix_impl::MyMatrix_impl(int lin,int lout)

  : gr::block("MyMatrix",

          gr::io_signature::make(1,1, sizeof(cx_double)),

          gr::io_signature::make(1,1, sizeof(cx_double))),

  my_lin(lin),

  my_lout(lout)

{

  set_relative_rate (1.0 * my_lout/ my_lin);

  set_output_multiple (my_lout);

}

/*
 * Our virtual destructor.
 */
MyMatrix_impl::~MyMatrix_impl()
{
}

void
MyMatrix_impl::forecast (int noutput_items, gr_vector_int &ninput_items_required)
{

 int ninput_items =  my_lin * noutput_items / my_lout;

 unsigned ninputs = ninput_items_required.size ();

 for (unsigned i = 0; i < ninputs; i++)

 ninput_items_required[i] = ninput_items;
```

```
}

int
MyMatrix_impl::general_work (int noutput_items,
                   gr_vector_int &ninput_items,
                   gr_vector_const_void_star &input_items,
                   gr_vector_void_star &output_items)
{

 const cx_double *in = (const cx_double *) input_items[0];

 cx_double *out = (cx_double *) output_items[0];

 cx_mat D;

 int M=my_lout,N=my_lin;

 arma_rng::set_seed(8);

 D = 1*1.0/sqrt(2*M)*(randn(M, N)+cx_double(0, 1)*randn(M, N)) ;

 cx_vec ve=zeros<cx_vec>(my_lin);

 for(int i=0;i<my_lin;i++)

 ve(i)=in[i];

 cx_vec am;

 am = zeros<cx_vec>(my_lout);

 am=D*ve;

 for(int i=0;i<my_lout;i++)
 {

  out[i]=am(i);

 }

  return my_lout;
```

Code of FypSource Block:

```
#ifdef HAVE_CONFIG_H

#include "config.h"

#endif

#include <gnuradio/io_signature.h>

#include "FypSource_impl.h"

#include<stdlib.h>

#include<stdio.h>

#include <armadillo>

using namespace arma;

using namespace std;

namespace gr {

  namespace Vec {

    FypSource::sptr

    FypSource::make(int length,int nz)
    {
      return gnuradio::get_initial_sptr

        (new FypSource_impl(length, nz));

    }

    /*
     * The private constructor
```

```
 */
FypSource_impl::FypSource_impl(int length,int nz)

  : gr::sync_block("FypSource",

          gr::io_signature::make(0, 0, 0),

          gr::io_signature::make(1,1, sizeof(cx_double))),

 my_length(length),

 my_nz(nz)
{
}

/*
 * Our virtual destructor.
 */
FypSource_impl::~FypSource_impl()
{
}

int
FypSource_impl::work(int noutput_items,
    gr_vector_const_void_star &input_items,
    gr_vector_void_star &output_items)

{

 cx_double *out = (cx_double *) output_items[0];

 int k;

 k = my_nz;

 cx_mat xs;

 xs = zeros<cx_vec>(my_length);

 uvec p;
```

```
int SZ=my_length;

uword perm[SZ];

for (int i = 0; i < SZ; i++)
{

 perm[i] = i;

}


// Random permutation the order
for (int i = 0; i < SZ; i++)
{

 int j, t;

 j = rand() % (SZ-i) + i;

 t = perm[j]; perm[j] = perm[i]; perm[i] = t; // Swap i and j

}

p=zeros<uvec>(my_length);

for(int i=0;i<SZ;i++)
{

 p(i)=perm[i];

}

//Using seed for generating random vector
arma_rng::set_seed(8);

xs(p(span(0, (k-1)))) = 1*1.0/sqrt(2*k)*(randn(k, 1)+cx_double(0,
1)*randn(k, 1));

//For normalization of the vector
```

```
    xs = xs*diagmat(1/(sqrt(arma::sum(xs%arma::conj(xs))))) ;

    cx_vec s=zeros<cx_vec>(my_length);

    s=xs;

    for(int i=0;i<my_length;i++)
    {

     out[i]=s[i];

    }


    return my_length;
   }

  } /* namespace Vec */
} /* namespace gr */
```

Code of MyConvertor:

/**********************MyConvertor used for double to Float Conversion***************************/**

