Master's Thesis Defense

Implementation of Compressive Sampling for Wireless Sensor Network Applications

Committee Members:
Dr. Xinrong Li
Dr. Kamesh Namuduri
Dr. Tao Yang

Nathan Ruprecht

Overview

- Background and Motivation
- Terminology
- Compressive Sampling Theory
  - Compression and Reconstruction
  - Measurement Matrix Properties
  - Minimum Representation
- Implementation
  - Matlab
  - MCU – MSP432
  - SBC – Raspberry Pi 3
- Results
- Conclusion
- Further Research
- Questions

Background

- ADCs are limiting advancement in RF tech
- Harry Nyquist (1928) and Claude Shannon(1948) Sampling Theorem
- Difficulties:
  - UWB communications 3.1GHz – 10.6GHz
  - EHF 30 – 300GHz for satellite-based sensing, weapons systems, radar, etc.

Motivation

- 1970s – Seismologist use CS
- 2004 – Emmanuel Candes and David Donoho
- Applications:
  - Imaging – MRI, Radar, Satellite (astronomy)
  - Communications – Cognitive Radio
- Big Picture – Analog-to-Information Conversion (AIC)
- Focus of Thesis – Application

Terminology

- xo: the original signal of various length
- x: an N-sized frame of xo
- y: an M-sized observation/representation of x
- A: measurement matrix consisting of Φ and Ψ
- Φ: Phi – distribution matrix or randomizer
- Ψ: Psi – transformation matrix or sparsifier
- RMS: Root Mean Squared Error

CS Theory – Compress

$$y = Ax$$

$$y \in \mathbb{R}^M, \qquad \mathrm{A} \in \mathbb{R}^{M \times N}, \qquad x \in \mathbb{R}^N$$

$$M \ll N$$

$$A = \Phi\Psi$$

Φ is a distribution matrix (randomizer)

Ψ is a transformation matrix (sparsifier)

$$y = \Phi\Psi x$$

$$\Phi \in \mathbb{R}^{M \times N}, \qquad \Psi \in \mathbb{R}^{N \times N}$$

CS Theory – Reconstruct

Goal: Solve $y = A\hat{x}$ using either LP or SOCP

minimum l_p norm

$$\min \|\hat{x}\|_p \text{ subject to } A\hat{x} = y$$

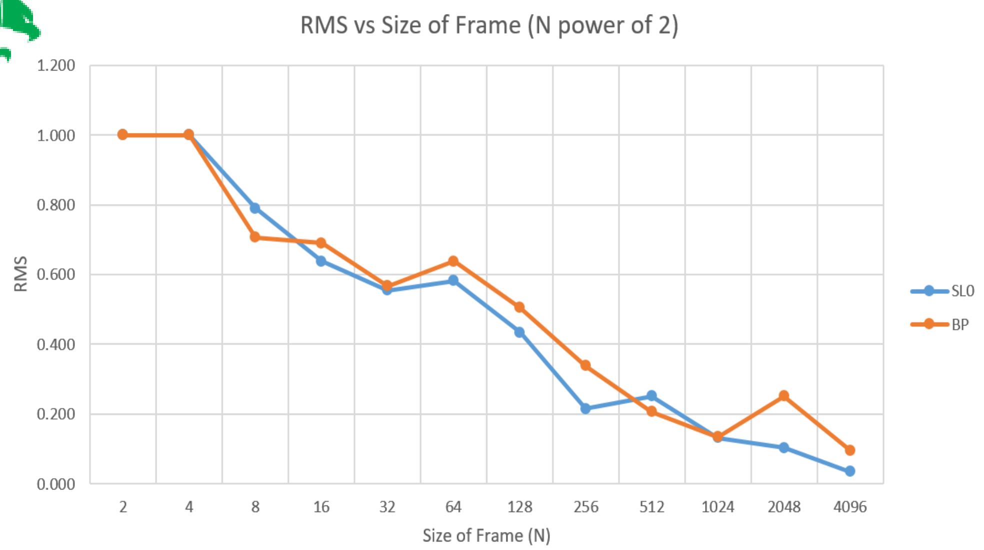
l_2 norm is $\hat{x} = A^{-1}y$

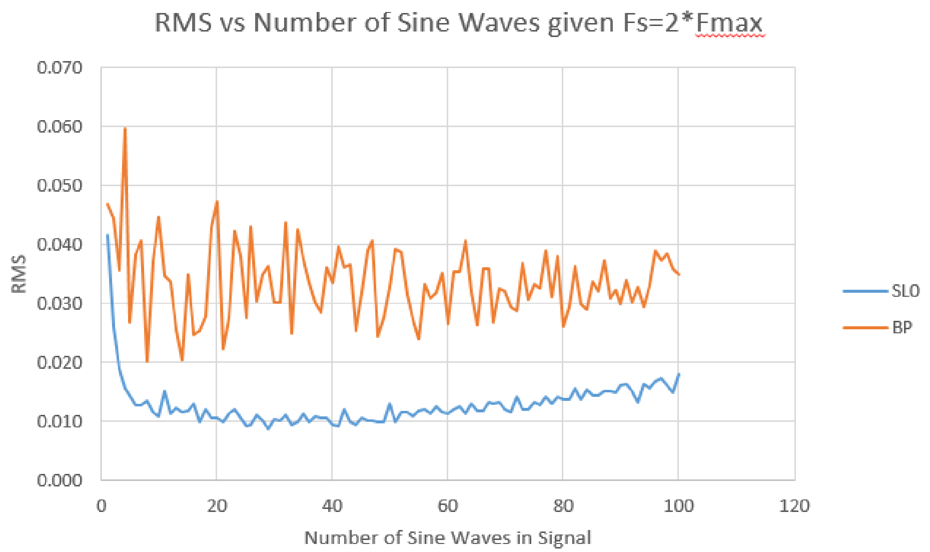
Basic Pursuit: for large SOEs, l_1 norm $=$ l_0 norm [11]

