College of Engineering
Electrical Engineering Department

EE497

Adaptive Noise/Clutter Cancellation (ANC) Techniques for Passive Radar

Students Team

Abdulaziz Abdullah Alaskar **Student ID: 436101565**

Ali Fahad Altuwaijri **Student ID: 436105437**

Supervisors

Name	Signature
Mubashir Alam	……
Irfan Ahmed	……

Submitted in Partial Fulfillment of the Requirement

for the B.Sc. Degree,

Jumada I 1442

December 2020

PROJECT ABSTRACT

The main objective of this project is to design an adaptive noise/clutter cancellation filter for passive radar applications. The filter was implemented using a simple Arduino based passive radar system. Various methods were tested and compared. The Arduino microcontroller was used to transmit a reference signal, as well as receive the signal to be filtered. The filtering processes were done using MATLAB and Simulink.

ACKNOWLEDGEMENT

We would like to give our thanks and express our gratitude to the supervisors of this project, Dr. Mubashir Alam, and Dr. Irfan Ahmad, who have continuously offered help and support in this project, in guiding and providing us with literature and material to use in the project.

We would also like to relay our sincerest gratitude to Dr. Omar AlDayel, who was more of a third supervisor than an examiner.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 Problem Formulation

1.1.1 Problem Statement:

A radar is an electrical system that transmits radio frequency (RF) electromagnetic (EM) waves toward a region of interest and receives and detects these EM waves when reflected from objects in that region [1]. A passive radar is a system that utilizes the radiation emitted by an external source as a reference and the reflections as detected signals [2]; therefore, it has no transmitting antenna of its own. The reflections detected are, like all other forms of signals, susceptible to noise. Noise distorts the information carried by the reflections. However, as noise is constantly changing, the use of an ordinary filter that is constant in value is not sufficient. The proposed solution is a filter that adapts to the changes in noise. The adaptive filters proposed are driven by algorithms that allow the filter to change according to the noise, by readjusting the weights of the coefficients periodically as the system operates.

1.1.2 Proposed Solution:

The noise/clutter distorting the radar's received signal increase the probability of error in detection greatly. The proposed solution of designing a digital filter is to be implemented in the following manner:

- The design of the filter will be through MATLAB coding.
- A reference signal will be generated using Arduino as well as random noise signal.
- Different algorithms will be used for each attempt.
- The algorithms' performances will be compared in terms of max error and mean squared error.

1.2 Project Specifications

The proposed designs are as follows:

- Least Mean Square (LMS) algorithm: the design is mostly successful. However, the choice of parameters is left to the user, who might cause the system's performance to deteriorate.
- Normalized Least Mean Square (NLMS) Algorithm: the design solves one of the issues of LMS by using the reference signals' power to determine the step size (μ); therefore, reducing the error that may be caused by the user.
- Recursive Least Square (RLS) Algorithm: This system is the most optimum in performance of all attempted algorithms in the project, in nearly all aspects. However, it should be noted that the complexity of the code far exceeds the complexity of the LMS/NLMS systems.
- Arduino was set up and programmed to generate a reference signal and receive the signal to be filtered.

2 BACKGROUND

2.1 Literature Review

The literature that was used in the project consisted of several textbooks and a research paper. The main reference of the project is Adaptive Filter Theory by Simon Haykin. The research papers are mainly concerned with noise cancellation in adaptive radar system.

2.2 Concept Synthesis

2.2.1 Concept Generation:

Solving noise problems with filtering has been used for decades, in many applications, like mobile communication systems, and other commercial services. The issue with passive radar is that the user has no control over the signal power, since the signal is obtained from the surroundings of the receiving antenna. Therefore, the signal to noise ratio (SNR) cannot be increased to overcome noise distortion. Therefore, the use of a filter is a necessity. Ordinary static filters cannot be used in applications where high accuracy is required, such as radar applications, which is the main motivation for using adaptive digital filters.

First, the concept of filtering should be defined:

Filtering: the extraction of information about a quantity of interest at time (τ) by using data measured up to and including time (τ) [3].

Filter: a device in the form of physical hardware that is applied to a set of noisy data in order to extract information about a prescribed quantity of interest [3].

The main issue here is the lack of prior knowledge of the statistics of the data to be processed, therefore the use of an ordinary static digital filter is not suitable. Instead, an adaptive filter is to be designed.

Adaptive Filter: a self-designing filter that relies for operation on a recursive algorithm, which makes it possible for the filter to perform satisfactorily in an environment where complete knowledge of the relevant signal characteristics is not available [3].

Figure 2.1 shows a simplified block diagram of an adaptive filter.

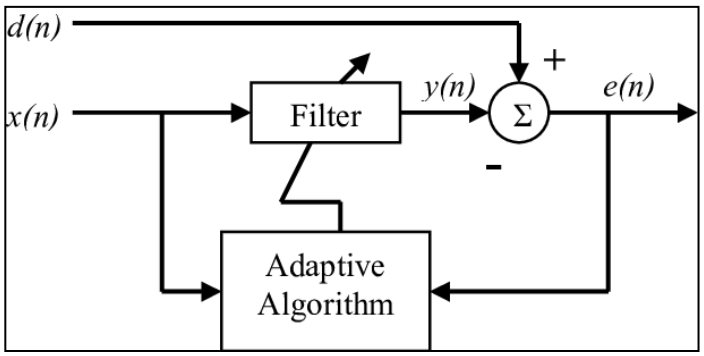

Figure 2.1: General Block of an Adaptive filter

2.2.2 Concept Reduction:

The choice of methods was based on the supervisor's instruction, based on their efficiency and popularity. MATLAB was mainly used because of the electrical engineering focus of the application, as well as the relatively simple coding. The attempted algorithms are famous and have been used in many commercial applications all over the globe and have been proven to be reliable.

2.3 Adaptive Filter Algorithm Analysis

2.3.1 Least Mean Square (LMS) Algorithm:

The least mean square algorithm is linear adaptive filtering algorithm that consists of two basic processes [3]:

1- **Filtering Process**: a process that involves computing the output of transversal filter produced by a set of tap-inputs, and generating an estimation error by comparing the output to a desired response [3].
2- **Adaptive Process**: a process that involves the automatic adjustments of the tap weights of the filter according to the estimation error [3].

The combination of the two processes mentioned above yields out a closed loop feedback system. as seen in figure 2.2:

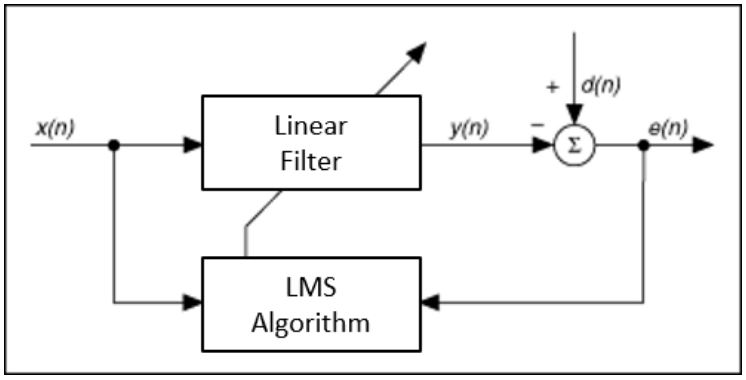

Figure 2.2: LMS Algorithm Block Diagram

The algorithm equations, for a number of taps (order) M, are as follows:

1. Filter Output:

$$y[n] = w[n]x[n] \tag{1}$$

2. Estimation Error:

$$e[n] = d[n] - y[n] \tag{2}$$

3. Tap-Weight Adaptation:

$$w[n+1] = w[n] + \mu x[n]e^{T}[n]$$

$$\text{Given: } 0 < \mu < \frac{2}{\lambda_{max}} \tag{3}$$

Where:

- $x[n]$: System input signal
- $y[n]$: Output of the system.
- $w[n]$: Variable tap weight.
- $d[n]$: Desired output signal.
- $u[n]$: Tap input signal.
- μ: Step-size.
- $\boldsymbol{\lambda_{max}}$: Maximum eigen value.
- $e[n]$: Estimation error signal.

2.3.2 Normalized Least Mean Square (NLMS) Algorithm:

LMS algorithm the correction term $\mu x[n]e^{*}[n]$ applied to the tap weight vector $w[n]$ at iteration (n+1) is directly proportional to the tap-input vector $\boldsymbol{u}[n]$. Therefore, when $x[n]$ is large, the LMS algorithm experiences a gradient noise amplification problem [3].

To solve the issue, the proposed solution to this problem the normalized LMS algorithm is used.

NLMS is different from LMS in that the correction term is normalized by the energy of the tap-input vector as follows:

$$w[n+1] = w[n] + \frac{\beta}{a + |x[n]|^2} x[n]e^{T}[n]$$

$$\text{Given: } 0 < \mu < 2 \tag{4}$$

Where:

- $x[n]$: System input signal
- $y[n]$: Output of the system.
- $w[n]$: Variable tap weight.
- $d[n]$: Desired output signal.
- $u[n]$: Tap input signal.
- β: Step-size.
- $e[n]$: Estimation error signal.
- a: Safety factor

(a) is a safety factor used to prevent the term from going to infinity in case the instantaneous energy $|\boldsymbol{u}[n]|^2$ reaches zero at a given value of n.

