Universität Rostock

Traditio et Innovatio

Specialization Module

Efficient Resampling of Inertial Sensor Signals

Submitted on 31.08.2018

by

Naga Mamatha Gonuguntla

Matrikel-Nr.: 216206223

Supervisor:
M.Sc. Jens Rudolf
Universität Rostock
Richard-Wagner-Str. 31
18119 Rostock

Supervisor:
Prof. Dr.-Ing. Christian Haubelt
Universität Rostock
Richard-Wagner-Str. 31
18119 Rostock

Table of Contents

1. Introduction……4

1.1. Motivation……...4

1.2. Structure of the project……...4

2. Resampling:In Detail……5

2.1. Decimation ……6

2.2. Interpolation ……8

3. Literature Research: Overview……...10

3.1. Polynomial Interpolation……...10

3.1.1. Zero order or Sample and Hold Interpolation……...10

3.1.2. Linear Interpolation……...11

3.1.3. Quadratic Interpolation……11

3.1.4. Cubic Interpolation……12

3.1.5. Cubic Spline Interpolation……12

3.2. Sinc Interpolation……...13

4. Implementation……..……..……14

4.1. Implementation Flow Chart……..14

4.2. Implementation Procedure……...15

4.3. Implementations……16

5. Testing and Results……..……..17

5.1. Testing……17

5.2. Results……21

5.3. Scope for Improvements in Future……23

6. Conclusions……..…24

7. References……...24

List of Figures

Figure 1: Traditional Resampling 5
Figure 2: Block diagram of modern Resampling methods 5
Figure 3: Schematic diagram of Sample Rate Conversion by M/N method 6
Figure 4: Decimation by a factor of M=4 [5] 6
Figure 5: Decimation with Aliasing 7
Figure 6: Block diagram of a Decimator 7
Figure 7: Pictorial sequence of Decimation in frequency domain [6] 7
Figure 8: Block diagram of an Interpolator 8
Figure 9: Interpolation by a factor of 3 [5] 8
Figure 10: Pictorial sequence of Interpolation in frequency domain [6] 9
Figure 11: Zero Order Interpolation [9] 10
Figure 12: Linear Interpolation [10] 11
Figure 13: Quadratic Interpolation [11] 11
Figure 14: Cubic Interpolation [12] 12
Figure 15: Cubic Spline Interpolation [13] 12
Figure 16: Sinc Interpolation [15] 13
Figure 17: Plot from Python implementation – case B 18
Figure 18: Plot from Matlab1 implementation – case B 18
Figure 19: Plot from Matlab2 implementation – case B 18
Figure 20: Plot from Python implementation - case C 19
Figure 21: Plot from Matlab1 implementation - case C 19
Figure 22: Plot from Matlab2 implementation - case C 20
Figure 23: Heat-map like representation of results from all the test cases 22
Figure 24: Total execution time for sensor data recorded while walking 22
Figure 25: Mean Statistical Error for sensor data recorded while walking 22
Figure 26: Total Execution time for sensor data recorded for with random motion 23
Figure 27: Mean Statistical Error for sensor data recorded with random motion 23

1. INTRODUCTION

1.1. Motivation

In this technology driven world, for various reasons, different systems operate for different bandwidth limits and in turn using different sample rates. When two such systems are interconnected, for retaining the originality of the signal exchanged between these systems, Resampling or Sample rate conversion is necessary [[1].

The systems of interest in our case are Computers that process the data recorded from consumer devices to recognize gestures. Consumer devices such as smart devices, play stations, navigation, virtual reality etc. a feature called gesture recognition. The movements of an object are tracked by the Inertial sensors connected to it. These recordings are processed by some softwares or machine learning algorithms to recognize the gestures. In the process of gesture recognition the data from these sensors might be exchanged between various systems that may process the data only at some fixed sample rate. Also, some of the Inertial sensors such as BMF055 (from Bosch Sensortec) can be configured to limited Bandwidth ranges. Thus, the data recorded by these sensors may not be available at desired sample rates.

The main aim of the project is to resample the sensor data and make them available at desired sample rates. To accomplish the same an algorithm was implemented in Python using Numpy library. Numpy is a package in Python used for scientific computations, it can store multidimensional data, has tools for integrating C/C++ codes and most importantly Python and all its packages are for free. Also, codes run faster in Python and take comparatively pretty much less computational time. Although MATLAB provides a ready-made function to resample any data, it is not available for free to everyone. An another goal of the project to show that implementations in Python are faster and efficient with acceptable accuracy.

1.2. Structure of the project:

The project starts with Literature research about Resampling, its role in Digital Signal Processing and various methods and algorithms such as Polynomial Interpolation and Sinc Interpolation, that can be used to resample a signal. The overview of the Literature study is presented in one of the upcoming chapters.

Amongst all the methods studied, I chose one efficient method, I.e., Sinc Interpolation, to implement an algorithm, in Python using Numpy library to resample Recorded Inertial Sensor signals. The resampled signals were compared to the original signal to calculate mean statistical error. The original signal, resampled signal and the Statistical error obtained are plotted using a Python library MatplotLib.

As part of the work, two similar algorithms were implemented in the MATLAB environment to compare the fastness and accuracy of the implementations in both Python and MATLAB. Of two implementations in MATLAB, One is implemented exactly with same logic as in the one implemented in Python and the other is implemented using MATLABs' resample function. The total execution time and mean statistical errors were recorded for three implementations and compared to prove the efficiency of the implementation in Python. The results are included to the end of the document.

2. RESAMPLING: IN DETAIL

Let us consider an example. I have sampled an audio track at Fs = 44.1kHz(say), I.e., I have 44100 samples per sec that used 16bits per sample and took certain amount of the device memory in which it is stored. Now if I want to transfer the audio track to an another device that has less space available than the actual size of the audio track, I must sample the audio track at a rate less than the original sample rate [2]. This can be achieved by Resampling the signal without degrading any information contained in the original signal. On that note, one definition of Resample that justifies the purpose is "Changing the sample rate of an already sampled signal using discrete operations is called Resampling" [2].

There are several methods that can sample the signal to a new rate. One simple method is to convert the digital input signal into an analog signal (reconstruction) and resample the signal to a new sample rate in the analog hardware[1], [2].

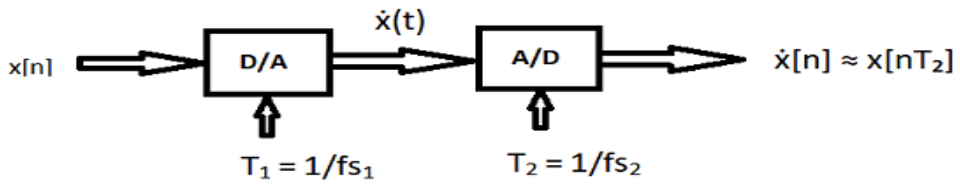


Figure 1: Traditional Resampling

However, the harmonics and the noise distortion resulting due to conversion from Digital to Analog and from Analog to Digital degrade the performance.

Yet another method, that is more sophisticated, is to perform some discrete operations on the already sampled input signal in the digital systems itself, which would give us the same output as in the method discussed above. We shall discuss about these discrete operations in the upcoming chapters.

Figure 2: Block diagram of modern Resampling methods

Signal Resampling can be achieved by performing two signal processing operations, Interpolation and Decimation. The sample rate of the signal can be increased by Interpolation and the same can be decreased by Decimation. When both Interpolation and Decimation are together performed on a signal, we derive to the most popular conventional technique called M/N method, as discussed in [4]. The new sample rate obtained from this method is given by

$$SR_{New} = SR_{Old} \times (M/N) \quad \text{---------- (1)}$$

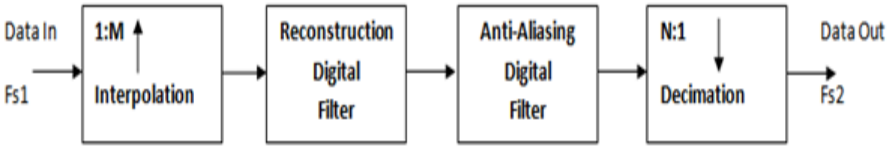


Figure 3: Schematic diagram of Sample Rate Conversion by M/N method

Where,
SR_{New} is the sample rate of the Resampled signal
SR_{Old} is the sample rate of the Original signal
M is the factor by which the signal is Interpolated
N is the factor by which it is Decimated
M/N is a rational factor by which the signal is resampled.