l_0 norm is a problem – too sensitive to noise

Other techniques compared to BP – OMP, ROMP, CoSaMP, SL0

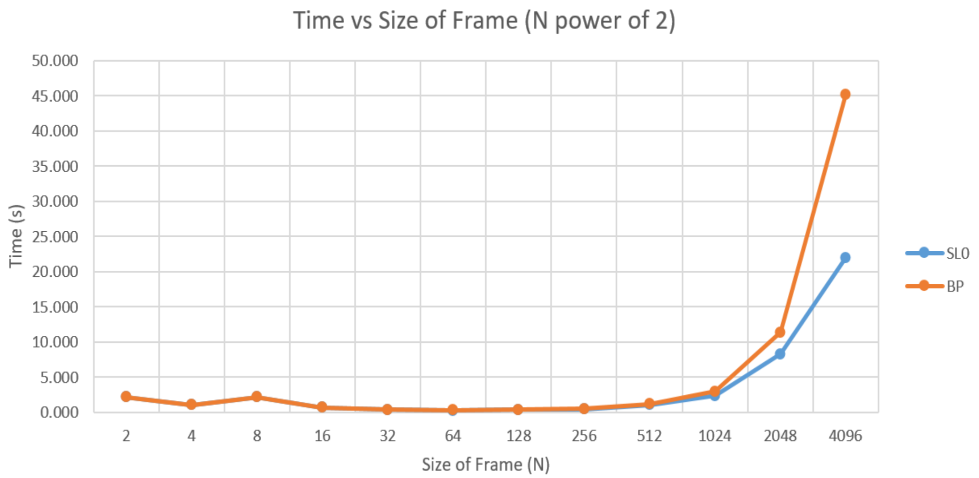
CS Theory – Reconstruct

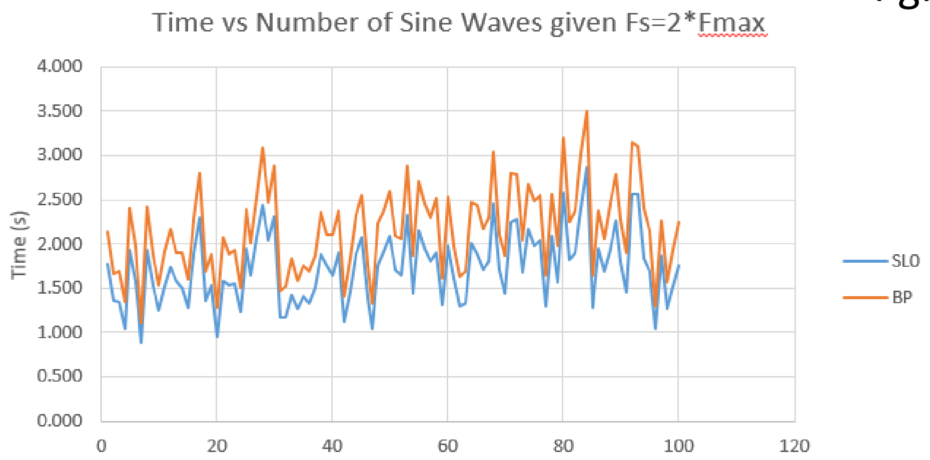




Pg. 13

Pg. 12





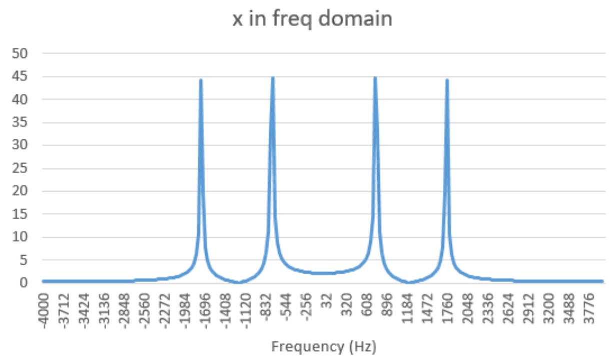
CS Theory – Measurement Matrix – Ψ

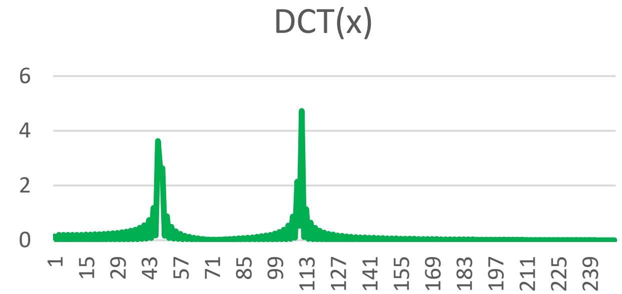
- Goal: Create K-sparse signal

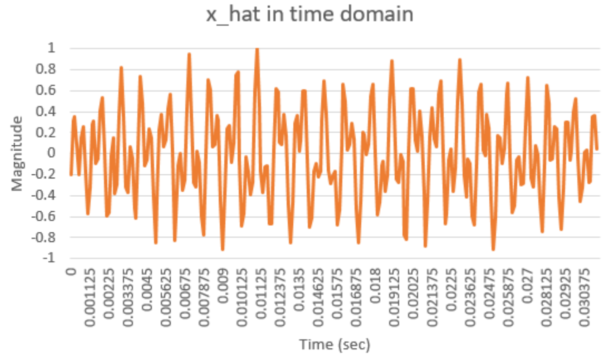
Pg. 7

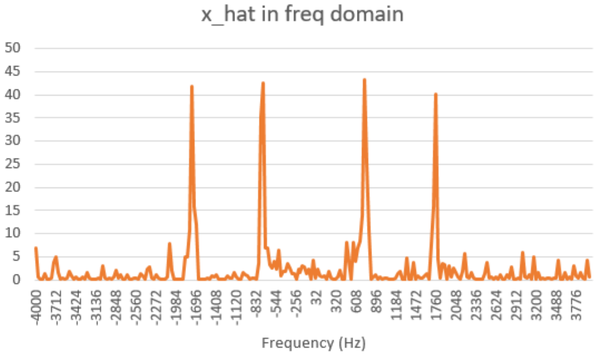
- $x(t) = \sin(\,1500\pi t\,) + \sin(\,3500\pi t\,)$
- Signal representable by a few, nonzero elements
- How we do it:
    - Detecting noise floor
    - Find K-sparsity of frame

Pg. 9









CS Theory – Measurement Matrix – Φ

- Goal: Randomize signal to create unique solution when reconstructing

$$\mu(\Phi, \Psi) = \max_{1 \le i,j \le N} \left| \langle \Phi_i , \Psi_j \rangle \right|$$