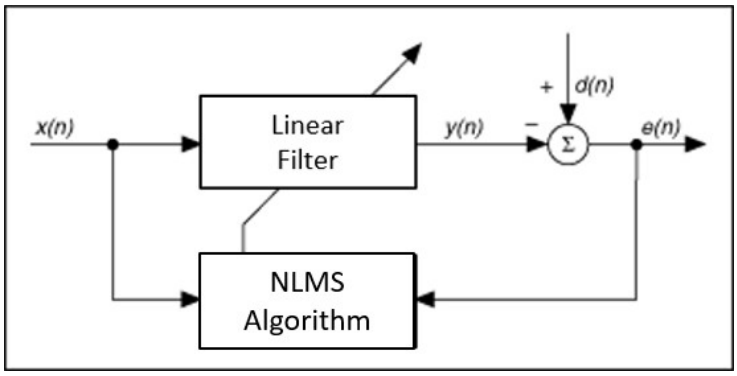

Figure 2.3: NLMS Algorithm Block Diagram

Figure 2.3 shows an illustration of a NLMS based adaptive filter.

2.3.3 Recursive Least Mean Square Algorithm (RLS)

The RLS algorithms aim to minimize the sum of the squares of the difference between the desired signal and the filter output signal using the new samples of the incoming signal. The RLS algorithms compute filter coefficients in a recursive form at each iteration. The RLS algorithms are well known for their fast convergence even when the eigenvalue spread of the input signal correlation matrix is large. These algorithms offer excellent performance with the cost of larger computational complexity and problem of stability in comparison with the LMS algorithm. In next section, we will study the standard RLS algorithm.[4] The main advantage of the RLS algorithm is the focus on the most recent values, and '*forgetting*' older values by multiplying the older values by powers of a forgetting factor denoted (λ). Where $\lambda < 1$.

Figure 2.4 shows a simplified block diagram of an RLS based adaptive filter.

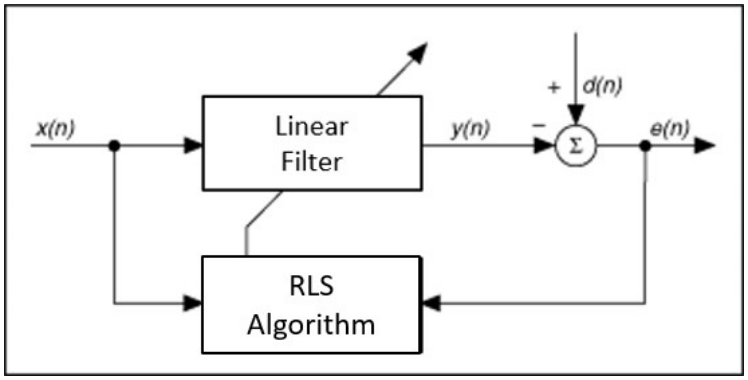

Figure 2.4: RLS Algorithm Block Diagram

3 ALGORITHM TESTING

3.1 Least Mean Square (LMS) Algorithm:

The LMS code is found in APPENDIX A.

For the following inputs:

- Reference frequency: F_r = 100 kHz
- Sampling Frequency: F_s = 10MHz
- Desired Signal: $d[n]$ = sin $(2\pi F_r n + \phi)$
- Filter order: 31
- Step-size: μ = 0.001

System output:

- Mean square error: $\bar{e2} = 1.6874 \times 10^{-4}$
- Maximum error: $e_{max} = 0.0962$

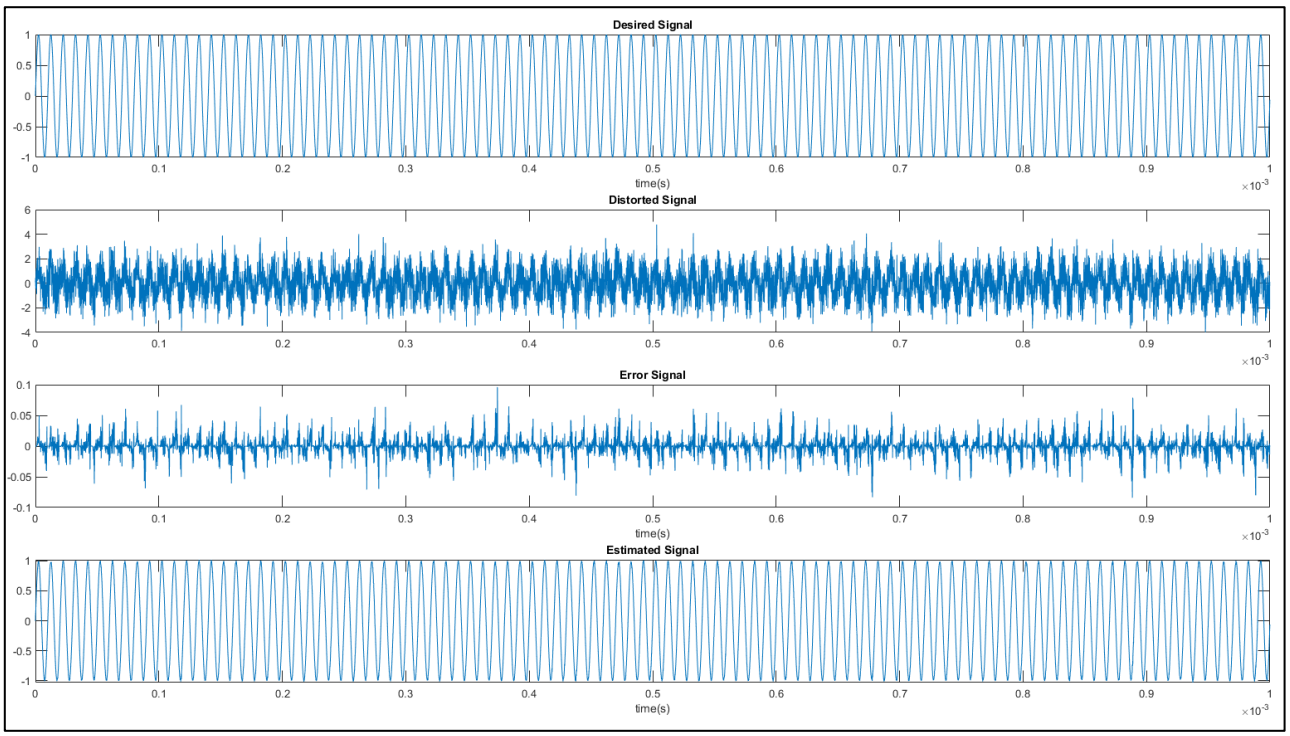

Figure 3.1: LMS Code Output

As illustrated in figure 3.1, the filtered signal greatly matches the desired signal.

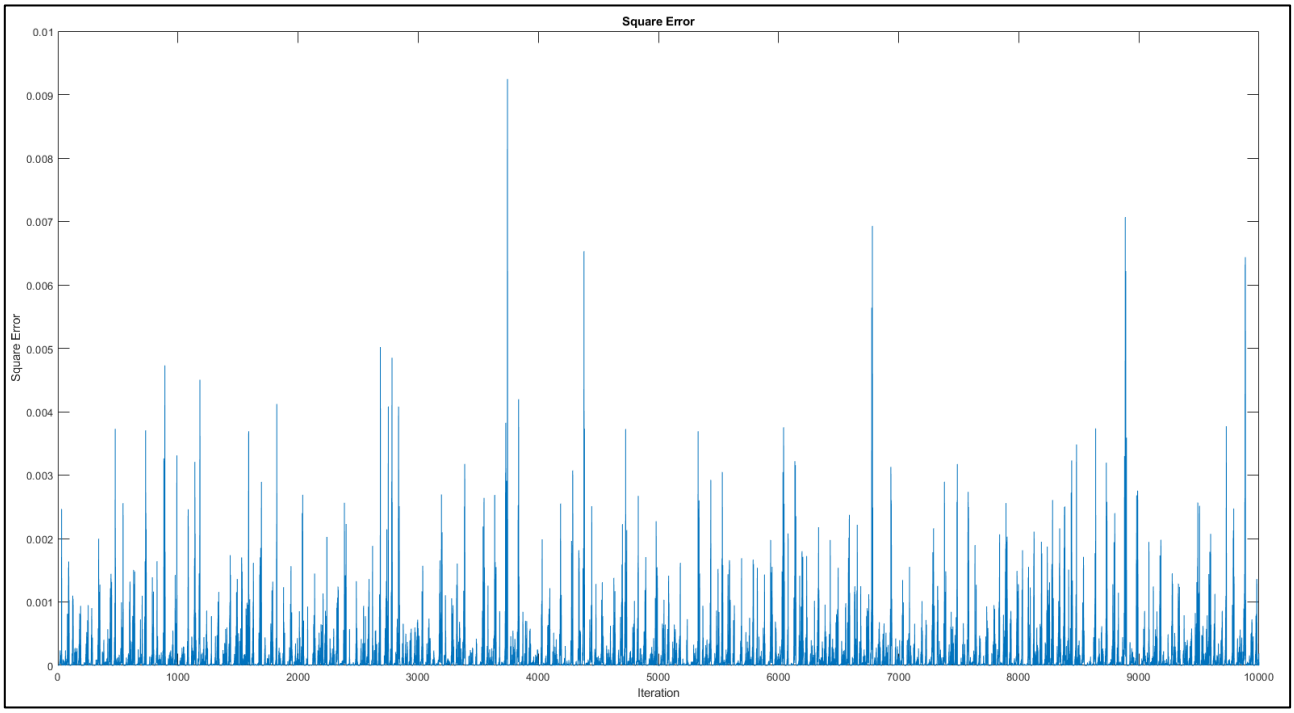

Figure 3.2: Square Error for 10000 Iterations

In figure 3.2, the square error of the system for 10000 samples is illustrated.
The error is fairly contained within a range, and only rarely spikes.
In figure 3.3, it can be seen that the 15th filter tap ($W_{15}(n)$) quickly converges to a mean value, for 10000 iterations