```
#ifdef HAVE_CONFIG_H

#include "config.h"

#endif

#include <gnuradio/io_signature.h>

#include "MyConverter_impl.h"

#include<stdlib.h>

#include<stdio.h>
```

```
#include <armadillo>

using namespace arma;

using namespace std;

namespace gr {

  namespace cover {

    MyConverter::sptr

    MyConverter::make(int length)
    {

     return gnuradio::get_initial_sptr

       (new MyConverter_impl(length));

    }

    /*
     * The private constructor
     */
    MyConverter_impl::MyConverter_impl(int length)

      : gr::block("MyConverter",

              gr::io_signature::make(1,1, sizeof(cx_double)),

              gr::io_signature::make(1,1, sizeof(cx_float))),

     my_length(length)
    {
    }

    /*
     * Our virtual destructor.
     */
    MyConverter_impl::~MyConverter_impl()
    {
```

```
    }

    void
    MyConverter_impl::forecast (int noutput_items, gr_vector_int
&ninput_items_required)
    {

      ninput_items_required[0] = noutput_items ;

    }

    int
    MyConverter_impl::general_work (int noutput_items,
                       gr_vector_int &ninput_items,
                       gr_vector_const_void_star &input_items,
                       gr_vector_void_star &output_items)
    {

     const cx_double *in = (const cx_double *) input_items[0];

     cx_float *out = (cx_float *) output_items[0];

     for(int i=0;i<my_length;i++)
     {

      out[i]=in[i];

     }

     return my_length;
    }

  } /* namespace cover */
} /* namespace gr */
```

Code of Out Convertor Block

/***
Out Convertor used for Float to double conversion**/

```
#ifdef HAVE_CONFIG_H

#include "config.h"

#endif

#include <gnuradio/io_signature.h>

#include "OutConvertor_impl.h"

#include<stdlib.h>

#include<stdio.h>

#include <armadillo>

using namespace arma;

using namespace std;

namespace gr {

  namespace cover1 {

    OutConvertor::sptr

    OutConvertor::make(int length)
    {
      return gnuradio::get_initial_sptr

        (new OutConvertor_impl(length));
    }

    /*
     * The private constructor
     */
    OutConvertor_impl::OutConvertor_impl(int length)

      : gr::block("OutConvertor",

              gr::io_signature::make(1,1, sizeof(cx_float)),
```

```
          gr::io_signature::make(1,1, sizeof(cx_double))),

    my_length(length)
  {

  }

  /*
   * Our virtual destructor.
   */
  OutConvertor_impl::~OutConvertor_impl()
  {
  }

  void
  OutConvertor_impl::forecast (int noutput_items, gr_vector_int
&ninput_items_required)
  {

   int ninput_items =   noutput_items ;

   unsigned ninputs = ninput_items_required.size ();

   for (unsigned i = 0; i < ninputs; i++)
   {

     ninput_items_required[i] = ninput_items;

   }

  int
  OutConvertor_impl::general_work (int noutput_items,
                  gr_vector_int &ninput_items,
                  gr_vector_const_void_star &input_items,
                  gr_vector_void_star &output_items)
  {
    const cx_float *in = (const cx_float *) input_items[0];

    cx_double *out = (cx_double *) output_items[0];


    for(int i=0;i<my_length;i++)
    {

     out[i]=in[i];
```

```
      }


     return my_length;
    }

  } /* namespace cover1 */
} /* namespace gr */
```

C++ code of CoSaMP Algorithm

```
#include<iostream>

#include "mconvert.h"

#include <cstdio>

#include<stdio.h>

#include <armadillo>

using namespace arma ;

void CoSaMP(cx_vec z, cx_mat Phi, int s, int max_iter, cx_vec& a)
{

 int  d;

 cx_vec v,y,tmp;

 uvec ix,Omega,T;

 double err ;

 int it, stop ;

 cx_vec b ;

 d = size(Phi, 1) ;
```

```
err = pow(10, (-3)) ;

a = arma::zeros<cx_vec>(d) ;

v = z ;

it = 0 ;

stop = 0 ;

 while (!stop)
 {

  //Taking transpose of the matrix and multiplying it with the compressed vector to form proxy of the vector

  y = arma::trans(Phi)*v ;

  cout<<"y is"<<y<<endl;