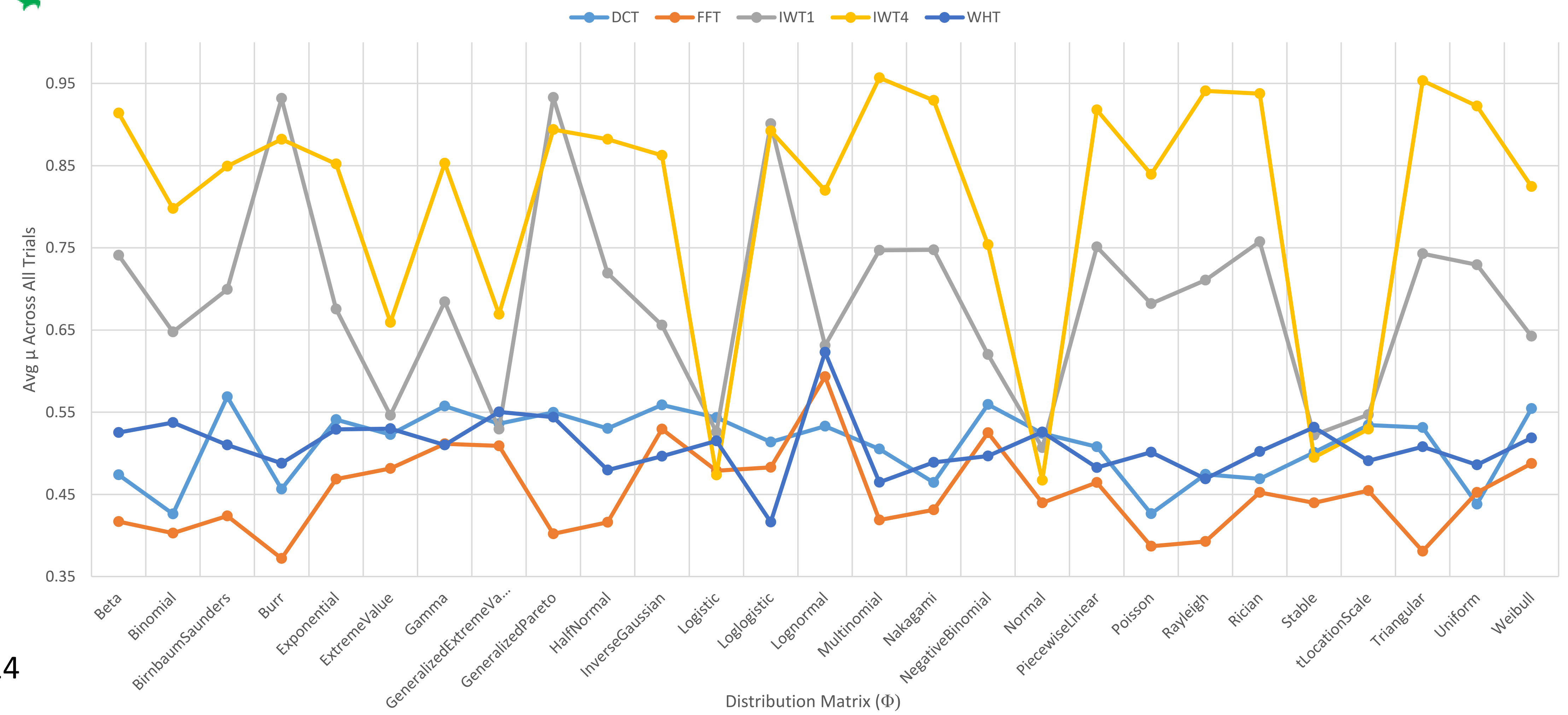
- Pass/fail RIP [19] almost always pass for random distributions, more so used for deterministic
- Matrix coherence coefficient, $1/\sqrt{N} \le \mu \le 1$, to actually characterize performance of $A = \Phi\Psi$

$$\mu(A) = \max_{1 \le i,j \le N} \frac{\left| \langle A_i , A_j \rangle \right|}{\|A_i\| \|A_j\|}$$

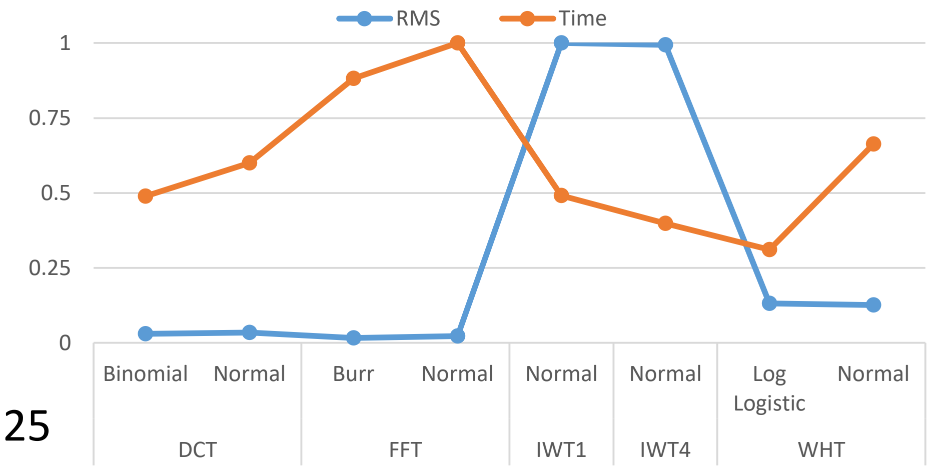
Pg. 22

```
for i=1:N
    for j=1:N
        temp = abs( dot(A(i,:),A(:,j)) );
        temp = temp./[norm( A(i,:) ).*norm( A(:,j) )];
        mu = max( mu, temp );
    end
end
```

CS Theory – Measurement Matrix – ΦΨ



CS Theory – Measurement Matrix – ΦΨ



Pg. 25

Ψ	Φ	RMS	Time
DCT	Binomial	0.007	3.126
	Normal	0.008	3.837
FFT	Burr	0.004	5.637
	Normal	0.005	6.394
IWT1	Normal	0.235	3.138
IWT4	Normal	0.233	2.548
WHT	Log Logistic	0.031	1.987
	Normal	0.030	4.240

Ψ	Φ	Overall Rank (Out of 135)
DCT	Binomial	12
	Poisson	13
	Uniform	15
	Normal	61
FFT	Burr	1
	Triangular	2
	Poisson	3
	Normal	17
IWT1	Normal	49
	Stable	59
	Logistic	64
IWT4	Normal	25
	Logistic	29
	Stable	42
WHT	Log Logistic	8
	Multinomial	24
	Rayleigh	27
	Normal	65

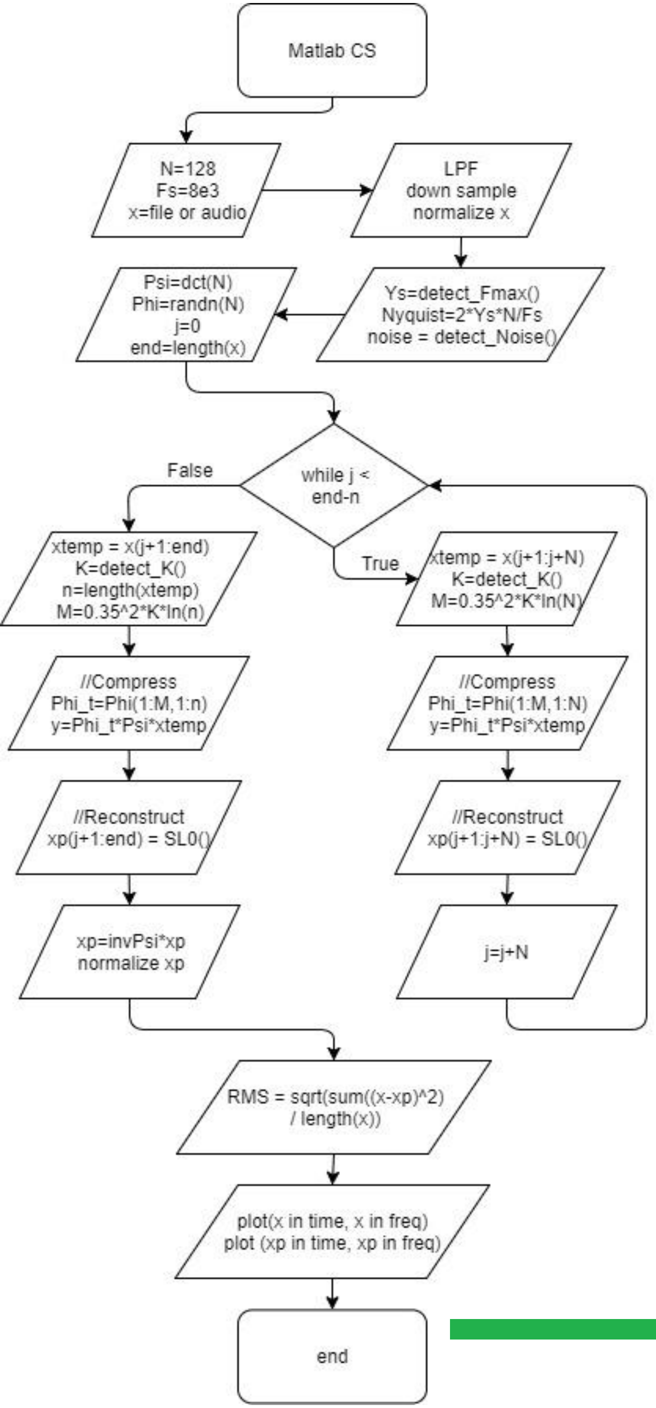
CS Theory – Minimum Representation

- Goal: Represent x with as few observations (y) as possible. Compression Ratio (CR)

$$M = O(K * logN)$$
[12]