2.1. Decimation

The discrete operation(in Fig 2.) performed on an input signal sampled at F_s to decrease the sample rate to F_s' , where $F_s' < F_s$, is called Decimation.

$$\begin{aligned} F_s' &< F_s \\ T_s' &> T_s \\ F_s' &= F_s \times (1/M) \\ T_s' &= M \times T_s \end{aligned} \qquad \text{---------- (2)}$$

Where, M is the factor by which the sample rate of the original signal is decremented.

The figure 4 below illustrates an example of downsampling a signal by a factor of 4.

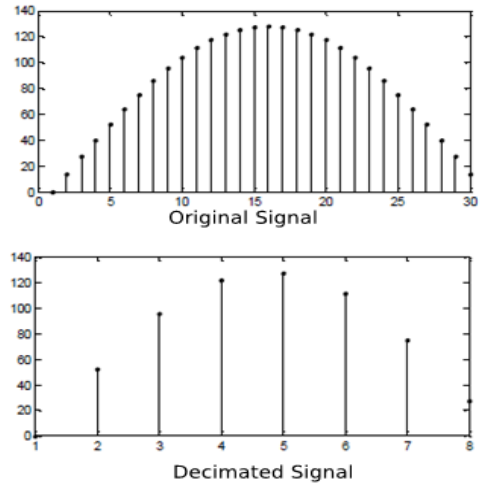


Figure 4: Decimation by a factor of M=4 [5]

In order to downsample a signal, which is a sequence of samples , we retain every Mth i.e., every 4th element in the sequence, dropping the rest. Let us consider the signal in Frequency domain.

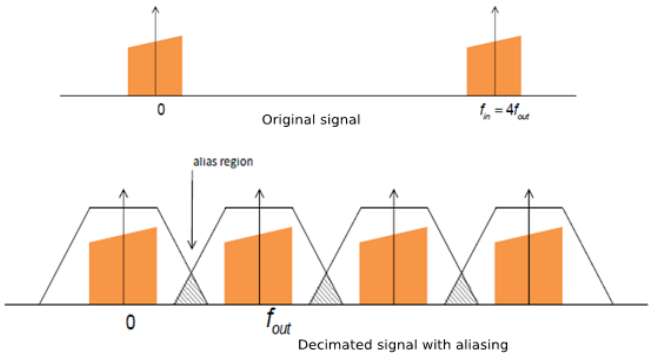


Figure 5: Decimation with Aliasing

To avoid aliasing, the original signal must be sampled at a rate that satisfies the Nyquist - Shannon Sampling Theorem. That means the effect of aliasing can be eliminated by band limiting the original signal to Π/M radians/sec. This can be achieved by introducing an anti-aliasing filter (a low pass filter) with cut off frequency Π/M and then downsampling the original signal. The block diagram of such a decimator is shown in the figure below.

Figure 6: Block diagram of a Decimator

Where $T_2 = MxT_1$

Generally, the combination of an Anti-Aliasing Filter and a Downsampler is called a Decimator.
The complete process of decimation can be generalized as below:

1. The original signal with sample rate F_s is filtered by a low pass Anti Aliasing Filter to avoid Aliasing effect. The sample rate of the signal is still F_s.
2. The signal is now downsampled by a factor M, retaining every M^{th} sample.
3. The output signal obtained has the sample rate $F_s' = F_s/M$.

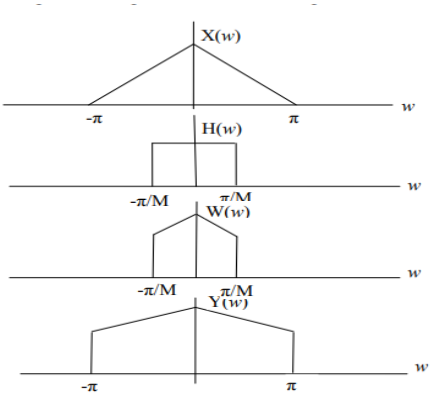


Figure 7: Pictorial sequence of Decimation in frequency domain [6]

2.2. Interpolation

The discrete operation(in Fig 2.) performed on an input signal sampled at F_s to increase the sample rate to F_s' , where $F_s' > F_s$, is called Interpolation.

$$F_s' > F_s$$
$$T_s' < T_s$$
$$F_s' = F_s \times L$$
$$T_s' = (1/L) \times T_s \qquad \text{----------(3)}$$

Where, L is the factor by which the sample rate of the original signal is incremented.

The block diagram of an Interpolator is shown in the figure below.

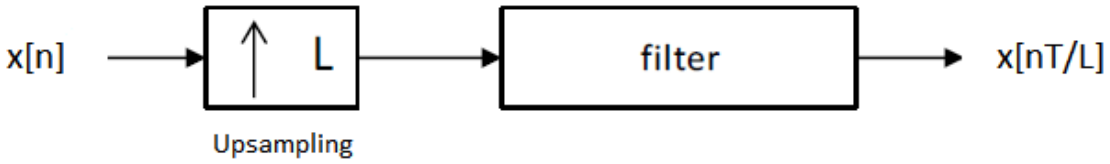


Figure 8: Block diagram of an Interpolator

Unlike in Decimator, where downsampling is all the operation performed; in an Interpolator, upsampling of the signal is always followed by filtering. Without the use of a low pass filter in the Interpolator the process is just upsampling and not interpolation. Then, what is upsampling and what is the role of a low pass filter in the system?

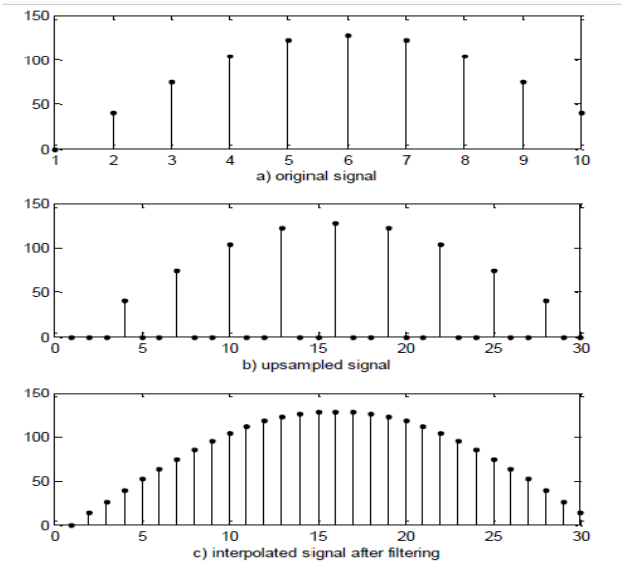


Figure 9: Interpolation by a factor of 3 [5]

Upsampling inserts L-1 zeros between every two input samples, also called as zero-stuffing. After upsampling the signal is no more same as the original signal, due to the undesired copies of the sample. The interpolation filter calculates the values at these L-1

points, interpolate the signal and filter out the undesired copies to output the resampled signal equivalent to the original signal. This process can also be called as Reconstruction. The interpolation filter is designedto have a gain of L and cut of frequency Π/L.

The complete process of Interpolation can be generalized as below.

1. The original signal is upsampling by insering L-1 zeroes between every two input samples.
2. The new sample rate Fs' = LxFs.
3. A low pass filter with Gain = L and cut of frequency ω = Π/L reconstructs the upsampled signal.