  //Sort index used for location wise sorting

  ix=sort_index( abs(y),"descend" );

  cout<<"ix1 is"<<ix<<endl;

  //only using first 2*s entries of y where s is the sparsity of the vetor

  Omega = ix(span(0, (2*s-1))) ;

  cout<<"Omega is"<<Omega<<endl;

  ix=sort_index( abs(a),"descend" );

  cout<<"ix2 is"<<ix<<endl;

  //Union is implemented with the use of intersection as direct union function was not available

  uvec HH=intersect(Omega,ix(span(0, (s-1))));

  //join_cols used for combining two vectors

  uvec YY = join_cols(Omega,ix(span(0, (s-1))));
```

```
uvec ZZ=sort(YY);

int dd=(YY.n_elem-HH.n_elem);

uvec SS(dd);

int jj=0;

for (int ii=1; ii<YY.n_elem ; ii++)
{

 if (ZZ(ii) != ZZ(ii-1))

 {

  SS(jj)=ZZ(ii);

  jj++;

 }

}

SS((dd-1))=ZZ(0);

T=sort(SS);

cout<<"union is"<<T;

b = arma::zeros<cx_vec>(d) ;

//solve function is used for least square solution with only Selected columns of the matrix and the compressed vector

b((T)) = arma::solve(Phi.cols(T) , z);

cout<<"Selected Colums are"<<Phi.cols(T)<<endl;

cout<<"Vector after applying solve function is"<<b(T)<<endl;

ix=sort_index( abs(b),"descend" );

cout<<"ix3 is"<<ix<<endl;
```

```
  a = arma::zeros<cx_vec>(d) ;

  a(ix(span(0, (s-1)))) = b(ix(span(0, (s-1)))) ;

  cout<<"Recovered vector uptil this iteration is"<<a<<endl;

  v = z-Phi*a ;

  it = it+1 ;

  if ((it>max_iter || arma::norm(v)<=err*norm(z)))
  {

   stop = 1 ;

  }

 }

}

int main(int argc, char** argv)

{

 cx_mat A;

 cx_vec b ,x;

 uvec p;

 int k, M, N ;

 cx_mat xs ;

 M =120;

 N = 250;

 arma_rng::set_seed(8);

 A = 1*1.0/sqrt(2*M)*(randn(M, N)+cx_double(0, 1)*randn(M, N)) ;
```

```
A = A*diagmat(1/(sqrt(arma::sum(A%arma::conj(A))))) ;

k =50;

xs = arma::zeros<cx_vec>(N) ;

int i;

int SZ=N;

uword perm[SZ];

for (int i = 0; i < SZ; i++)
{

 perm[i] = i;

}

// Random permutation the order

for (int i = 0; i < SZ; i++)
{

 int j, t;

 j = rand() % (SZ-i) + i;

 t = perm[j];

 perm[j] = perm[i];

 perm[i] = t; // Swap i and j

 }

 p=zeros<uvec>(my_length);

 for(i=0;i<SZ;i++)
 {

  p(i)=perm[i];

 }
```

```
  //Using seed for generating random vector

  arma_rng::set_seed(8);

  xs(p(span(0,  (k-1)))) = 1*1.0/sqrt(2*k)*(randn(k, 1)+cx_double(0, 1)*randn(k, 1));

  //For normalization of the vector

  xs = xs*diagmat(1/(sqrt(arma::sum(xs%arma::conj(xs))))) ;

  cout<<"Norm of xs is"<<'\t'<<norm(xs)<<endl;

  b = A*xs ;

  cout<<" b is"<<b<<endl;

  cout<<" A is"<<A<<endl;

  cout<<"p is"<<p<<endl;

  cout<<"xs is"<<xs<<endl;

  int fa=6*(k+1);

  CoSaMP(b, A, k,fa,x);

  cout<<"Reconstructed x is"<<x<<endl;

  cout<<"Norm of (x-xs) is"<<'\t'<<norm(x-xs)<<endl;

  return 0 ;
}
```