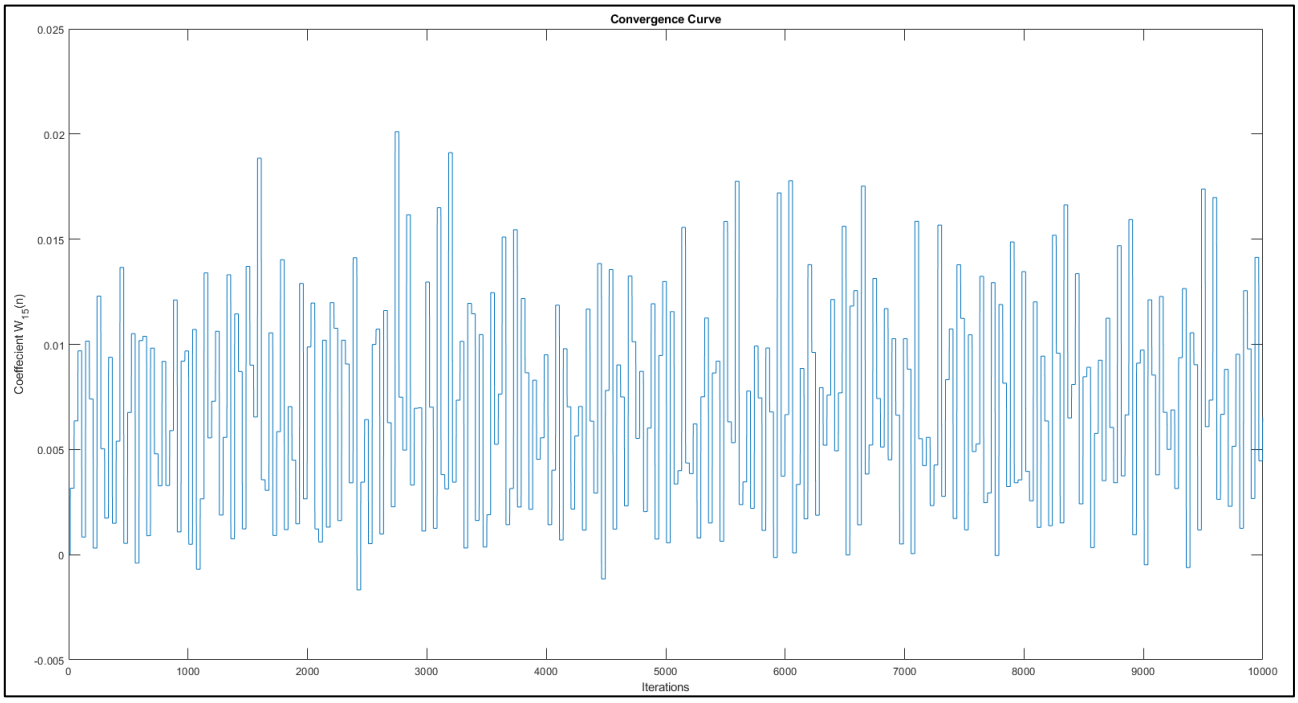

Figure 3.3: Convergence curve for the 15th filter tap for 10000 iterations

3.2 Normalized Least Mean (NLMS) Square Algorithm:

The LMS code is found in APPENDIX A.

For the following inputs:

- Reference frequency: $F_r = 100$ kHz
- Sampling Frequency: $F_s = 10$MHz
- Desired Signal: $d[n] = \sin(2\pi F_r n + \phi)$
- Filter order: 31
- Step-size: $\beta = 0.001$

System output:

- Mean square error: $\bar{e2} = 0.8513$
- Maximum error: $e_{max} = 40.9899$

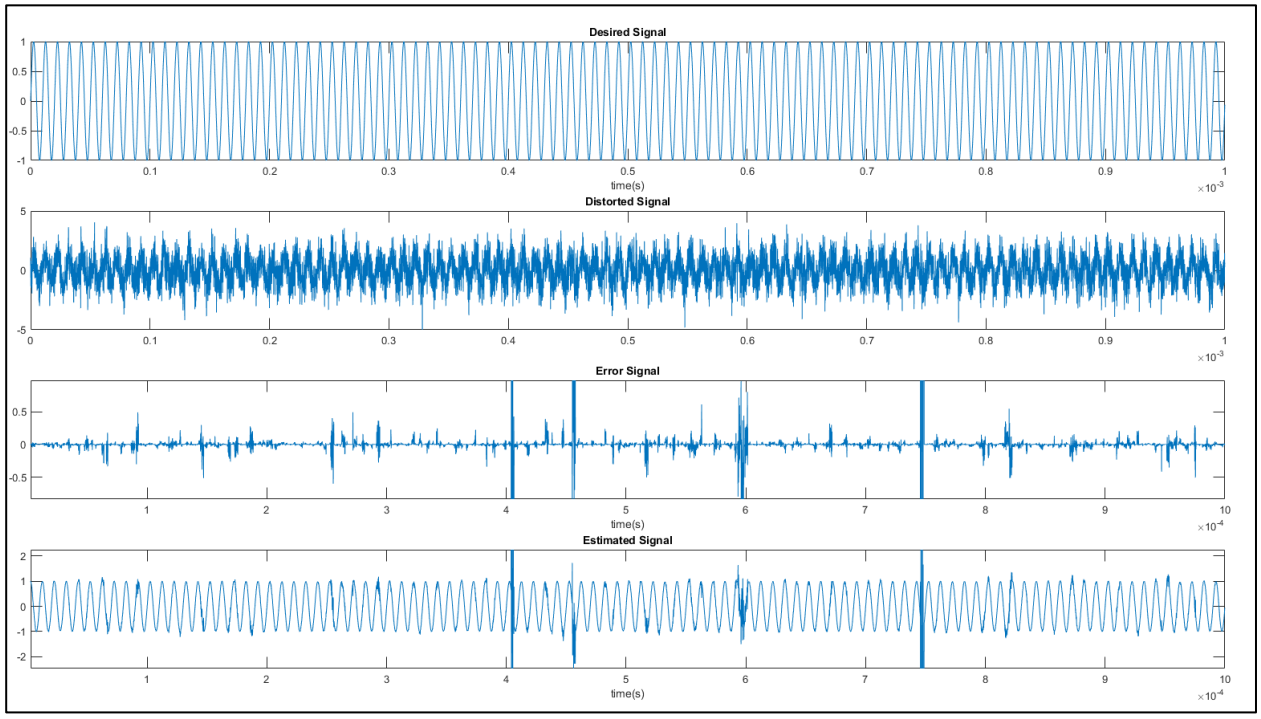

Figure 3.4: NLMS Code Outputs

As shown in figure 3.4, the NLMS system suffers from error surges frequently.
The results correlate with the mean squared error and max error calculated by the code.

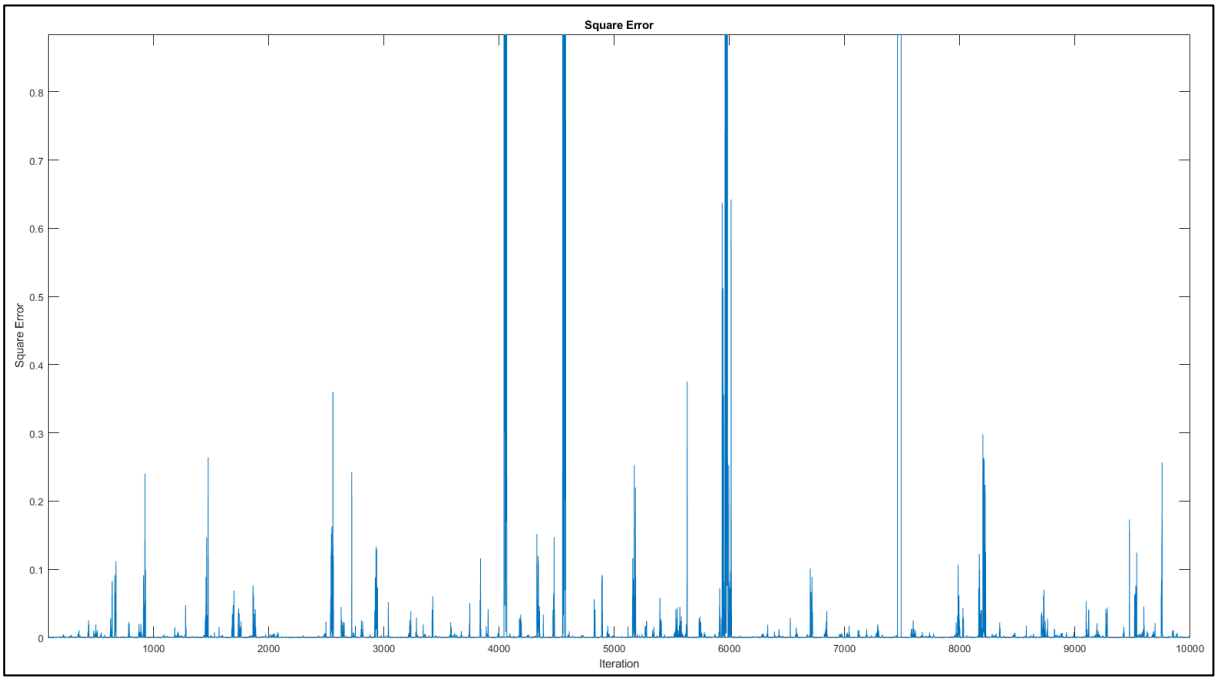

Figure 3.5: Square error for 10000 iterations

In figure 3.5, the square error very frequently surges to significant values, with a mean squared error of 0.8513.