- Equation works for $N \rightarrow \infty$, how about powers of 2?
- Better specify an equation for M that depends on sparsity, N sized frame that is "small", and μ
- Absolute minimum – subjective judging would argue that this is too aggressive

$$M = \mu^2 K \ln N$$
[14]



Matlab Implementation

- Retrieve entire signal (not real time)
- Break it into N-sized frames
- Compress and Reconstruct
- RMS error then plot x and $\hat{x}$

Pg. 39

		Average Mu
DCT	Normal	0.3589
	Bernoulli	0.3488
FFT	Normal	0.2776
	Bernoulli	0.2728

MCU Implementation

```
MAP_ADC14_toggleConversionTrigger();
while(!ADC15_isBusy()){}
triple_buffer[input_index][sample_index] =
ADC14_getResult(ADC_MEM0);

//Update buffer indices and sample index
if(++sample_index >= N) {
    sample_index = 0;
    if(data_ready){buffer_overrun = true;}

    tmp_ui32 = input_index;
    input_index = data_index;
    data_index = output_index;
    output_index = tmp_ui32;
    data_ready = true;
}
```

- What we did…
- Failed for memory space, and processing speed

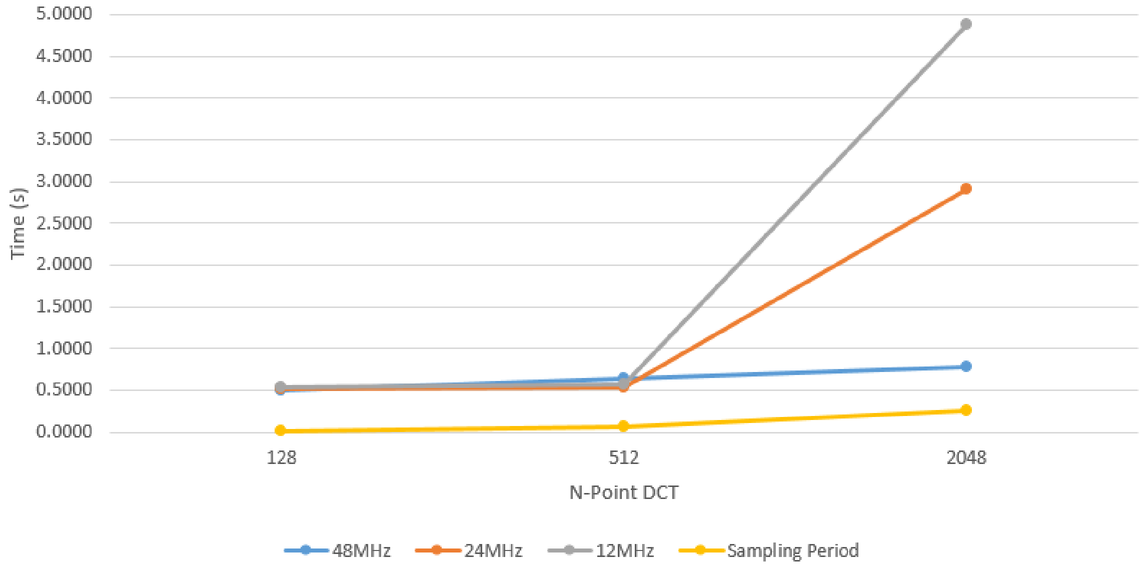
Pg. 46&48

```
while(1) {
    MAP_PCM_gotoLPM0();
    if( buffer_overrun ) {
        UARTprintf( EUSCI_A0_BASE, “Error: buffer
overrun\r\n”);
        buffer_overrun = false;
    }
    if( data_ready ) {
        x.pData = triple_buffer[data_index];
        arm_dct4_f32( &S, (float32_t *)pState,
           x.pData);
        K = detect_K(x.pData);
        M = pow(0.35, 2)*log(N)*K;
        for( i=0; i<=M/rows; i++ ) {
            if( i==0 ) {
                Phi.pData = A0;
            } else if( i==rows ) {
                Phi.pData = A1;
            } else if( i==n*rows ) {
                Phi.pData = An;
            }
        arm_mat_mult_f32( &Phi, &x, &y );
        y_f32[0] = *y.pData
}}}
```

MCU Implementation

- DCT only available at certain N. Vary FCLK and N to compare sampling period (time for input buffer) against processing time (time to beat sampling)