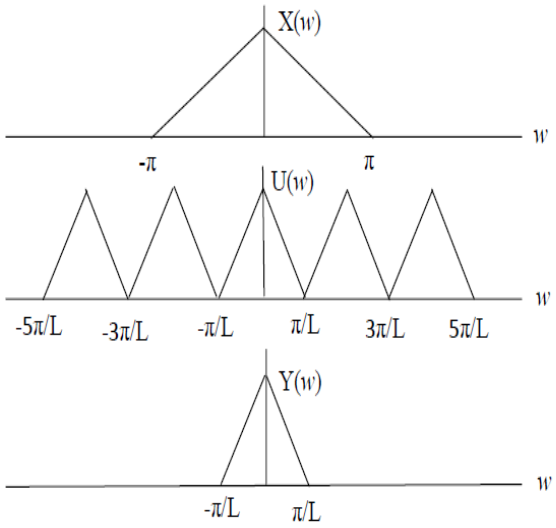


Figure 10: Pictorial sequence of Interpolation in frequency domain [6]

3. Literature Research: Overview

In Digital environment, there are many algorithms that can resample a given signal. In my study[7],[8] I have learnt about different algorithms of Polynomial Interpolation and the most sophisticated Sinc Interpolation. I this chapter I shall discuss these algorithms in detail along with pros and cons of each of them.

3.1. Polynomial Interpolation

The interpolation function is given by a polynomial of the form

$$f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_kx^k$$

The order of the polynomial is chosen according to the method of Interpolation.
The following algorithms of Polynomial Interpolation are discussed:

1. Zero Order or Sample and Hold Interpolation
2. Linear Interpolation
3. Quadratic Interpolation
4. Cubic Interpolation
5. Cubic Spline Interpolation

3.1.1. Zero Order or Sample and Hold Interpolation

This method is considered to be the most simplest of the all the algorithms. As in any interpolation method, the process starts with inserting L-1 number of zeros between each pair of the input samples where L is interpolation factor.

Procedure

1. Insert L-1 zeros between each pair of samples.
2. The value of the signal at each of these L-1 points is equal to the sample preceding these L-1 points.
3. This method results in Staircase-like curve, with each step corresponding to one single value.

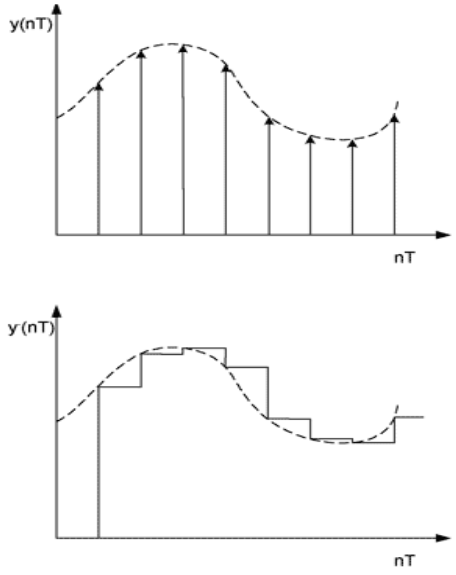


Figure 11: Zero Order Interpolation [9]

This method is very fast and simple, but has high distortions at higher frequencies degrading the quality of the signal

3.1.2. Linear Interpolation

This method is also called First-Order Interpolation

Procedure

1. Insert L-1 zeros between each pair of input samples.
2. Samples at L-1 points are determined by the line joining the pair of input samples between which these L-1 points are inserted.
3. The result is a curve that is much smoother than that obtained by Zero-Order Interpolation.

This method is simple and produces good results, but also produces noise and high distortions at high frequencies.

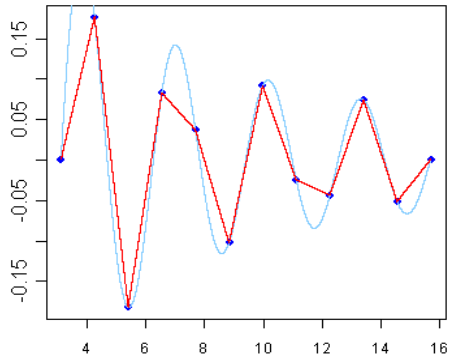


Figure 12: Linear Interpolation [10]

3.1.3. Quadratic Interpolation

This method forms a parabolic curve joining three adjacent samples. Let's say N_i and N_{i+1} are pair of input samples with L-1 new sample points in between, then the third input sample required to compute L-1 new samples can be chosen according to Lagrange method of order three. The third input sample could be N_{i+2} or N_{i-1} whichever is nearest to the L-1 sample points.

Procedure

1. Insert L-1 zeros between each pair of input samples.
2. Samples at L-1 points are determined by a 2nd order polynomial.
3. Results in a Parabola, which is smoother than the previous method.

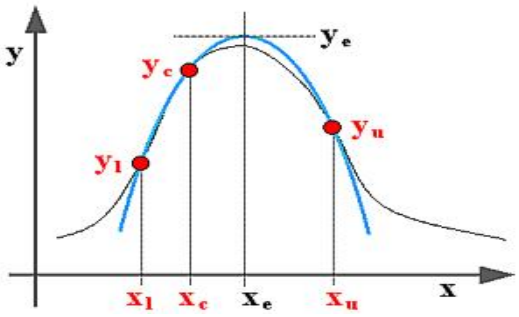


Figure 13: Quadratic Interpolation [11]

This method gives smooth performance than Linear Interpolation. But the curve is asymmetrically placed and has sharp edges between adjacent curves.

3.1.4. Cubic Interpolation

This method is an extension to the Quadratic Interpolation. It results in an third order curve using 4 adjacent samples. Let's say N_i and N_{i+1} are pair of input samples with L-1 new sample points in between, then the other two input samples are N_{i-1} and N_{i+2}.

Procedure

1. Insert L-1 zeros between each pair input samples.
2. The values at L-1 new sample points are determined using 3rd order polynomial.
3. Results in a third order curve.

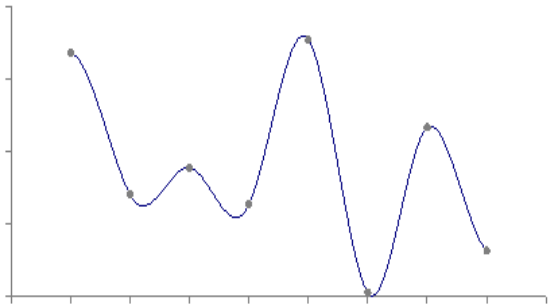


Figure 14: Cubic Interpolation [12]

This method produces viable results. But the method itself is computationally intensive and requies high memory.

3.1.5. Cubic Spline Interpolation

This method is an extension to the Cubic Interpolation to smoothen the sharp edges between the adjacent curves.

Procedure

1. Insert L-1 zeros between each pair input samples.
2. The values at L-1 new sample points are determined using 3rd order polynomial.
3. To achieve the desired smoothes between the curves, the first derivative of the one curve must be equal to the tangent of the other curve beginning.

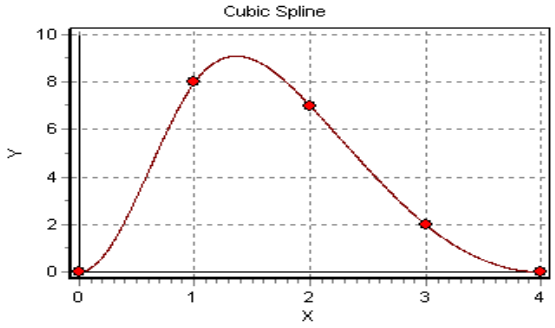


Figure 15: Cubic Spline Interpolation [13]

This method also produces viable results with smoother edges between the adjacent curves. But it is also computationally intensive and requires more memory.

3.2. Sinc Interpolation

The Interpolation formula is given by using a sinc function.

$$X(t) = \sum_{n=-\infty}^{\infty} x_n \operatorname{sinc}\left(\frac{\pi}{T}(t - nT)\right) \quad (4)$$

T is the sampling rate of the original signal
X_n are the samples of the original signal
X(t) is the reconstructed signal after interpolation.
The formula is a Linear convolution of the sample sequence x_n and the sinc function which is scaled and shifted [14]. To interpolate the value at a new sample point, a waveform is generated for each discrete sample of the original signal when convolved with sinc function. The sum of all N waveforms(say the signal has N samples) gives the value at new sample point.