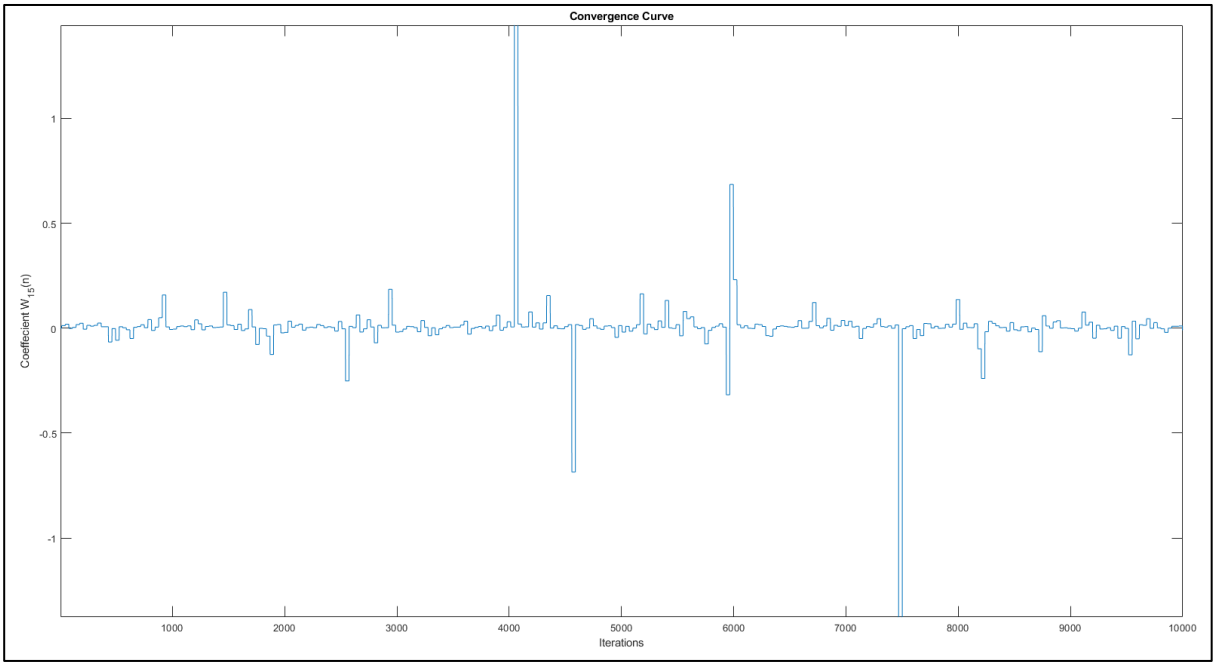

Figure 3.6: Convergence curve for the 15th filter tap.

Figure 3.6 shows that the weight of the 15th filter tap shifts significantly at various iterations of 1000 testing iterations

3.3 Recursive Least Square:

The RLS algorithm's complexity, in terms of the mathematical model and coding process is much greater than the previously tested variations of the LMS algorithm. The RLS algorithm's main advantage is that it focuses on the most recent tap weights, by multiplying older coefficients with small values, their effect on the error correction is minimized, which leads to much faster convergence and smaller error. Another advantage of the RLS algorithm is that the algorithm's performance is much better than LMS at a much lower order.

Using the code in Appendix, the following results were obtained: Order =4

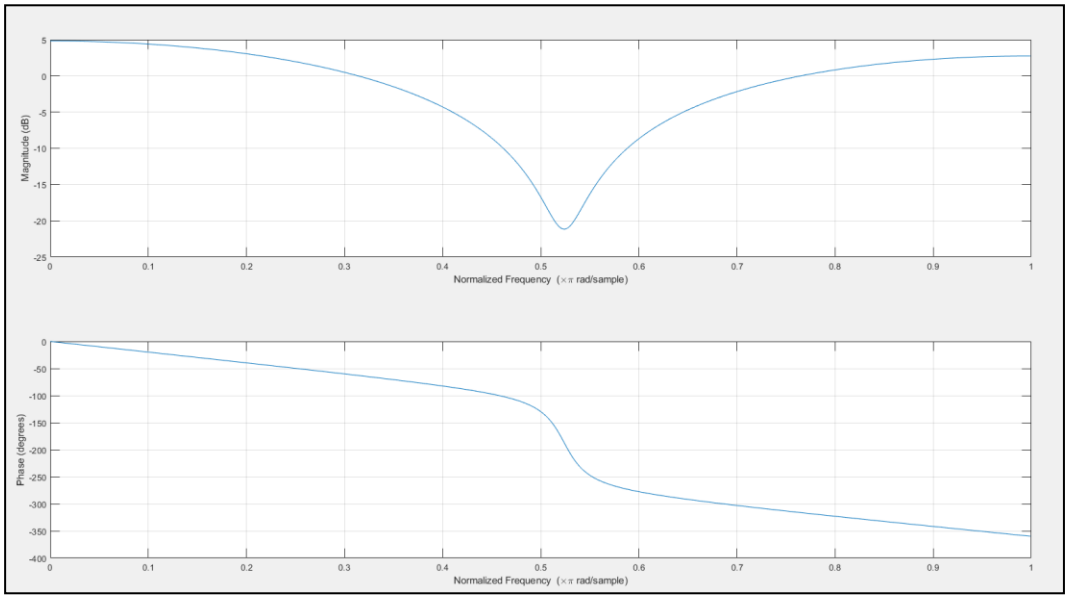

Figure 3.7: RLS Filter Magnitude and Phase responses.

Figure 3.7 shows the frequency magnitude and phase responses of the filter that results from the RLS MATLAB code.

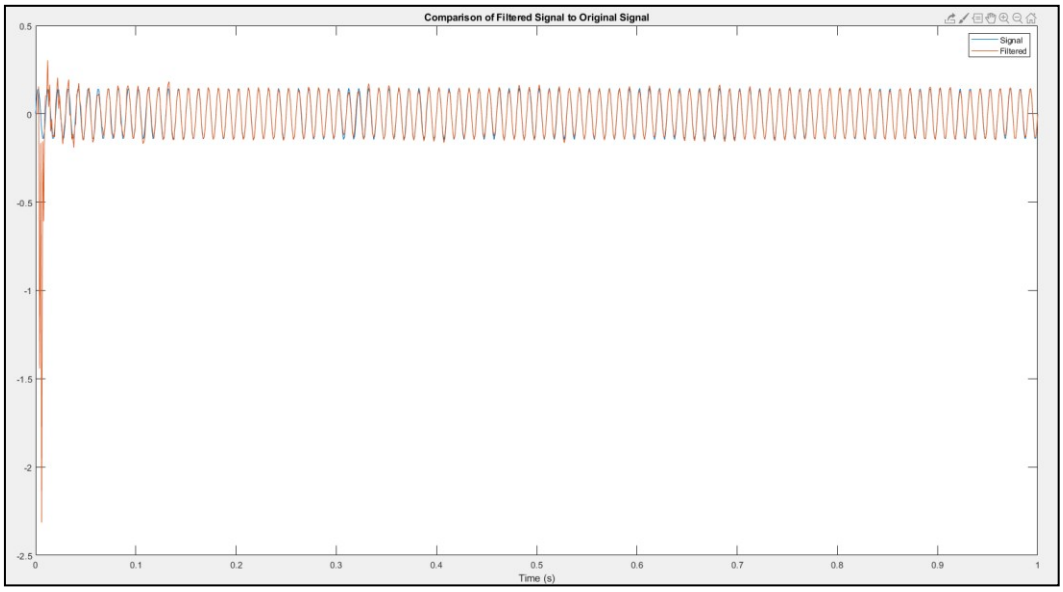

Figure 3.8: Comparison of the desired signal and the filtered signal

As seen in figure 3.8, the estimated signal very quickly converges and follows the desired signal, at a very low order (4).