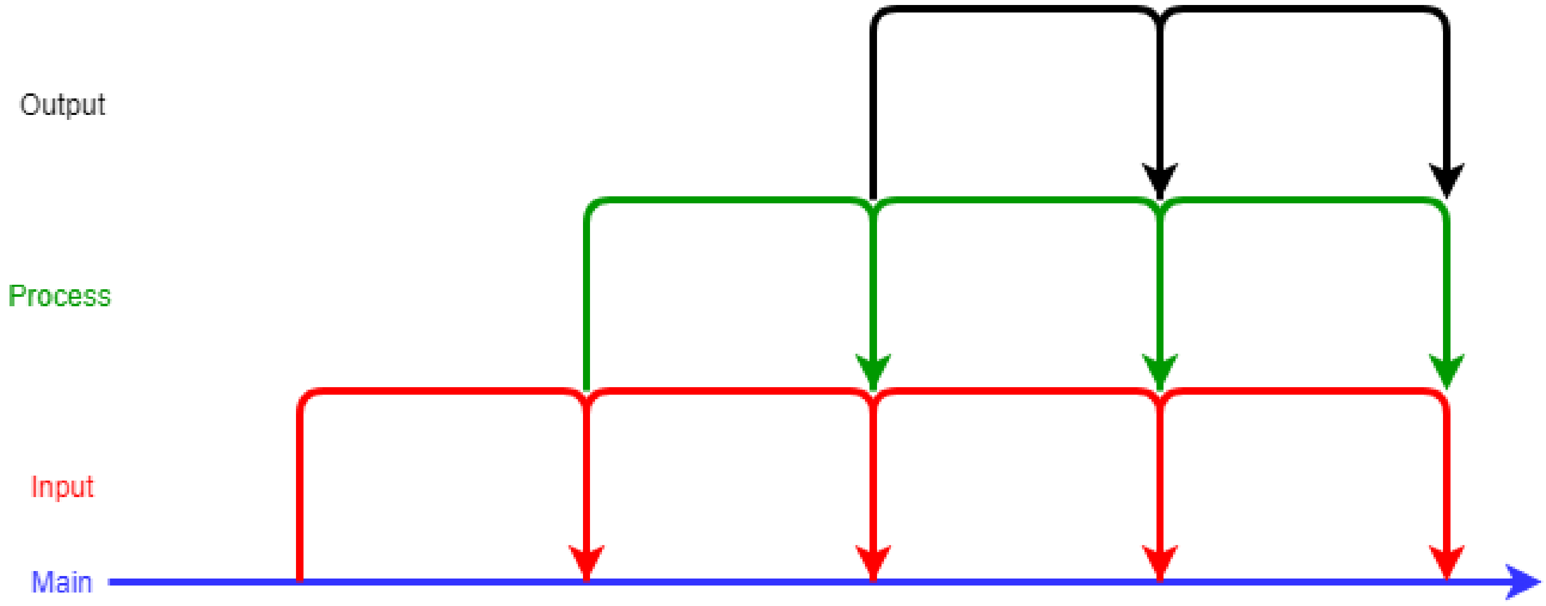
Pg. 50



	N	128	512	2048
Clock Frequency (s)	Sampling Period	0.0160	0.0640	0.2560
48MHz	Processing Time	0.4983	0.6398	0.7838
24MHz	Processing Time	0.5224	0.5447	2.9056
12MHz	Processing Time	0.5380	0.5795	4.8783

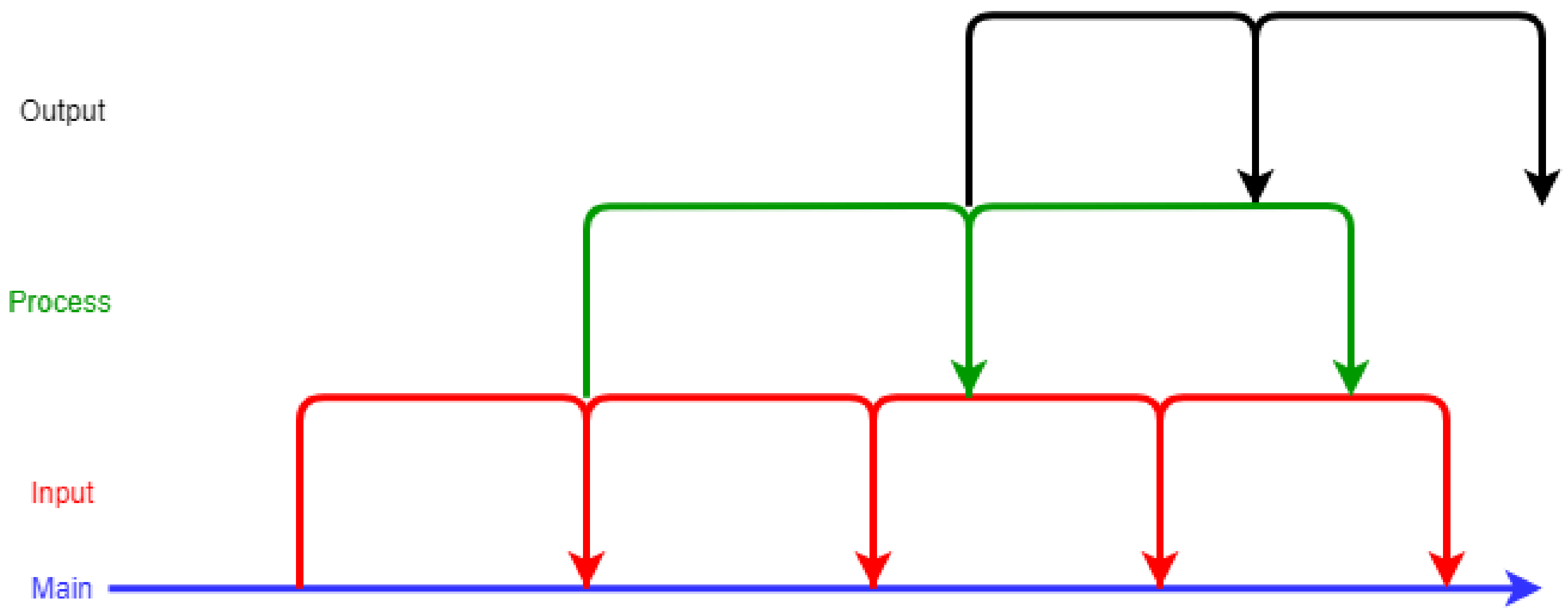
SBC Implementation

Expectation



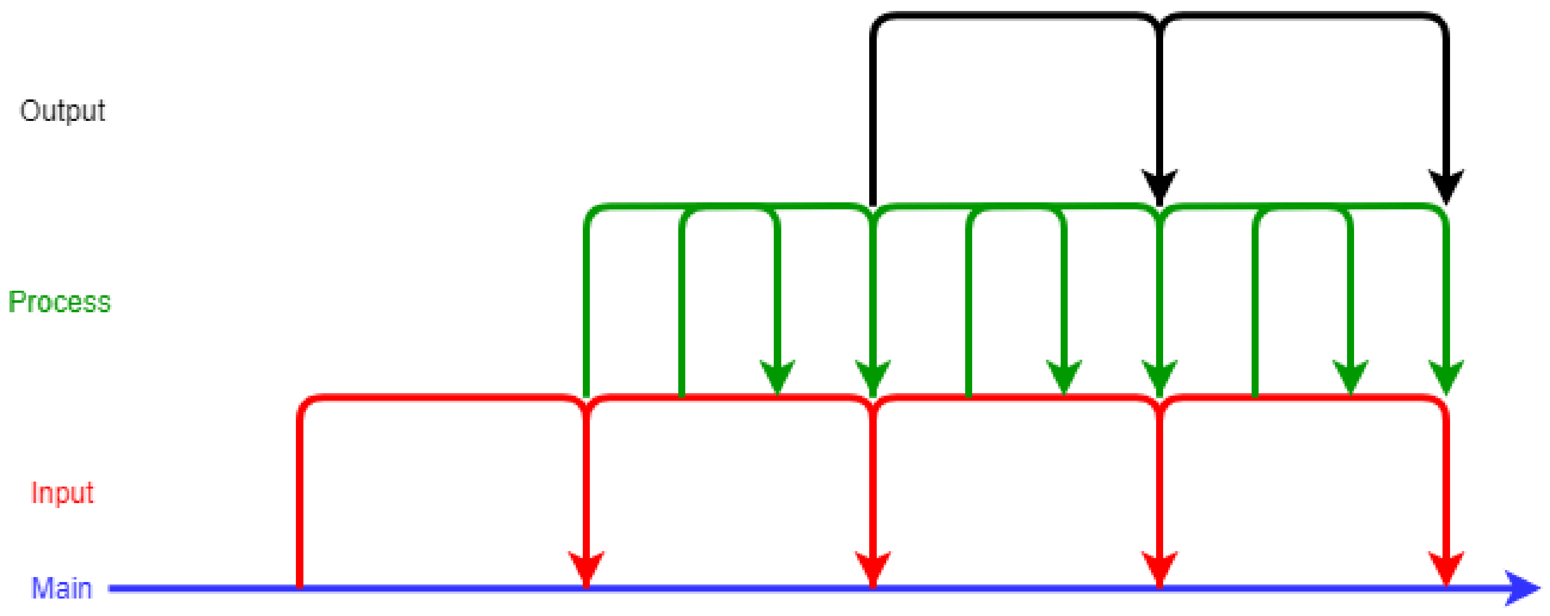
SBC Implementation

Reality – Buffer Overrun!



