Matched Filter

Documentation for explaining UAVRT-SDR's current matched filter

Flagstaff, AZ
June 24, 2017

Edited by

Michael Finley
Northern Arizona University
Dynamic and Active Systems Laboratory

Contents

1 Project Description

The aim of this project is to construct an unmanned aerial system that integrates a radio telemetry receiver and data processing system for efficient detection and localization of tiny wildlife radio telemetry tags. Current methods of locating and tracking small tagged animals are hampered by the inaccessibility of their habitats. The high costs, risk to human safety, and small sample sizes resulting from current radio telemetry methods limit our understanding of the movement and behaviors of many species. UAV-based technologies promise to revolutionize a range of ecological field study paradigms due to the ability of a sensing platform to fly in close proximity to rough terrain at very low cost. The new UAV-based (UAV-RT) system will dramatically improve wildlife tracking capability while using inexpensive, commercially available radio tags. The system design will reflect the unique needs of wildlife tracking applications, including easy assembly and repair in the field and reduction of fire risk.

Our effort will focus on the development, analysis, real-time implementation, and test of software-defined signal processing algorithms for RF signal detection and localization using unmanned aerial platforms. It combines (1) aspects of both wireless communication and radar signal processing, as well as modern statistical methods for model+data inference of RF source locations, and (2) fusion-based inference that draws on RF data from both mobile UAV-based receivers and ground-based systems operated by expert humans.

The complete design, including parts, assembly plans, software, and training modules for the UAV-RT system will be open sourced for widespread adoption of the technology, with dissemination and outreach by a user community website and demonstrations at wildlife conferences.

1.1 Component Descriptions and Definitions

Airspy R2 Software Defined Radio (SDR) receiver, by Itead Studio (see: http://airspy.com/airspy-r2/)

Beacon Radio telemetry tag, provided by HOLOHIL (see: http://www.holohil.com/transmitters/tablebats)

LNA Low noise amplifier, provided by LNA4ALL

1.2 Document description

This document provides explanations for SNR calculations with the use of a software defined radio (SDR) receiver. This document will use the Airspy R2 as an example SDR receiver, but any SDR receiver should suffice assuming the correct drivers and dependencies can be acquired for usage with GNURadio. This document will also assume the use of GNURadio as a means of getting the actual I/Q data but many other SDR applications would work in its place (e.g. SDRSharp, etc.). This document will also be using MATLAB to calculate the SNR but open source programs like Octave will work as well. This document assumes the user has a raw binary I/Q data file readily available (see: I/Q Data Acquisition Guide) for processing.

1.2.1 Software

GNURadio GNURadio is a free and open source toolkit for software defined radio

GNURadio Companion GNU Radio Companion is a graphical user interface for GNURadio

MATLAB MATLAB, short for matrix laboratory, is a numerical computing environment

Octave Octave is a numerical copmuting environment very similar to MATLAB but open source

1.2.2 Hardware

Airspy R2 The Airspy R2 is an SDR receiver, specifications below

- 24-1800 MHz native RX range, down to DC with the SpyVerter option.
- 3.5 dB noise floor between 42 and 1002 MHz
- 10MSPS I/Q output, 16bit fixed or 32bit float ouput streams
- 2.5MSPS (experimental) I/Q output for low power devices
- 10MHz panoramic spectrum
- No I/Q imbalance, DC offset or 1/F noise
- SMA antenna connector
- Micro-B USB socket
- MCX connector for an external clock

2 Matched Filter Explanation

2.1 Introduction

The fundamental use of a matched filter in signal processing is the maximization of the signal-to-noise ratio in the presence of additive stochastic noise, or more specifically white Gaussian noise. The matched filter can be obtained by correlating a known signal, or template, with an unknown signal to detect the presence of the template in the unknown signal; detecting the presence of a target signal in noise. This correlation is equivalent to the convolving of an unknown signal with a conjugated time-reversed version of the template. Furthermore, and computationally more efficient, the fast-Fourier-transform convolution of a signal is equivalent to this convolution. The ladder approach can reduce processing time by up to one hundred times that of the alternative approaches. For the sake of this explanation we will illustrate the more computationally expensive, but more intelligible, methods.