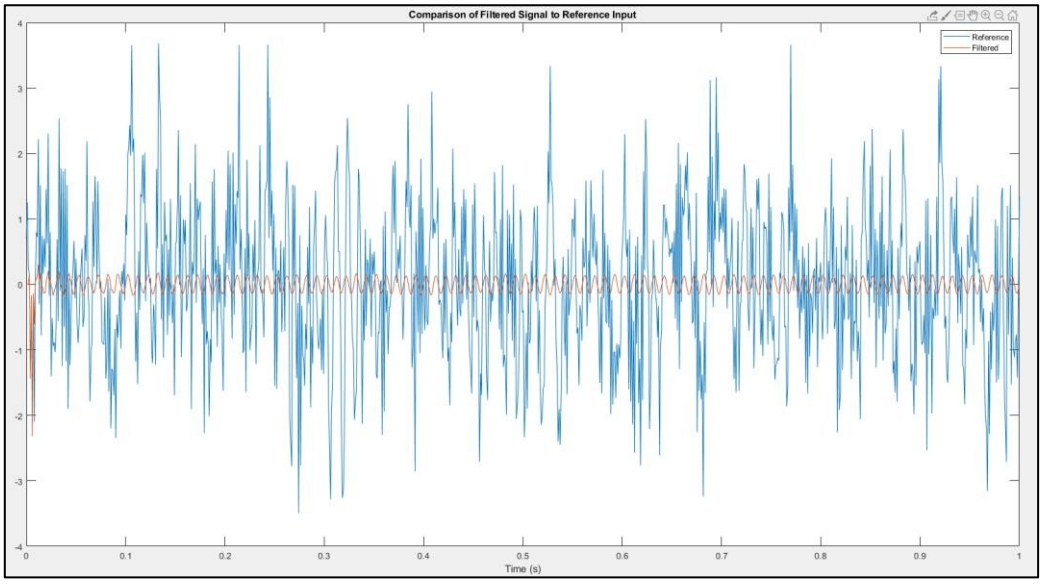

Figure 3.9: Comparison of the filtered signal to the distorted signal

Figure 3.9 shows the magnitude of the filtered signal is very small compared to the distorted signal.

3.4 Simulation Results Comparison and Implementation Decisions:

From the obtained results in sections (3.1.1-3.1.3), it was deduced that the RLS algorithm gives the best results for the lowest orders.

Based on the following reasons the final decision for the adaptive algorithm was made:

1. Instructor and examiner comments.
2. Lower order filters were prioritized.
3. Fastest convergence.
4. Stability.

LMS greatly lacks in terms of stability, and NLMS compromises error performance for stability.
Both LMS and NLMS require high order to function satisfactorily.

Based on the superior performance, the final decision for hardware implementation is to use RLS.

4 ARDUINO MICROCONTROLLER AND COMPLIMENTARY HARDWARE

4.1 Arduino

4.1.1 Arduino Microcontroller:

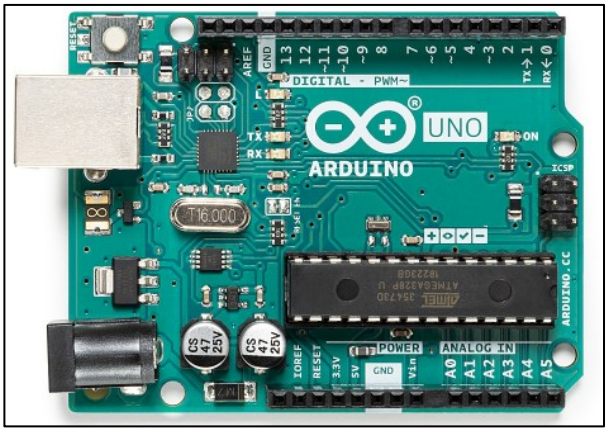

Figure 4.1: Arduino Uno microcontroller

Arduino (shown in figure 4.1) is an open source microcontroller designed and intended for educational purposes. The controller chip is an 8-bit processing unit. Arduino is most commonly used for mechatronics prototyping. Arduino is most commonly programmed in the Arduino IDE environment, using a slightly modified C code language.

Arduino was used to transmit and detect signals. The signals received by the Arduino is passed into Simulink to be filtered and displayed.

4.1.2 RF Transmitter/Receiver Units:

The transmitter/receiver modules used for the project (shown in figure 4.2) utilize Multilevel Amplitude Shift-Keying (M-ASK) to transmit digital data. The primary use for modules is the transmission and reception of text data. The center frequency of the transmission is 433 MHz.

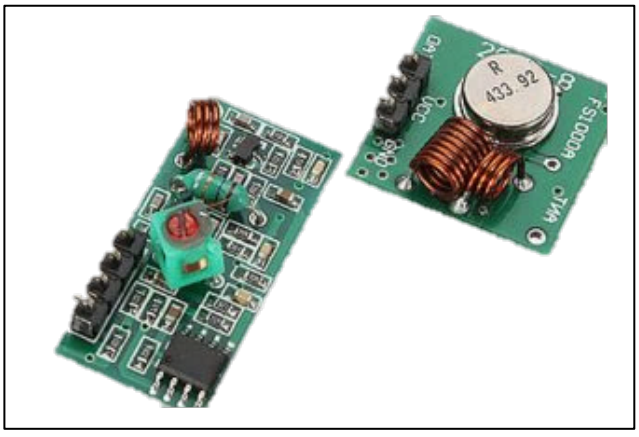

Figure 4.2: Arduino RF Transmitter/Receiver Units

4.2 Arduino Hardware Restrictions:

There some notable restriction that using Arduino imposed on the project.

The most notable issues are:

1. Arduino cannot receive *sinusoidal* data.
2. The M-ASK RF modules are only capable of passing the *envelope* of the transmitted pulses.
3. The transmission and reception of pulses are not perfect, as the pulses were not uniform, nor were they consistent, as seen in figure 4.3.
4. The RF modules are equipped with omnidirectional coil loop antennae, which makes controlling the radiation directivity difficult.

To overcome the issues the transmitting Arduino was programmed to transmit 1 second period pulse train with 50% duty cycle.

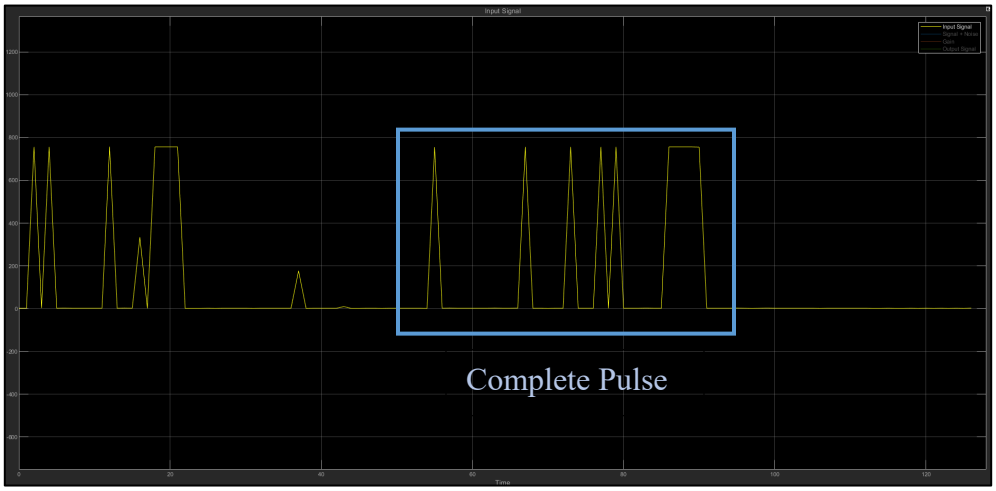

Figure 4.3:Complete discretized pulse.

Pulses received using the RF units are discretized in time due to noise, interference, and losses. Figure 4.3 demonstrates the pulse error.

5 DATA ACQUISITION AND SIMULINK FILTERING

5.1 Pulse Transmission:

The pulse transmitter circuit was constructed, as follows:

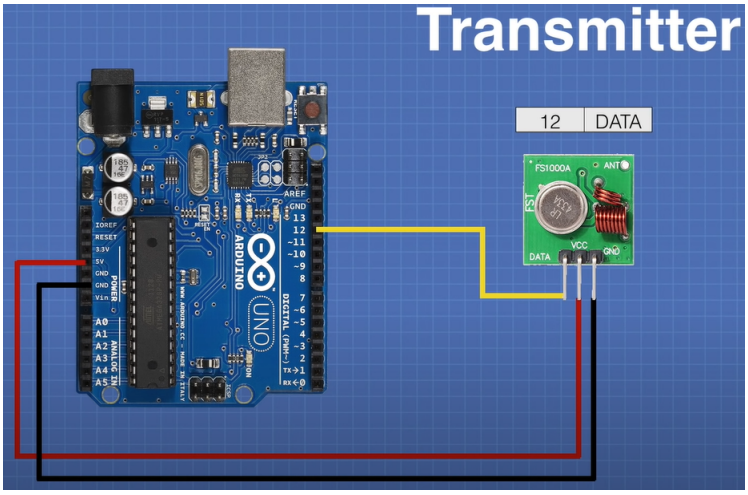

Figure 5.1: Transmitter Arduino unit

The Arduino was programmed to input a modulating signal of two states, where 5V represents high and 0V represents low.

The signal is modulated onto a 433 MHz carrier signal using the RF transmitter unit, as seen in figure 5.1.

5.2 Pulse Reception:

The pulse receiver circuit was constructed using a second Arduino, as in shown in figure 5.2:

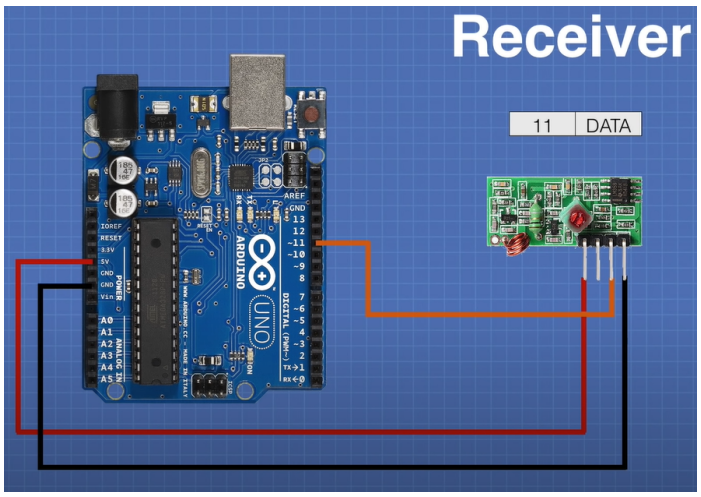

Figure 5.2: Receiver Arduino unit.