SBC Implementation

Fixed – More Process Data Buffers/Threads



SBC Implementation

Pg. 67-69

```
class data_buffer:
    def __init__(self, buff, M, start, end):
        self.buff = buff
        self.M = M
        self.start = start
        self.end = end
```

```
t=[]
db=[]
for i in range(NUM_DB):
    db=np.append(db, data_buffer([], M, i*N, (i+1)*N))
    t=np.append(t, threading.Thread(target=processData,
           args=(db[i],), daemon=True))
tout = threading.Thread(target=outputData, daemon=True)
tstop = threading.Thread(target=stopData, daemon=True)
```

```
stream = pa.open(format=FORMAT,
           channels=CHANNELS,
           rate=Fo, input=True,
           frames_per_buffer = SAMPLES,
           stream_callback=inputData)
stream.start_stream()
for i in range(NUM_DB):
t[i].start()
tout.start()
tstop.start()
```

```
while not stop:
    for i in range(NUM_DB):
        if not t[i].isAlive():
            t[i] = threading.Thread(target=processData,
                      args=(db[i],), daemon=True)
            t[i].start()
        if not tout.isAlive():
            tout = threading.Thread(target=outputData,
                      daemon=True)
            tout.start()
```

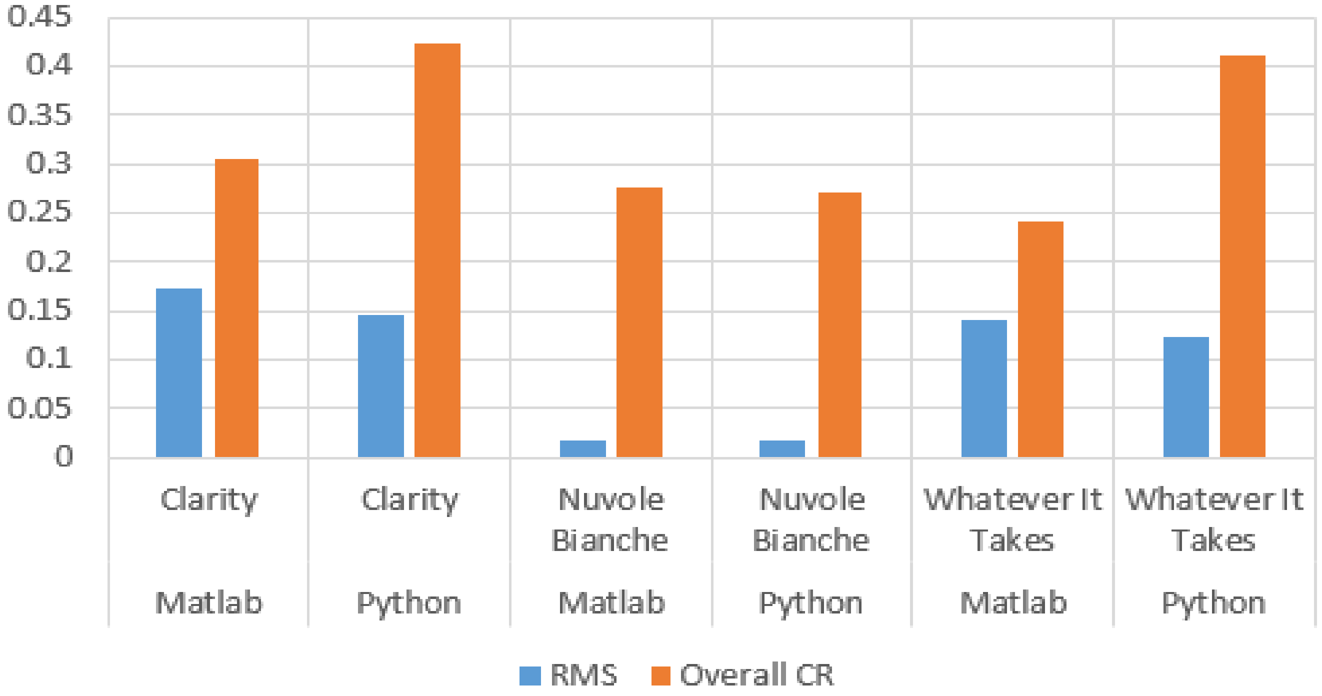
SBC Implementation

- Still detecting noise, K sparsity, and M in each processing thread
- On Windows, user input to start and stop recording. Pi goes on timer.
- Multithreading vs Multiprocessing