Procedure

1. Insert L-1 zeros between each pair of input samples.
2. The value at these new sample points is given by the equation (4).
3. Results in a very smooth curve with no higher harmonics added.

This method produces very good results with low distortion. It can also be used to limit the signal bandwidth too. But it is a computationally very intensive method.

These pros make this method the most efficient of all and is the method of interest in the project. The computational intensity can be handled by Numpy.

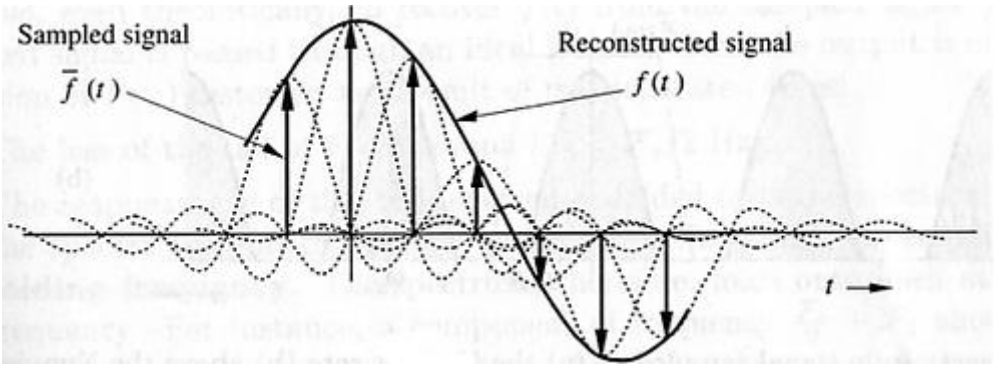
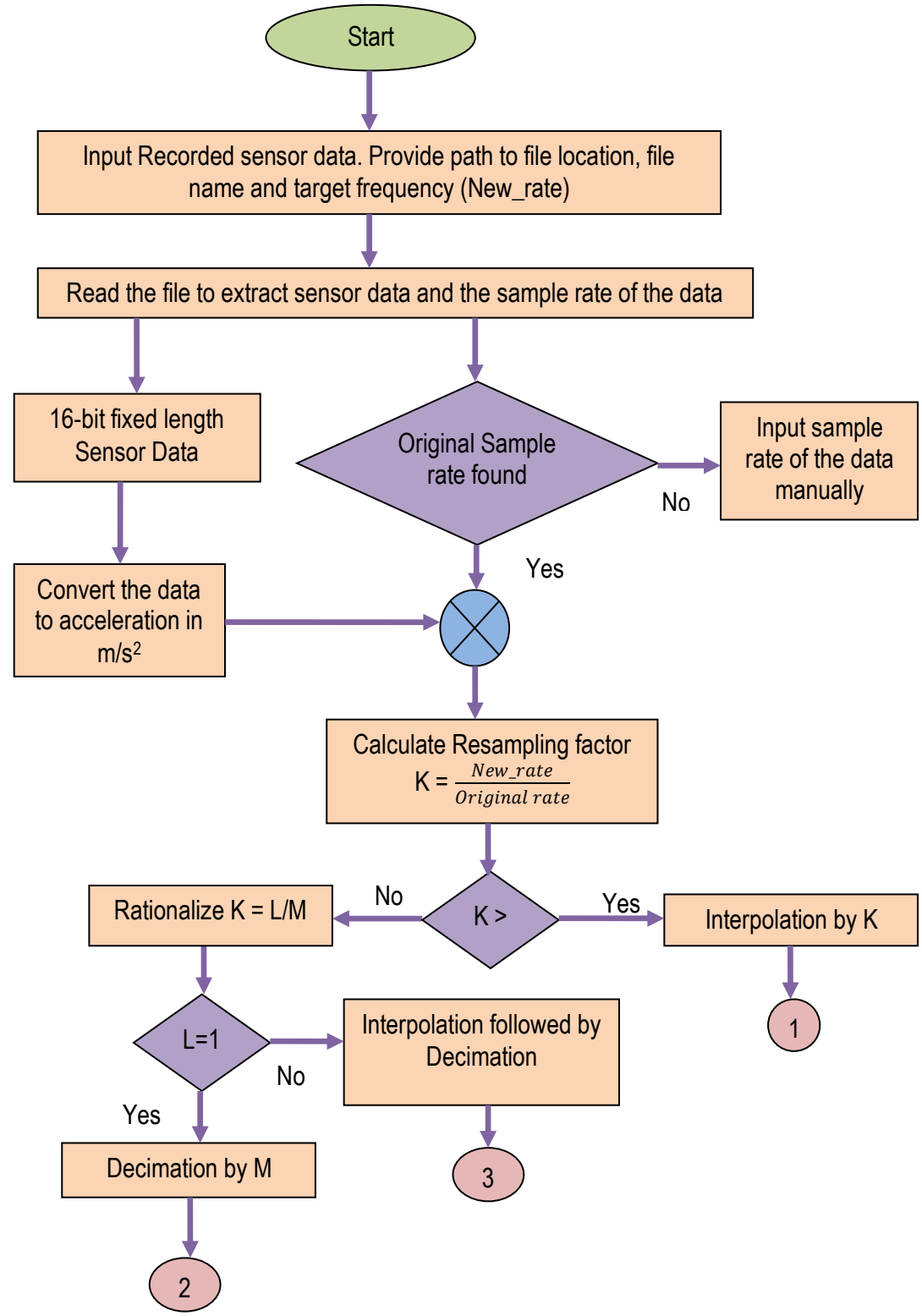


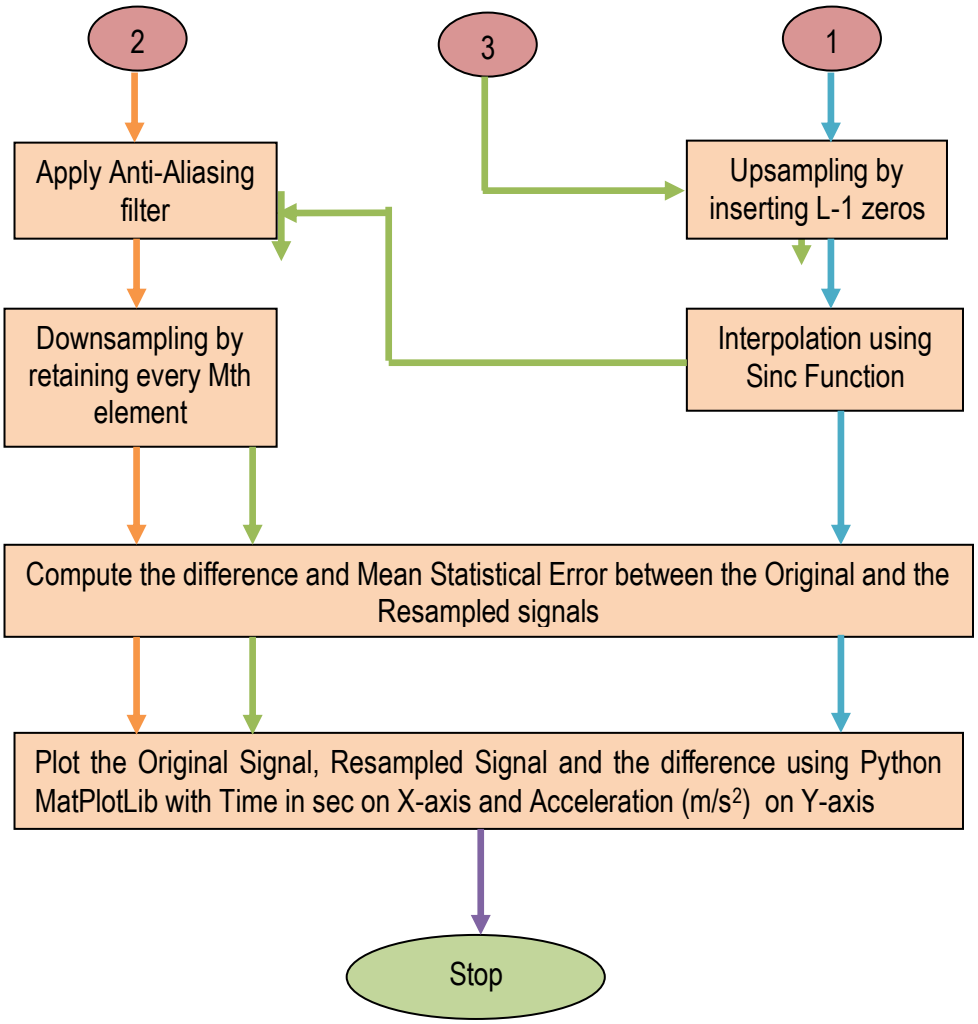
Figure 16: Sinc Interpolation [15]