The receiver RF unit receives the modulated signal and independently demodulates to extract the pulses.

The Arduino (connected to the PC) passes the data to Simulink.

5.3 Simulink Schematic:

The Arduino hardware support package allows MATLAB and Simulink to acquire, process, and send data to/from Arduino.

Digital pins were used to acquire the data in this case, but it should be noted that analog pins are also useable.

The input and reference signals are inputted into a conversion block, which changes the variable to *double.*

The filtration process is done according to the equations in section 2.3.3.

Finally, the output, error, and filter tap weights are displayed.

Note: The amplifier preceding the output signal is used to compensate for amplitude decays (if any).

Filter taps are displayed in a separate window that shows the discrete values of filter weights.

The dynamic frequency response of the filter that results from the adjustments the RLS algorithm performs are also displayed on a separate window.

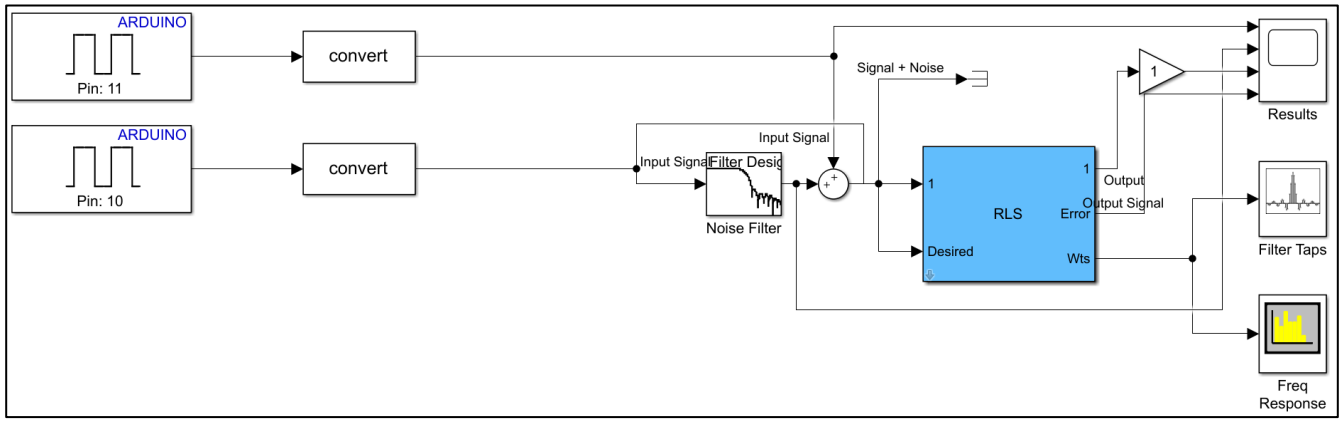

Figure 5.3: Simulink schematic of the filter system

As seen in figure 5.3, the reference signal is obtained from digital pin 11, while pin 10 provides the input signal to the system.

6 Design Concept and Hardware

6.1 Passive Radar using Arduino:

As deduced from hardware testing, only one Arduino can be connected to Simulink at one time, which imposes that the same Arduino system be used to obtain the input signal and the reference signal.

A passive radar requires the use of two antennae simultaneously, which Arduino, for the available hardware is incapable of.

Therefore, the following solution was proposed:

- The reference signal is transmitted via wire, which allows it to be received in near perfect form.
- The input signal is transmitted wirelessly.

The wireless transmission of the reference signal will distort and attenuate the signal due to ambient noise, interference, and processing.

The main idea behind the solution is that if the reference is accessible *via wire*, a highly effective passive radar system can be constructed and operated.

6.2 Hardware Setup:

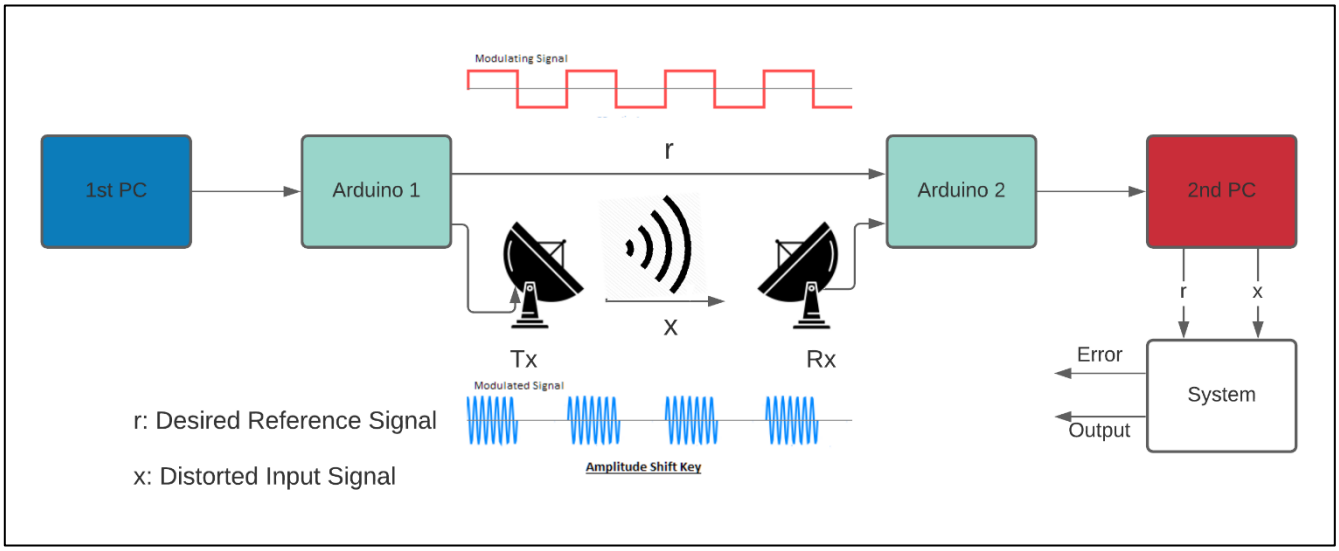

Figure 6.1: Block diagram schematic of hardware setup

Figure 6.1 shows the system block diagram in which the first and second PC's communicating using two channels, where two signals, a reference signal r and an input signal x, are transmitted via wire and wirelessly, respectively.

1. The first PC controls transmitter Arduino.
2. The transmitter Arduino generates a pulse train reference signal and transmits it directly (baseband transmission) via the wired channel
3. The transmitter Arduino modulates the pulse train onto a carrier of 433MHz center frequency, using Amplitude Shift Keying, and transmits it via the wireless channel.
4. The receiver Arduino receives the reference pulse train via wire.
5. The receiver Arduino receives the modulated pulse train and independently demodulates it.
6. The reference pulse train and input demodulated pulse train are then passed by the Arduino to the second PC.
7. The second PC uses the reference and input signals as inputs to the system shown in figure 5.3.
8. The reference signal obtained from pin 11 and the input signal is obtained using pin 10.

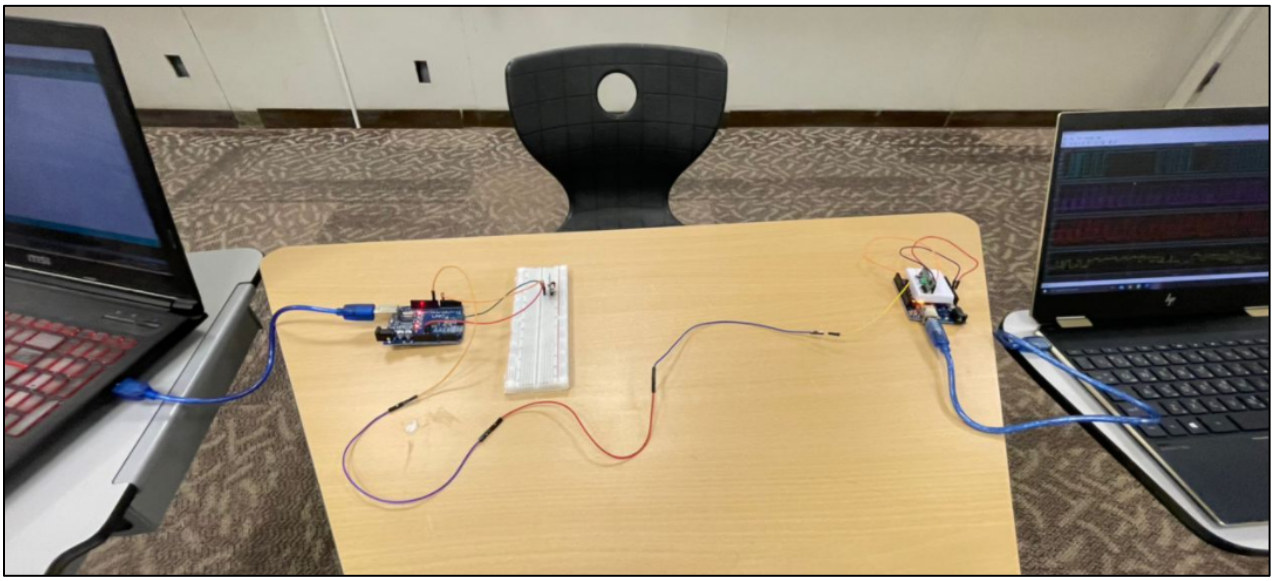

Figure 6.2: Hardware setup

Figure 6.2 shows the hardware setup used for the project, where two PC's, two Arduino's, wired channel, and wireless channel are used.