Results

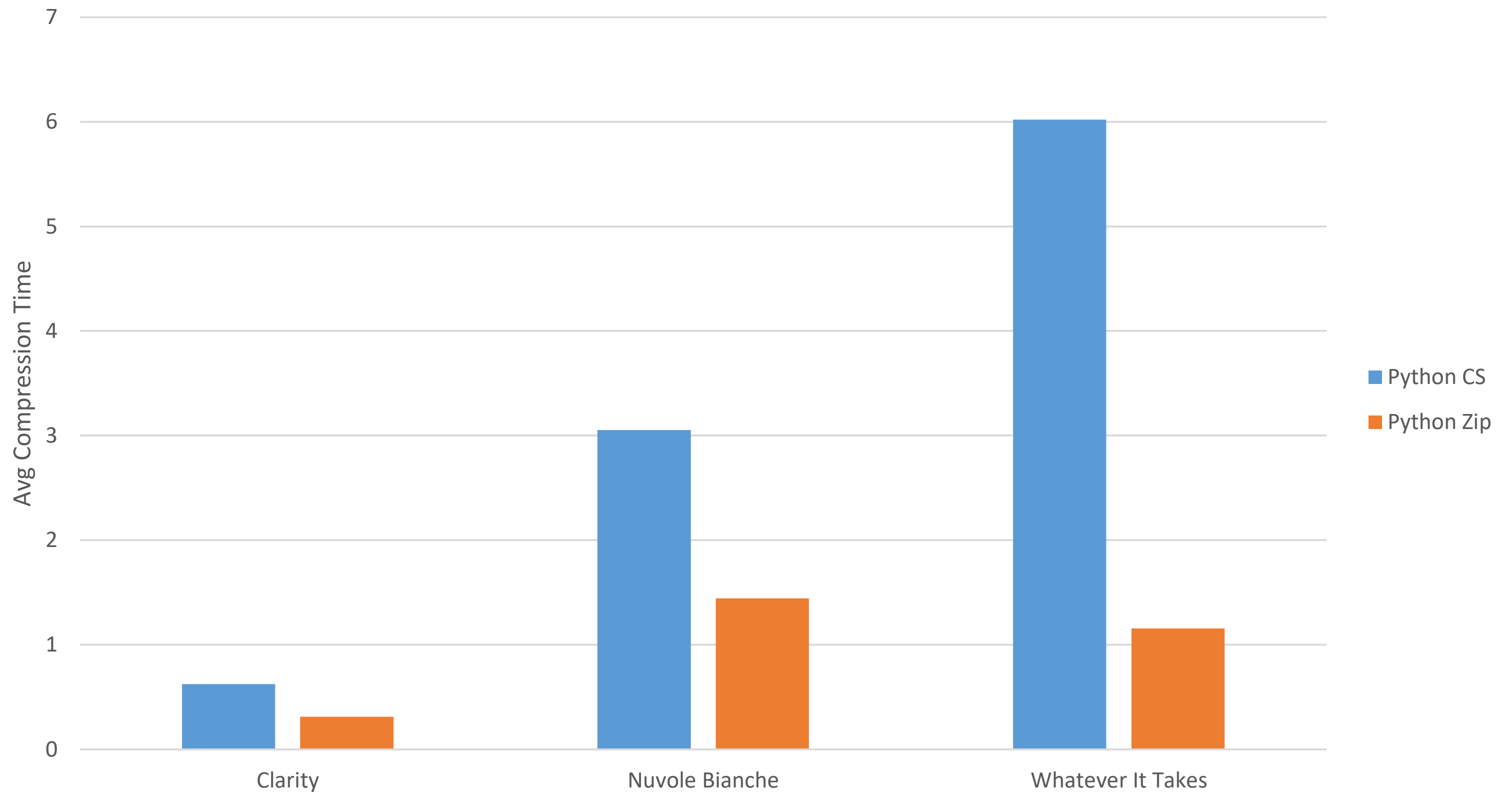
Comparing Python to Matlab

- Python less error, but not as compressed
- Implementation matching simulation



Initial comparison – CS vs Zip

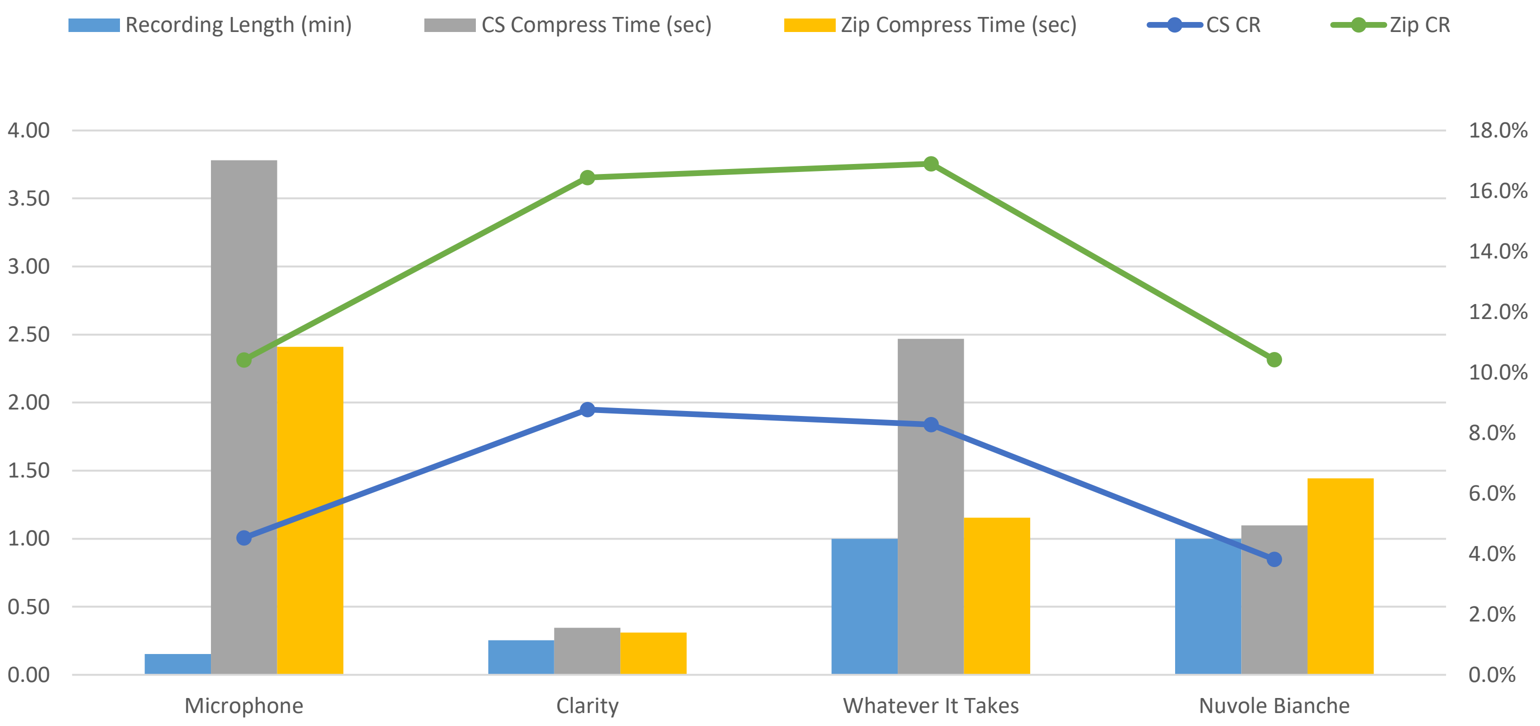
- Zip is faster on average



Results

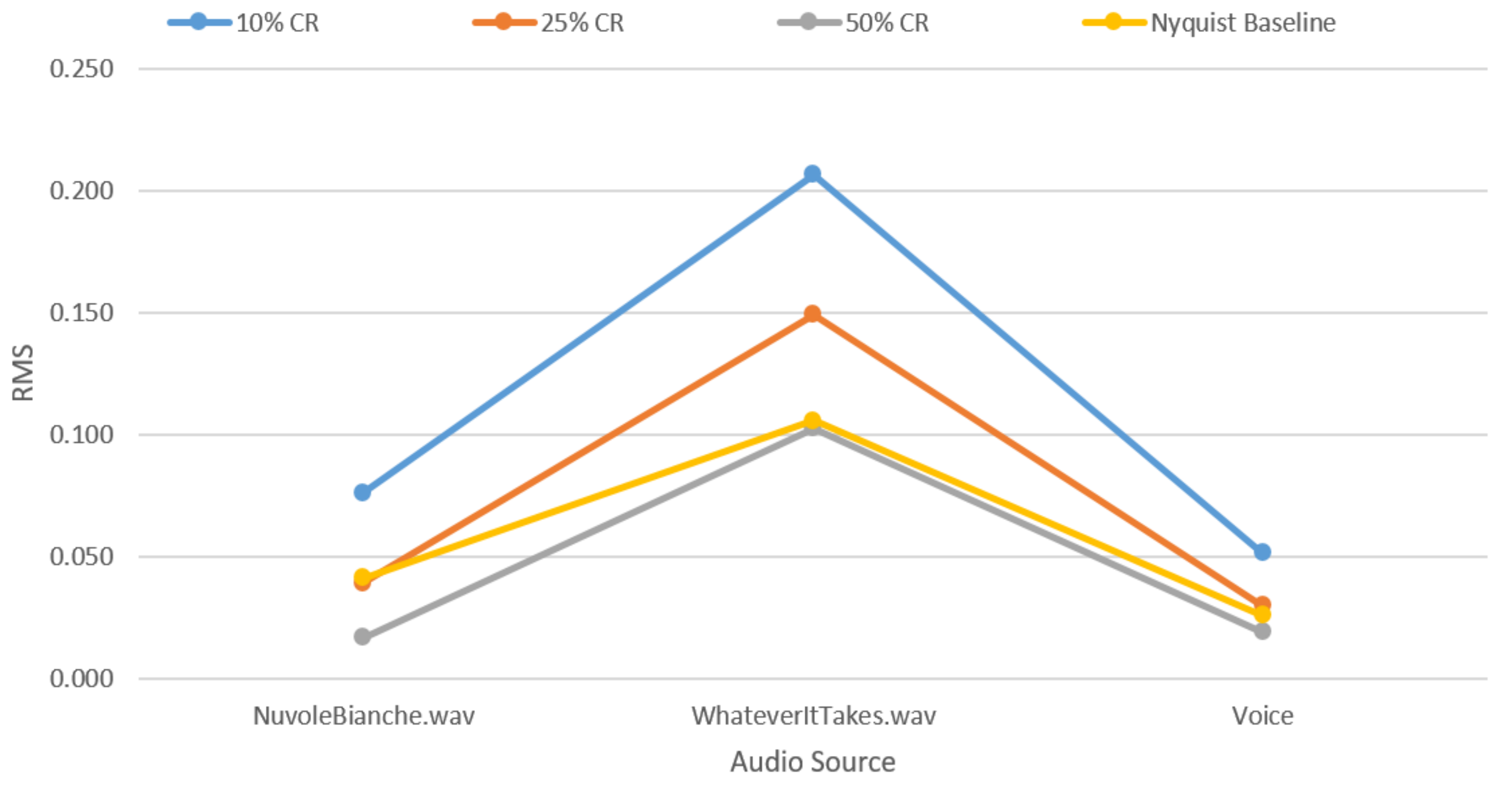
In depth comparison – CS vs Zip

- CS compresses to smaller file, but at what cost?
- Zip not always faster, but one trial



Results

- Opinion: RMS $\leq$ 0.02
- CR ~30%



Conclusion

- We still need to sample at Nyquist Rate to have original xo
- Using a Bernoulli distribution (Φ) and DCT (Ψ) for A, we can assume a $\mu = 0.35$, as well as better storage and ease of use with only real values
- Python producing expected results to Matlab simulations
- Raspberry Pi 3 can utilize multithreading to achieve real-time processing
- CS comparable to zip – possibly superior if sampling sub-Nyquist

Further Research

- Random sampling for complete sequence [17]
- Buffer overrun at 44.1 kHz in Raspberry Pi
- Raspberry Pi microphone with adjustable Fs
- 2D and 3D signals

References	File folder	
create_Phi.m	MATLAB Code	1 KB
detect_Fmax.m	MATLAB Code	1 KB
detect_K.m	MATLAB Code	1 KB
detect_Mu.m	MATLAB Code	1 KB
detect_Noise.m	MATLAB Code	1 KB
Matlab_TxRx.m	MATLAB Code	7 KB
SL0.m	MATLAB Code	6 KB
Thesis Data.xlsx	Microsoft Excel W...	46,865 KB
Ruprecht_ThesisReport.pdf	PDF File	1,704 KB
AudioFile.py	Python File	5 KB
MicrophoneRecord.py	Python File	8 KB
Python_Rx.py	Python File	5 KB
Python_Tx.py	Python File	1 KB
Phi128.txt	TXT File	96 KB
Phi256.txt	TXT File	384 KB
Phi512.txt	TXT File	1,536 KB
Phi1024.txt	TXT File	6,144 KB
Clarity.wav	VLC media file (.w...	2,617 KB
NuvoleBianche.wav	VLC media file (.w...	64,073 KB
NuvoleBianche_1min.wav	VLC media file (.w...	10,337 KB
WhateverItTakes.wav	VLC media file (.w...	37,889 KB
WhateverItTakes_1min.wav	VLC media file (.w...	10,335 KB

References

[1] D. L. Donoho, "Compressed Sensing," IEEE Transactions on Information Theory, vol. 52, no. 4, pp. 1289-1306, 2006.

[2] C. Moler, ""Magic" Reconstruction: Compressed Sensing," MathWorks, 2010. [Online]. Available: https://www.mathworks.com/company/newsletters/articles/magic-reconstruction-compressed-sensing.html. [Accessed 2017].

[3] E. Candes and J. Romberg, L1 magic: Recovery of Sparse Signals via Convex Programming, Caltech, 2005.