4. IMPLEMENTATION

4.1. Implementation Flow chart

The Resampling algorithm was implemented in both Python and MATLAB. Below presented is a flow chart of the algorithm.





4.2. Implementation Procedure

Pre-requisites of the Data to be resampled:

The program for now only accepts csv files and the sensor data contained in the file is of 16-bit fixed-length format.

1. The program starts with accepting the inputs from command line. The path to the location of the file, file name and the target sample rate to which the original signal is to be resampled to.
2. The program reads through the file to extract the sensor data and the frequency at which it is already sampled. The extracted sensor data is stored in multi-dimensional array using Numpy.
3. If the scan could not find the original frequency of the data, then the user must input it manually.
4. The data is converted to actual acceleration by the formula $X = X * (9.8 * 2^{12})$. The sensor data is collected at say ±2g, ±4g, ±8g or ±16g, where g = 9.8m/s^2.
5. Resampling factor is calculated by desired sample rate and the original sample rate by the formula.

$$K = \frac{\text{New\_rate}}{\text{Original rate}}$$

6. If K > 1, then it means the signal has to be resampled to an higher frequency. An Interpolation operation by a factor of K is performed.
7. Else If K = 1, no operation is performed.
8. Else, K< 1, then means the signal has to be resampled to an lower frequency. But since downsampling cannot be performed with a factor of float type, therefore, K is rationalized to L/M.
9. If L=1, signal is decimated by a factor of M.
10. Signal is first interpolated by a factor of L and then the interpolated signal is decimated by a factor of M.
11. To Interpolate the signal, the signal is first upsampled by inserting L-1 sample points between each pair of input samples and these are interpolated using sinc function.
12. To Decimate the signal, the signal is passed through an Anti-Aliasing filter to remove higher frequencies that cause Aliasing effect. Then the signal is downsampled by a factor of M.
13. The difference between the original signal and the resampled signal is computed and the mean statistical error is calculated by the formula

    $$\text{Mean Statistical Error} = \frac{\text{Sum of the values at all sample points}}{\text{Total number of sample points}}$$

14. The original signal, the resampled signal and the difference between the original and the resampled signals are plotted using MatPlotLib with Time(s) on X-axis and Acceleration(m/s^2) on Y-axis.

4.3. Implementations

To prove the efficiency of the algorithm to Resample recorded Inertial Sensor signals, a total of 3 programs are implemented, one in Python using Numpy and MatPlotLib and the other two in MATLAB.

1. An algorithm to resample the signal is implemented in Python using Numpy with Sinc Interpolation and Decimation with Ant-Aliasing Filter.
2. A similar algorithm is implemented in MATLAB also with Sinc Interpolation and Decimation with Anti-Aliasing Filter.
3. The third program is implemented using MATLABs' resample function.

5. TESTING AND RESULTS

The algorithm implemented was tested with Accelerometer data for various cases of Resampling – Interpolation, Decimation and Decimation in L/M method.

Note:

Python: Implemented using Sinc Interpolation and Decimation with Anti-Aliasing filter.

Matlab1: Implemented using MATLAB resample function

Matlab2:Implemented using Sinc Interpolation and Decimation with Anti-Aliasing filter.

Time – measured in seconds

Error – measured in m/s^2

For easy understanding and precise observation, all the calculations and plots are limited to single axis of the signal.

5.1.Testing

5.1.1. Accelerometer Data recorded while Walking sampled at 500Hz Number of samples = 500

A. Original sample rate = 500Hz
New sample rate = 150Hz

Resampling Factor = $\frac{3}{10}$, Interpolation by 3 and Decimation by 10.

	Time	Statistical Error
Python	0.187	-0.022107854284319665
Matlab1	0.309	-0.001175332567012
Matlab2	0.418	-0.022044166725216

B. Original sample rate = 500Hz
New sample rate = 1000Hz

Resampling Factor = 2, Interpolation by 2.

	Time	Statistical Error
Python	0.125	1.5109933311732715e-07
Matlab1	0.279	-4.438703584177872e-05
Matlab2	0.282	1.510993333462239e-07

The plots below are plotted in Python and MATLAB for case B. For simplicity only x-axis is plotted.