7 SIGNAL FILTRATION

7.1 Filtration Process:

The filtration process begins at the static noise filter, which is a 31st order Hamming window FIR filter.

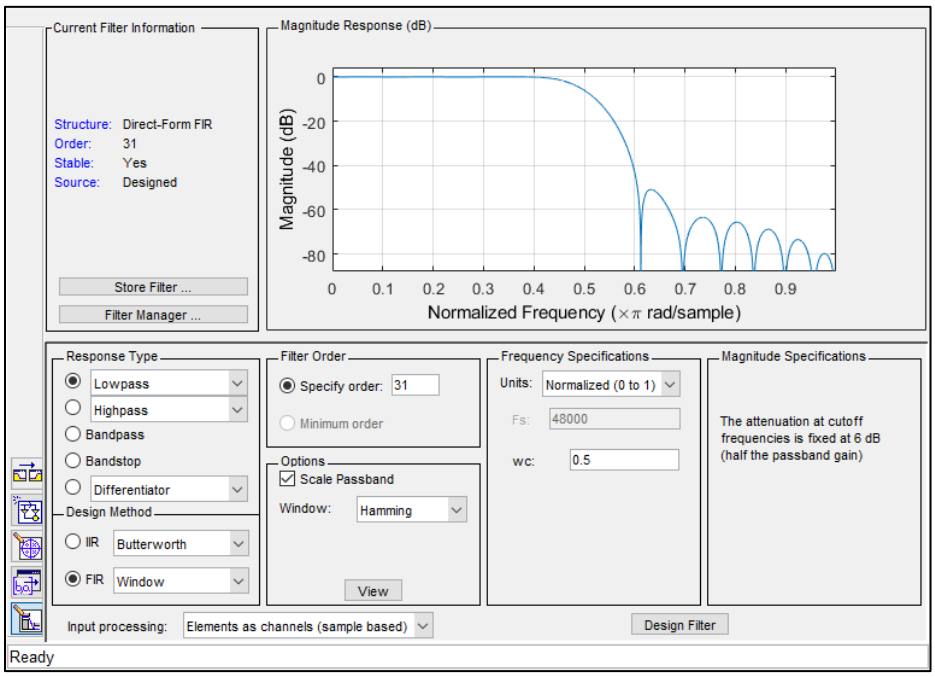

Figure 7.1: Static Filter Designer

Figure 7.1 shows the static filter design window.
The filter output is then summed with the input signal and passed into the RLS algorithm block.

7.2 System Test Run Results:

A test run of the program was recorded with the following results:

Filter frequency response:

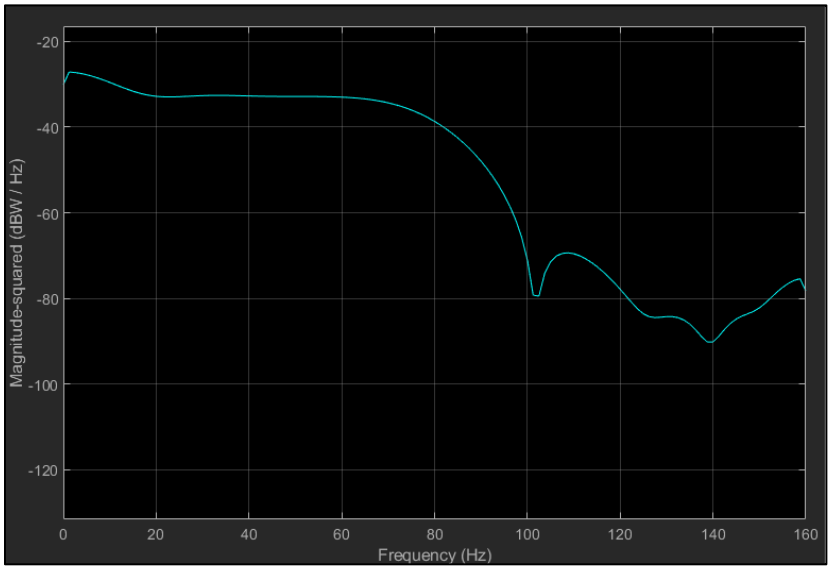

Figure 7.2: Filter frequency response.

The filter frequency response in figure 7.2 is dynamic in nature, as it keeps shifting value throughout the filtration process.
It should be noted that most of the changes occur in the transition band.

Filter Tap-Weights:

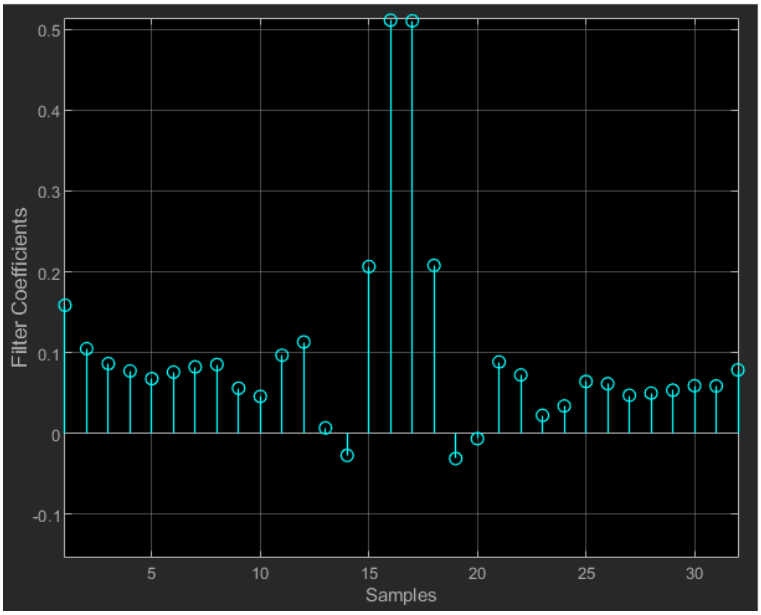

Figure 7.3: Filter tap weights

The filter tap-weights appearing in figure 7.3 are time variant, and keep changing weights until finally settling to a steady state.
Input/output Signals:

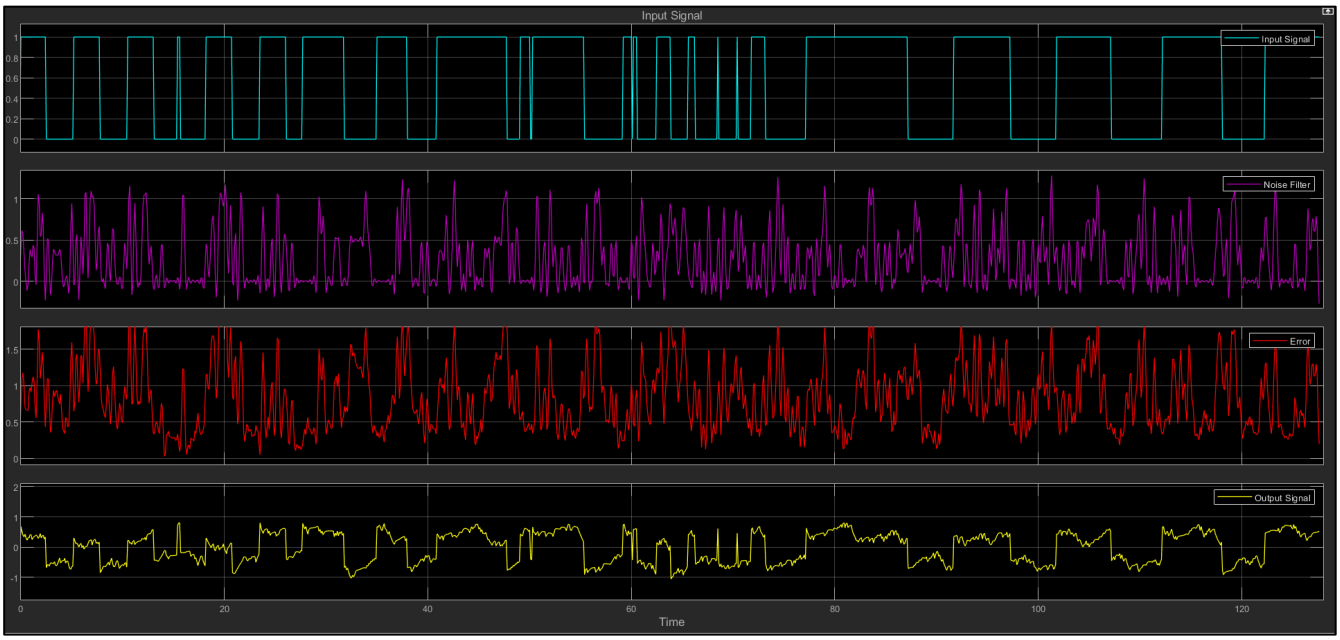

Figure 7.4: Blue: Reference Signal, Purple: Distorted Input Signal, Red: Error Signal, Yellow: Filtered Output Signal.

7.3 Observations:

The following observations were obtained from the results in figure 7.4

- For the used order (31) the output signal is correlated with the reference signal to an acceptable degree.
- The filtration is not perfect, as the output signal suffers from distortions.
- The observed dynamic changes in the frequency response of the filter was most noticeable in the transition band.
- A very sharp impulse was observed in the early points of simulation, that match the RLS simulation results in figure 3.8.
- The convergence speed was high, as expected from the RLS algorithm.