2.2 Convolution

Convolution is a formal mathematical operation of two functions. In the context of signal processing, convolution is the process of creating a third signal from two separate signals. In the specific context of a matched filter, the convolution of a signal and a template result in the output of the matched filter. In linear systems convolution is used to describe the relationship between an input signal, the impulse response, and the output signal.

In the context of our system specifically: the input signal to our matched filter is a signal with an unknown presence of a target signal, the impulse response is the target signal or template, and the output signal is the output of the matched filter. The naming convention we will use for convolution in the context of our matched filter is detailed below.

The phrases impulse response, template, and target signal, are equivalent in the context of this matched filter, the phrase template will be used moving forward for ease of reading.

$$Y[n] = H[n] * X[n] \tag{1}$$

Where $Y[n]$ represents the discrete-time signal output of our matched filter, $H[n]$ represents the template of our matched filter (the template), and $X[n]$ represents the discrete-time signal input of our matched filter. It is also worth bringing attention to the use of $*$ symbolizing convolution.

The length of the output signal will be equal to the length of the input signal plus the length of the template minus 1.

$$N_Y = N_X + N_H - 1 \tag{2}$$

Where N_y represents the number of samples in the matched filter output, N_H represents the number of samples in the template, and N_X represents the number of samples of the input signal. The length of the convolution is necessary for initializing an output array filled with zeros to be populated with the sums of multiplications from the input signal and the template.

2.3 Matched Filter Derivation

Now that a convolution has been properly defined we can approach the application of convolution to develop a matched filter. Given an input signal $x'(t)$ consisting of both target and noise components, the output of the matched filter is given by the convolution

$$y(t) = \int_{-\infty}^{\infty} x'(s)h(t-s)ds = \int_{-\infty}^{\infty} x'(s)x^*(s+T_M-t)ds \tag{3}$$

The second portion of equation (3) represents the cross-correlation of the target-plus-noise signal $x'(t)$ with the transmitted waveform $x(t)$, evaluated at lag $T_M - t$. Thus, the matched filter implements a correlator with the transmitted waveform as the reference signal.

It can be shown that the maximum achievable SNR depends only on the energy of the waveform and not on other details such as its modulation. It is also worth pointing out that the duration of the signal component of the matched filter output is exactly 2τ seconds, since it is the convolution of the τ-second pulse with the τ-second matched filter impulse response (template). To illustrate the previous ideas in the context of our system, consider a simple pulse of duration τ:

$$x(t) = \begin{cases} 1 & 0 \le t \le \tau \\ 0 & otherwise \end{cases} \tag{4}$$

The corresponding matched filter impulse response (template) is

$$x(t) = \begin{cases} A & t \le T_M - t \\ 0 & otherwise \end{cases} \tag{5}$$

Where $T_M > \tau$ for causality. Because $x(t)$ is a much simpler function than its fourier transform, a sinc function, it is computationaly easier to work with the correlation interpretation of equation (3) to compute the output. Figure 2 illustrates the two terms in the integrand, helping estbalish the regions of integration. Part a of the figure shows that

$$y(t) = \begin{cases} 0 & t \le T_M - t \\ \int_0^{t-T_M+\tau} (1)(A)ds & T_M - \tau \le t \le T_M \end{cases} \tag{6}$$

while part b is useful in identifying the next two regions

$$y(t) = \begin{cases} \int_{t-T_M}^{\tau} (1)(A)ds & T_M \le t \le T_M + \tau \\ 0 & t \ge T_M + t \end{cases} \tag{7}$$