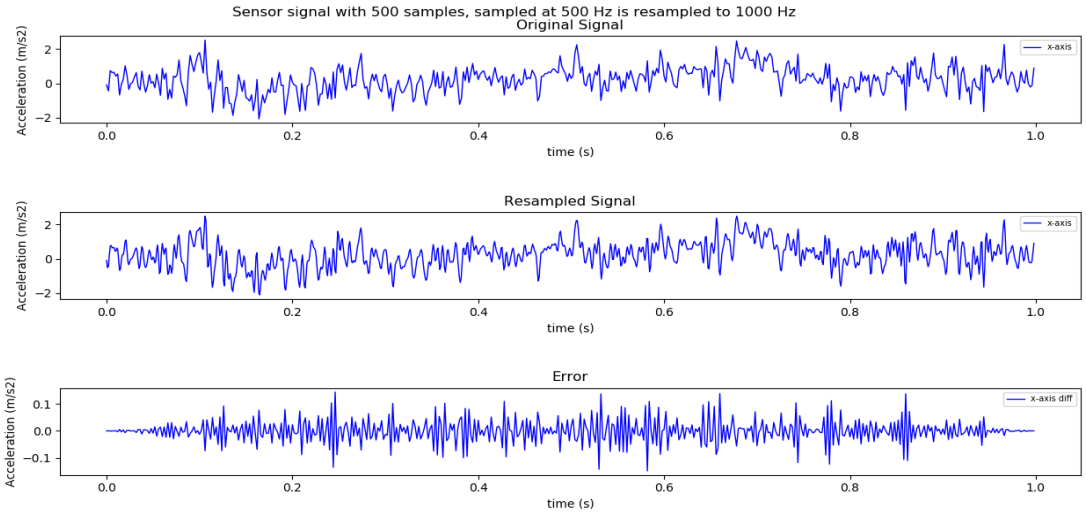


Figure 17: Plot from Python implementation – case B

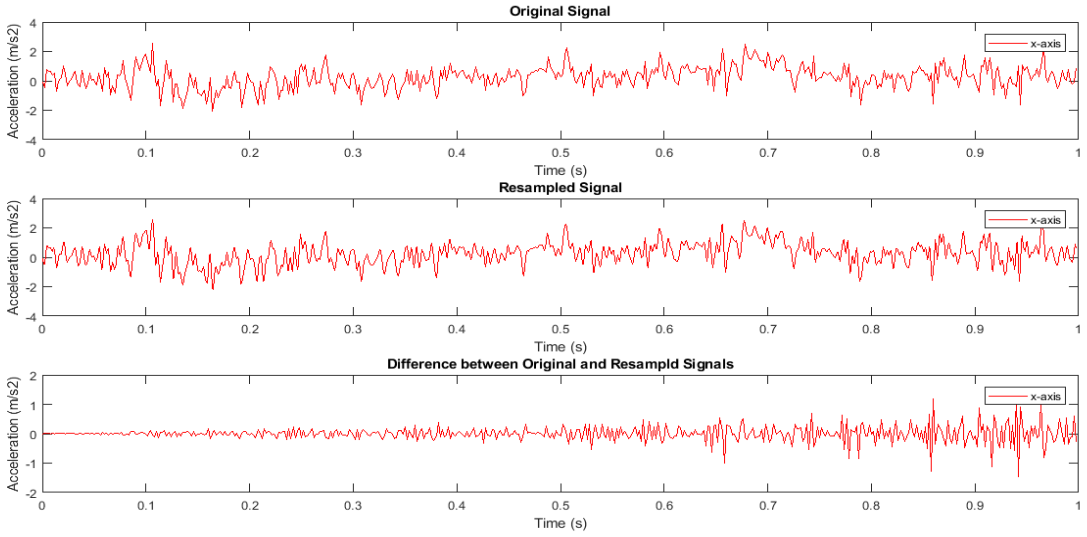


Figure 18: Plot from Matlab1 implementation – case B

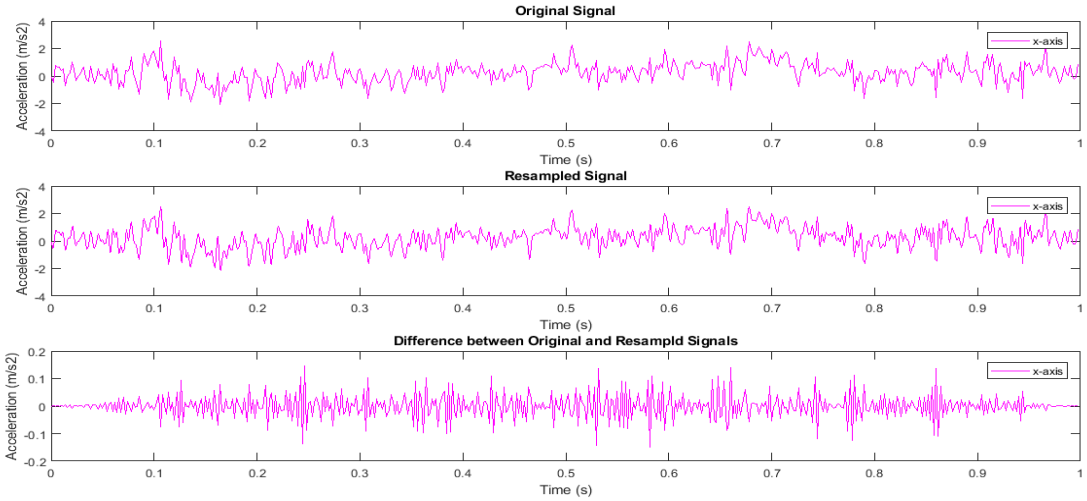


Figure 19: Plot from Matlab2 implementation – case B

C. Original sample rate = 500Hz

New sample rate = 125Hz

Resampling Factor = 4, Decimation by 4.

	Time	Statistical Error
Python	0.016	0.010178940837147906
Matlab1	0.302	4.039807341880564e-05
Matlab2	0.218	6.493907737347564e-04

The plots below are plotted in Python and MATLAB for case C. For simplicity only x-axis is plotted.

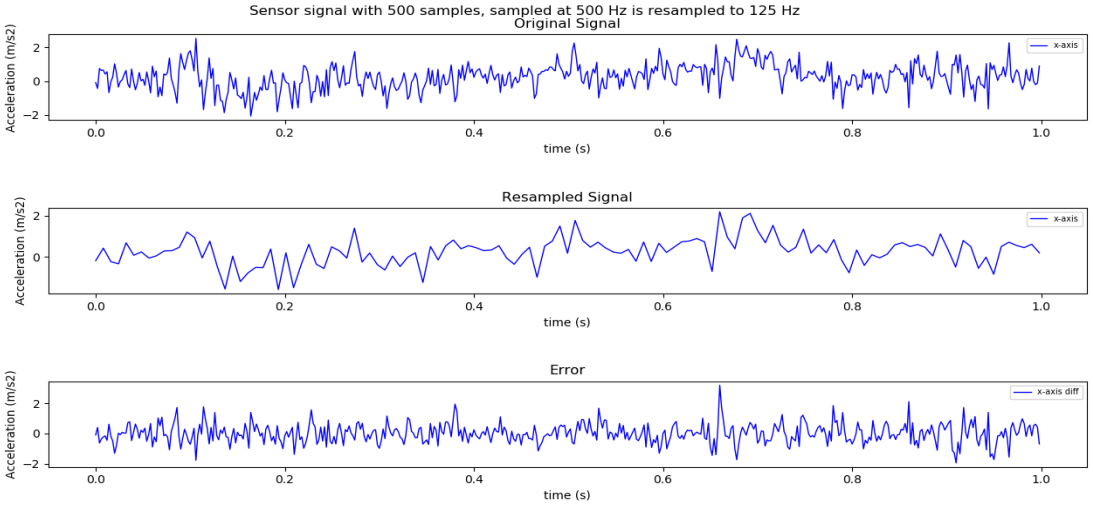


Figure 20: Plot from Python implementation - case C

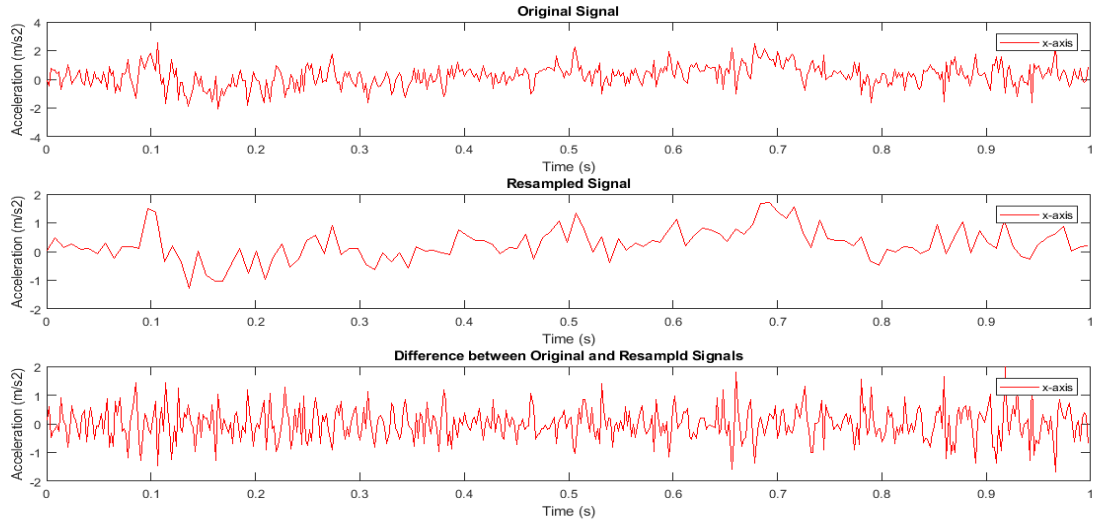


Figure 21: Plot from Matlab1 implementation - case C

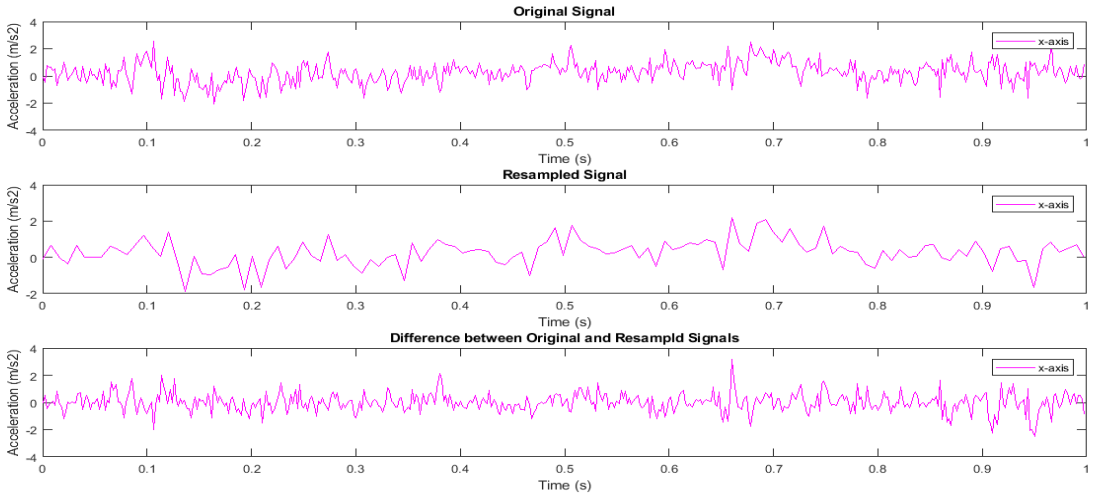


Figure 22: Plot from Matlab2 implementation - case C

D. Original sample rate = 500Hz
New sample rate = 750Hz

Resampling Factor =1.5, Interpolation by 1.5.