8 CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

The adaptive filtering process is not perfect and requires testing and optimization. By testing several algorithms, it was found that each algorithm has special traits and advantages; as found, the Least Mean Square Algorithm, in all its variations, is simple and easy to code, which allows for great alterations to be made to suit the needs of the user. However, it lacks stability and has to run at high orders if the reference signal is of high frequency, which means that it might not be suitable for passive radar applications in environments where the ambient radiation is of high frequency. In contrast, the Recursive Least Squares algorithm is much more stable and performs significantly better and faster at very low orders, as tested in this project. However, the complexity of the code makes it very difficult to alter and adjust to suit the environment. Which finally leads to the conclusion, that if the design is flexible in terms of speed, LMS is excellent for the simplicity and alterations it offers; whereas, if the design required high speed of convergence, the best algorithm to use would be RLS.

Hardware implementation testing showed that Arduino is an unsuitable choice for signal processing projects. The complementary hardware for Arduino projects is mainly sensors and mechatronics parts. Although hardware imposes difficulties on Arduino DSP designs, there various ways to overcome the issue and be able to process data according to the project needs.

The use of a wired channel to transmit the reference signal help overcome the main drawback of the Arduino based passive radar system. Using the wired reference has also made the filtration process easier, as the reference was highly protected from ambient interference.

If the ambient radiation source is accessible via, single antenna passive radars can be built and operated.

The passive radar simulation made in Simulink shows that the filtering process is simple to design and implement using schematic blocks. For a relatively low filter order, the Simulink filter operated successfully with high stability and quick convergence.

The results obtained indicate that low-cost passive radars can be built with educational hardware such as Arduino.

8.2 Recommendations for Future Work:

Future projects should focus on hardware capability and complementary devices. Arduino was found to be unsuitable for DSP applications, therefore other devices should be used. It is also worth noting that Simulink reduces design and testing work time significantly through the use of hardware support blocks (Arduino support package in this project). Future designs should attempt to implement the design using different filter algorithms.

9 REFERENCES

[1] Richards, M. A., Scheer, J. A., & Holm, W. A. (2010). *PRINCIPLES OF MODERN RADAR, BASIC PRINCAPLES*. Edison, NJ

[2] F. Colone, R. Cardinali and P. Lombardo, "Cancellation of clutter and multipath in passive radar using a sequential approach," 2006 IEEE Conference on Radar, Verona, NY, USA, 2006, pp. 1-7-, doi: 10.1109/RADAR.2006.1631830.

[3] Haykin, S. S. (1995). *Adaptive filter theory*. Upper Saddle River, NJ: Prentice Hall.

[4] Tan, L., & Jiang, J. (2019). *Digital signal processing: Fundamentals and applications* (3rd ed.). London: Elsevier, Academic Press.

[5] E. Eleftheriou and D. Falconer, "Tracking properties and steady-state performance of RLS adaptive filter algorithms," in IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 34, no. 5, pp. 1097-1110, October 1986, doi: 10.1109/TASSP.1986.1164950.

[6] J. L. Garry, G. E. Smith and C. J. Baker, "Direct signal suppression schemes for passive radar," 2015 Signal Processing Symposium (SPSympo), Debe, 2015, pp. 1-5, doi: 10.1109/SPS.2015.7168278.

10 APPENDICES

10.1 Appendix A. MATLAB Codes

10.1.1 Least Mean Square:

```
close all
clc
clear


fr=100e3;
Fs=100e5;
dt=1/Fs;
mu=0.001;
M=32;
w=zeros([1,M]);
v=0;
t=0:dt:0.001-dt;

d=sin(2*pi*fr*t);
N=randn(size(d));
x=d+N;

for i=1:length(t)
   e(i)=w(i-v)*x(i);
   y(i)=d(i)-e(i);
   w(i+1-v)=w(i-v)+mu*y(i)*x(i);
   q(i)=w(15);
   if (i-v)==M

      v=v+M;
   end
end


figure
subplot(4,1,1),plot(t,d);title('Desired Signal');xlabel('time(s)');
subplot(4,1,2),plot(t,x);title('Distorted Signal');
subplot(4,1,3),plot(t,e);title('Error Signal');xlabel('time(s)');
subplot(4,1,4),plot(t,y);title('Estimated Signal');xlabel('time(s)');

figure
plot(e.^2);xlabel('Iteration)');ylabel('Square Error');title('Square Error');
figure
plot(q);xlabel('Iterations');ylabel('Coeffecient W_1_5(n)');title('Convergence Curve');

display(mean(e.^2),'Mean Sqaure Error')

display(max(e),'Max. Error')
```

10.1.2 Normalized Least Mean Square:

```
close all
clc
clear


fr=100e3;
Fs=100e5;
dt=1/Fs;
B=0.001;
M=32;
w=zeros([1,M]);
v=0;
t=0:dt:0.001-dt;

d=sin(2*pi*fr*t);
N=randn(size(d));
x=d+N;

for i=1:length(t)
   e(i)=w(i-v)*x(i);
   y(i)=d(i)-e(i);
   w(i+1-v)=w(i-v)+B*y(i)*x(i)/(x(i)^2);
   q(i)=w(15);
   if (i-v)==M

      v=v+M;
   end
end


figure
subplot(4,1,1),plot(t,d);title('Desired Signal');xlabel('time(s)');
subplot(4,1,2),plot(t,x);title('Distorted Signal');
subplot(4,1,3),plot(t,e);title('Error Signal');xlabel('time(s)');
subplot(4,1,4),plot(t,y);title('Estimated Signal');xlabel('time(s)');

figure
plot(e.^2);xlabel('Iteration)');ylabel('Square Error');title('Square Error');
figure
plot(q);xlabel('Iterations');ylabel('Coeffecient W_1_5(n)');title('Convergence Curve');

display(mean(e.^2),'Mean Sqaure Error')

display(max(e),'Max. Error')
```

10.1.3 Recursive Least Squares:

```
function [e,w] = RLSFilterIt(n,x,fs)
close all;
clear all;
clc;
tic

%--------------------------------------------------------------------------
% Generate Data
%--------------------------------------------------------------------------

% Generate data if no inputs provided
if nargin < 1
   % Data Parameters
   numPts  = 1000;          % number of points to generate
   freq    = 100;           % frequency of fundamental tone
   filtord = 4;             % filter order
   filt    = rand(filtord, 1); % filter coefficients
   nVar    = 1;             % white noise variance
   SNR     = -20;           % signal to noise ratio of tone

   % Generate the data!
   [n,x,s,fs] = genData(numPts, freq, filt, nVar, SNR);

end

%--------------------------------------------------------------------------
% Filtering
%--------------------------------------------------------------------------

% Filter Parameters
p      = 4;              % filter order
lambda  = 1.0;           % forgetting factor
laminv  = 1/lambda;
delta   = 1.0;           % initialization parameter

% Filter Initialization

w      = zeros(p,1);     % filter coefficients
P      = delta*eye(p);   % inverse correlation matrix

e      = x*0;            % error signal

for m = p:length(x)

   % Acquire chunk of data
   y = n(m:-1:m-p+1);

   % Error signal equation
   e(m) = x(m)-w'*y;

   % Parameters for efficiency
   Pi = P*y;

   % Filter gain vector update
   k = (Pi)/(lambda+y'*Pi);
```

```
    % Inverse correlation matrix update
    P = (P - k*y'*P)*laminv;

    % Filter coefficients adaption
    w = w + k*e(m);

    % Counter to show filter is working
    %if mod(m,1000) == 0
    %    disp([num2str(m/1000) ' of ' num2str(floor(length(x)/1000))])
    %end

end

toc
max_error=max(abs(e))
e0=mean(abs(e).^2);
e0
w


%--------------------------------------------------------------------------
% Plot
%--------------------------------------------------------------------------

% Plot filter results
t = linspace(0,length(x)/fs,length(x));
figure;
plot(t,x,t,e);
title('Result of RLS Filter')
xlabel('Time (s)');
legend('Reference', 'Filtered', 'Location', 'NorthEast');
title('Comparison of Filtered Signal to Reference Input');

% Plot comparison of results to original signal (only for generated data)
if nargin < 1
    figure;
    plot(t,s,t,e);
    title('Result of RLS Filter')
    xlabel('Time (s)');
    legend('Signal', 'Filtered', 'Location', 'NorthEast');
    title('Comparison of Filtered Signal to Original Signal');
end


% Calculate SNR improvement
SNRi    = 10*log10(var(x)/var(e));

disp([num2str(SNRi) 'dB SNR Improvement'])

return

function [n,x,s,fs] = genData(numPts, freq, filt, nVar, SNR)

    % Generate time values
    t = linspace(0,1,numPts)';
```

```
    fs = numPts;

    % Generate tone
    s = sin(2*pi*freq*t);

    % Generate noise
    n = sqrt(nVar)*randn(numPts,1);

    % Filter noise
    addnoise = filter(filt, 1, n);

    % Plot filter
    freqz(filt,1,1000)

    % Adjust SNR of tone
    s = s/sqrt(var(s)/(10^(SNR/10)*var(n)));
    disp(['Calculated SNR = ' num2str(10*log10(var(s)/var(n)))])

    % Add noise to signal
    x = s + addnoise;

return
```