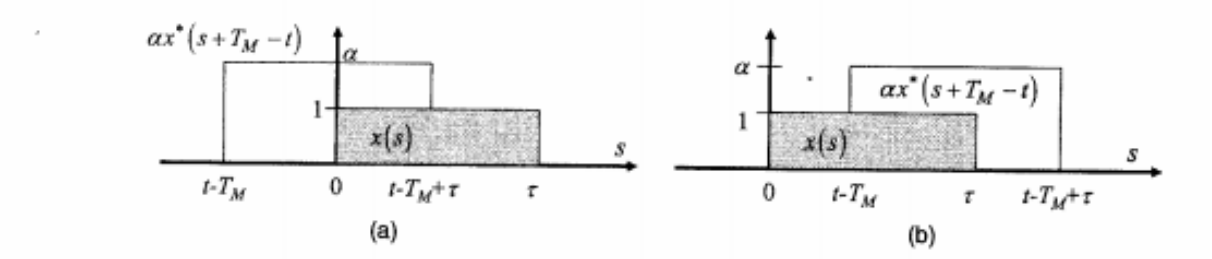

Figure 1: Convolution of simple pulse and its matched filter: (a)$T_M - \tau \leq t \leq T_M$, (b) $T_M \leq t \leq T_M + \tau$

Lastly, the matched filter output for a simple pulse visually:

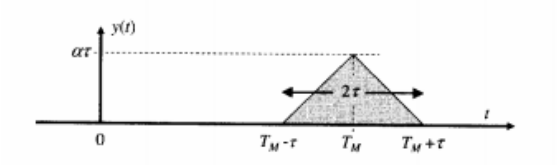

Figure 2: Matched filter output of a simple pulse

The final result is:

$$y(t) = \begin{cases} A[t-(T_M-\tau)] & T_M-\tau \leq t \leq T_M \\ A[(T_M+\tau)-t] & T_M \leq t \leq T_M+\tau \\ 0 & otherwise \end{cases} \tag{8}$$

The result illustrated in Figure 2 shows the matched filter output is a triangle function of duration 2τ secibds wutg uts oeaj at $t = T_M$ as expected. The peak value is $A\tau$; since the energy of the unit amplitude pulse is just τ, the peak value equals AE as predicted. What follows is that the SNR is independent of anything else but energy, as noted before.

2.4 Matched Filter Convolution

As an example of how convolution works we will illustrate a small-sample-size example. Again, $X[n]$ represents an input signal with 9 sample points, $H[n]$ represents the template of a signal to be looked for in the signal, and $Y[n]$ represents the output of the matched filter. The result is:

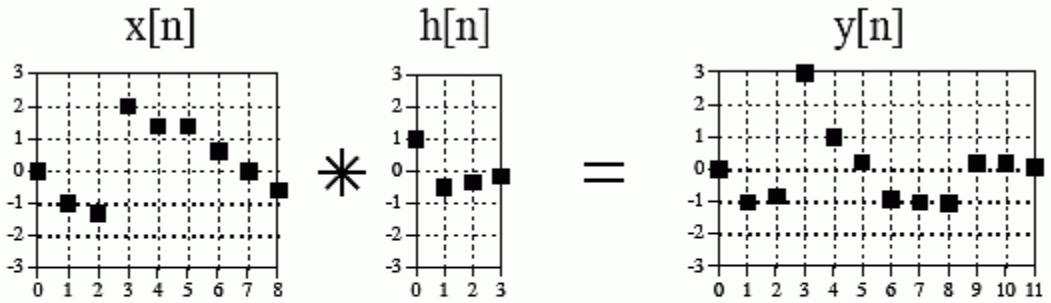

FIGURE 6-5
Example convolution problem. A nine point input signal, convolved with a four point impulse response, results in a twelve point output signal. Each point in the input signal contributes a scaled and shifted impulse response to the output signal. These nine scaled and shifted impulse responses are shown in Fig. 6-6.