	Time	Statistical Error
Python	0.094	0.002958964701664497
Matlab1	0.316	0.002893742165074
Matlab2	0.295	0.002958964701664

5.1.2. Accelerometer Data recorded with Random Motion sampled at 250 Hz Number of samples = 1000

E. Original sample rate = 250Hz
New sample rate = 150Hz

Resampling Factor = $\frac{3}{5}$, Interpolation by 3 and Decimation by 5.

	Time	Statistical Error
Python	0.560	-0.0036446050311457617
Matlab1	0.297	-0.001886781645576
Matlab2	0.695	-0.003171834974892

F. Original sample rate = 250Hz
New sample rate = 1000Hz

Resampling Factor = 4, Interpolation by 4.

	Time	Statistical Error
Python	0.719	3.6867698977141596e-07
Matlab1	0.279	-0.002242802978971
Matlab2	0.774	3.686769917587776e-07

G. Original sample rate = 250Hz
New sample rate = 125Hz
Resampling Factor = 2, Decimation by 2.

	Time	Statistical Error
Python	0.008	-0.0012266793137653104
Matlab1	0.283	-0.002006816666224
Matlab2	0.220	0.001686286188531

H. Original sample rate = 250Hz
New sample rate = 400Hz
Resampling Factor = 1.6, Interpolation by 1.6.

	Time	Statistical Error
Python	0.297	-6.932385642576023e-05
Matlab1	0.301	-8.471067081713145e-04
Matlab2	0.429	-6.932385642535202e-05

5.2. Results

The consolidated results of all the test cases are presented in this section. A heat-map like result sheet is generated in excel. Line graphs are generated in excel to illustrate Total execution time and Mean statistical error of three implementations considering all the test cases, that show the performance of each implementation. From the consolidated result sheet (Figure 23), the following observations are made. The algorithm implemented in both Python and MATLAB work with same accuracy, yet Python is faster than MATLAB. MATLAB resample function has consistent execution times, yet Python is faster in most cases. Python code gives high accuracy for Interpolation operation and for decimation all the three work with similar accuracy. As the number of samples increase, Python total execution time is more than that of MATLAB resample function for Interpolation operation, but Python has high accuracy than the latter.

Efficient Resampling	Test-case scenario	Sensor data recorded while walking, sampled at 500 Hz and contain 500 samples				Sensor data recorded with random motion, sampled at 250 Hz and contain 1000 samples			
	Resampling type	Decimation	Decimation with L/M method	Interpolation		Decimation	Decimation with L/M method	Interpolation	
	Target Frequency	125 Hz	150 Hz	750 Hz	1000 Hz	125 Hz	150 Hz	400	1000 Hz
Total Execution Time	Python	0.016	0.187	0.094	0.125	0.008	0.56	0.297	0.719
	Matlab1	0.302	0.309	0.316	0.279	0.283	0.297	0.301	0.279
	Matlab2	0.218	0.418	0.295	0.282	0.22	0.695	0.429	0.774
Mean Statistical Error	Python	0.010179	-0.022108	0.00295896	1.51E-07	-0.0012	-0.003645	-6.93E-05	3.69E-07
	Matlab1	4.04E-05	-0.001175	0.00289374	-4.44E-05	-0.002	-0.001887	-8.47E-04	-0.0022428
	Matlab2	6.49E-04	-0.022044	0.00295896	1.51E-07	0.0017	-0.003172	-6.93E-05	3.69E-07