[4] S. Pinto, L. Menoza, H. Velandiz, V. Molina and a. L. Cervelon, "Compressive Sensing Hardware in 1-D Signals," TECCIENCIA, vol. 7, no. 19, pp. 5-10, 2015.

[5] E. Candes and M. Wakin, "An Intoduction to Compressive Sampling," IEEE Signal Processing Magazine, pp. 21-30, 2008.

[6] Y. C. Eldar, "Compressed Sensing of Analog Signals in Shift-Invariant Spaces," Israel Science Foundation, 2009.

[7] H. Huang, S. Misra, W. tang, H. Barani and H. Al-Azzawi, "Aplications of Compressed Sensing in Communications Networks," New Mexico State University, 2014.

[8] T. L. N. Nguyen and Y. Shin, "Deterministic Sensing Matrices in Compressive Sensing: A Survey," Soongsil University, 2013.

[9] "Complex Matrices; Fast Fourier Transform," MIT OCW, Cambridge, 2011.

[10] G. Mohimani, M. Babaie-Zadeh and C. Jutten, "Fast Sparse Representation based on Smoothed l0 Norm," Center for International Research, Tehran, 2006.

[11] E. Candes and J. Romberg, Sparsity and Incoherence in Compressive Sampling, Caltech, 2006.

[12] H. Xue, L. Sun and G. Ou, Speech Reconstruction based on Compressed Sensing Theory using Smoothed L0 Algorithm, Nanjing: IEEE, 2016.

[13] T. T. Do, L. Gan, N. H. Nguyen and T. D. Tran, "Fast and Efficient Compressive Sensing using Structurally Random Matrices," IEEE Transactions on Signal Processing, pp. 1-16, 2011.

[14] C. Luo, F. Wu, J. Sun and C. W. Chen, "Compressive Data Gathering for Large-Scale Wireless Sensor Networks," Microsoft Research Asia, Beijing.

[15] Z. Yu, S. Hoyos and B. M. Sadler, Mixed-Signal Parallel Compressed Sensing and Reception for Cognitive Radio, IEEE, 2008.

[16] F. Chen, A. P. Chandrakasan and V. Stojanovic, A Signal-Agnostic Compressed Sensing Acquisition System for Wireless and Implantable Sensors, Cambridge: IEEE, 2010.

[17] J. Laska, S. Kirolos, Y. Massoud, R. Baraniuk, A. Gilbert, M. Iwen and M. Strauss, Random Sampling for Analog-to-Information Conversion of Wideband Signals, DARPA, 2011.

[18] J. Moreira, What is... A RIP Matrix?.

[19] Y. Wnag, J. Ostermann and Y.-Q. Zhang, Video Processing and Communications, Upper Saddle River: Prentice-Hall, 2002.

[20] J. G. Proakis and D. G. Manolakis, Digital Signal Processing Principles, Algorithms, and Applications, Upper Saddle River: Pearson Education, 2007.

Questions

?