Figure 23: Heat-map like representation of results from all the test cases

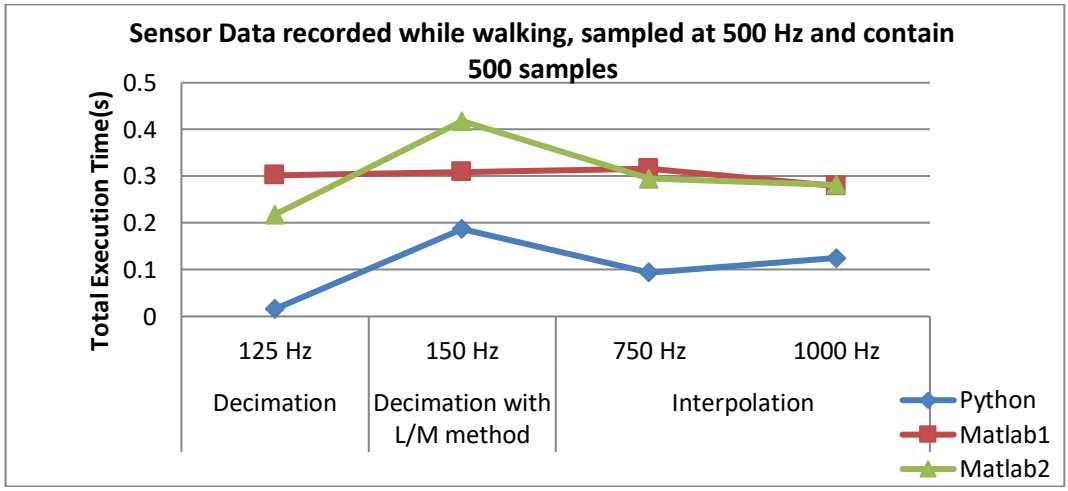


Figure 24: Total execution time for sensor data recorded while walking

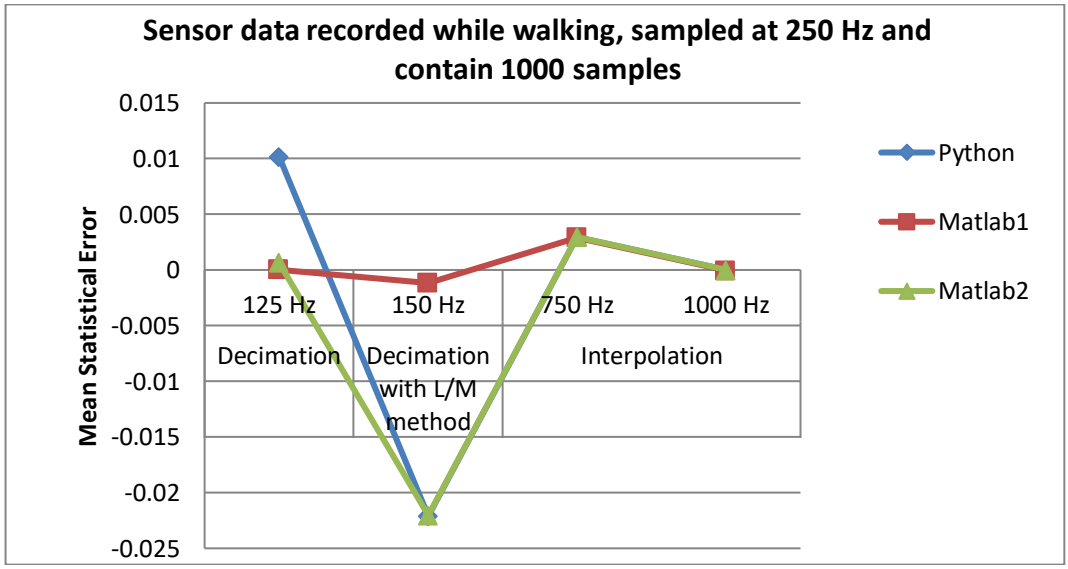


Figure 25: Mean Statistical Error for sensor data recorded while walking

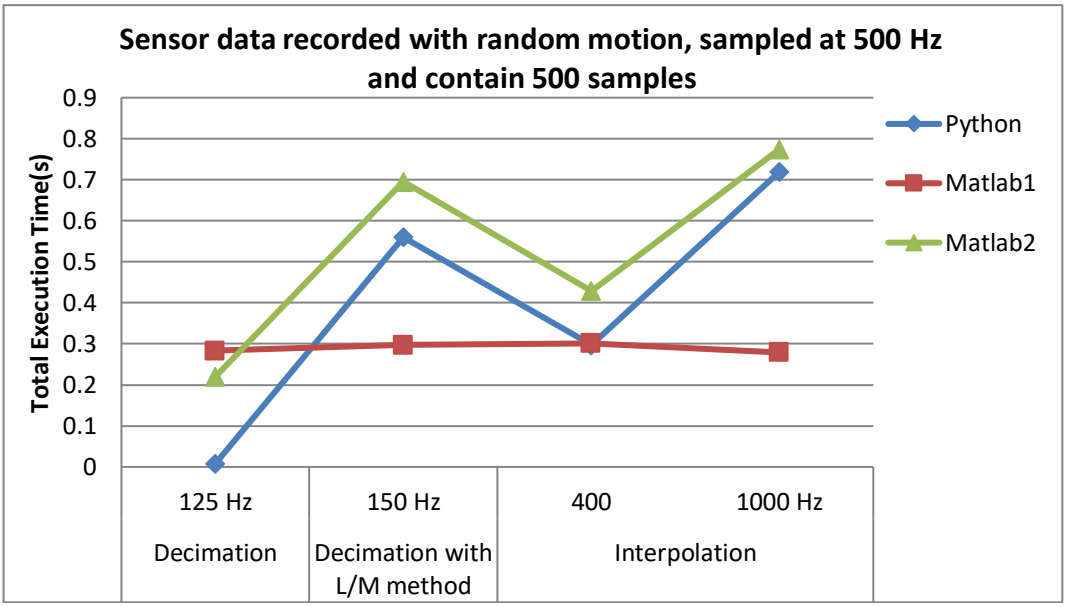


Figure 26: Total Execution time for sensor data recorded for with random motion

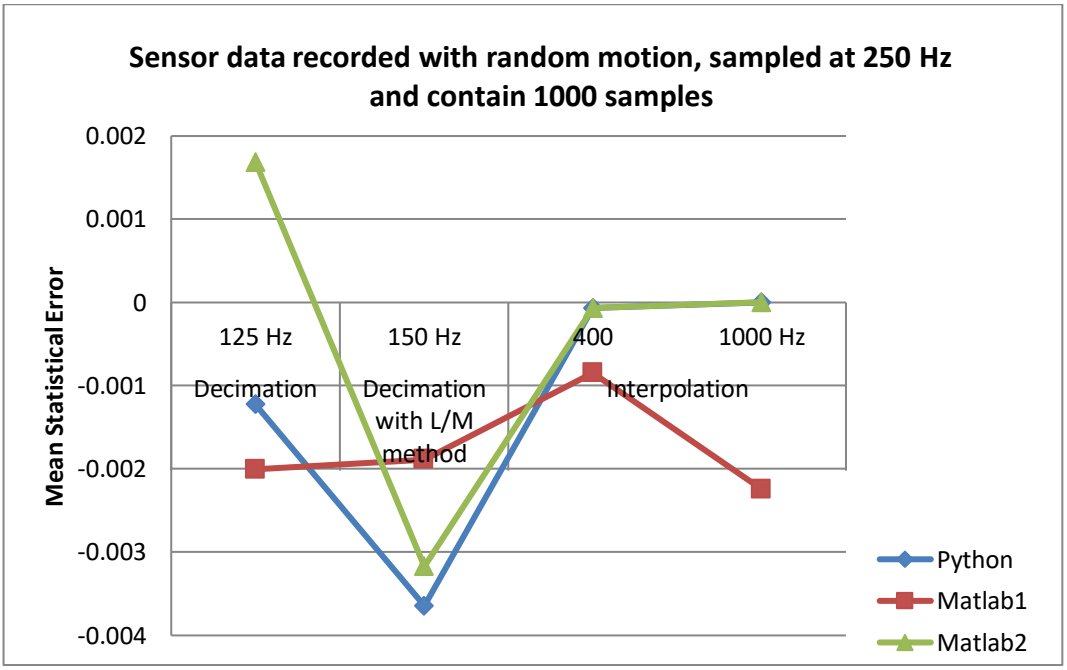


Figure 27: Mean Statistical Error for sensor data recorded with random motion

5.3. Scope for Improvements in Future

Accuracy for Decimation operation in all three implementations is less when compared to that for Interpolation operation. Further work could be done to improve the efficiency of the implementation for decimation. Also, with larger number of samples Python implementation takes more execution time for interpolation. Improvements could be made to speed up the execution for such cases.

6. CONCLUSION

In this project, A resampling algorithm is developed to efficiently resample the recorded Inertial sensor data using Python Numpy. After testing the algorithm in both Python and MATLAB, it is observed that Python code runs faster than the MATLAB code for the same magnitude of the error produced. The algorithm is very efficient for interpolation operation. However, results obtained by performing decimation operation are just satisfactory. Future work could be done to improve the performance. The code can run even more faster with parallel execution over multi-cores. I would like to concluded that, use of Numpy in the implementation made all the difference.

7. References

[1] IEEE SIGNAL PROCESSING LETTERS, VOL. 7, NO. 10, OCTOBER 2000 “An Efficient Algorithm for Sample Rate Conversion from CD to DAT” Kannan Rajamani, Yhean-Sen Lai, and C. W. Farrow

[2] https://www.youtube.com/watch?v=C4gP7L7d91M&index=2&list=PLzecsHoSJ8tJELHC9iyCH_jOrUNaSyaRG SigProcessing - Sampling-Rate conversion: Motivation by Prof. Waheed U. Bajwa

[3] Sangil Park, Garth Hillman and Roman Robles "A Novel Structure for Real-Time Digital Sample-Rate Converters with Finite Precision Error Analysis" Motorola Inc. Digital Signal Processing Operations Austin, Texas 78735

[4] L. R. Rabiner, “Digital Techniques for Changing the Sampling Rate of a Signal”, Digital Audio; Collected papers from the AES Premier conference, pp. 79-89, Rye, NY, June 3-6, 1982

[5] Dr Mike Porteous, "Introduction to Digital Resampling"

[6] Jagdeep Singh, “A Novel Architecture for Sample Rate Converter using FIR Filter”. Master Thesis project, submitted at Thapar University

[7] Vladimir Arnorst, “Comparision of Interpolation Algorithms in Real Time Sound Processing”.

[8] David A.Hall, “Techniques for Interpolation and Digital Upcconversion”. https://www.ecnmag.com/product-release/2007/08/techniques-interpolation-and-digital-upconversion

[9] https://www.maximintegrated.com/en/app-notes/index.mvp/id/3853

[10] http://www.codecogs.com/library/maths/approximation/interpolation/univariate.php

[11] http://www.ebyte.it/library/codesnippets/P3Interpolation.html

[12] http://paulbourke.net/miscellaneous/interpolation/

[13] https://www.orcina.com/SoftwareProducts/OrcaFlex/Documentation/Help/Content/html/InterpolationMethods.htm

[14] https://demonstrations.wolfram.com/SincInterpolationForSignalReconstruction/