The above figure 6-5 shows the final results of a convolution, the figure 6-6 below illustrates each step in the convolution process. The mathematics of this process will be explained in the next section, algorithmic explanation. Figures courtesy of [1]. In the figure below squares represent non-zero contributions to the matched filter output and diamonds represent zeroes, or place holders. The final result of the convolution in figure 6-5 is the summation of all of the multiplications seen in figure 6-6 below.

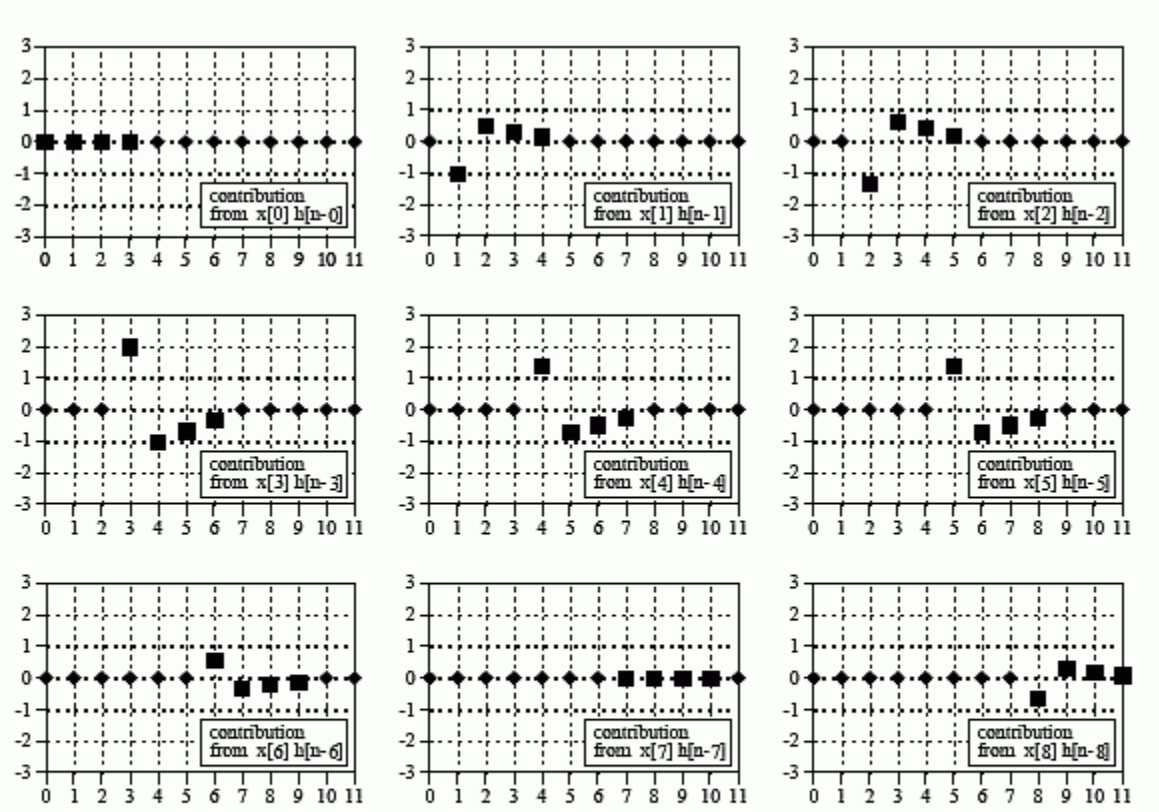

FIGURE 6-6
Output signal components for the convolution in Fig. 6-5. In these signals, each point that results from a scaled and shifted impulse response is represented by a square marker. The remaining data points, represented by diamonds, are zeros that have been added as place holders.

One way to comprehend the matched filter is to visualize the template sliding along a signal and based on the level of correlation the matched filter output increases. Remember though each time the template "slides" along to a new index of the input signal, each index of the template is multiplied by that index of the signal, so there is effectively two "slides" going on.

2.5 Algorithmic Explanation

An algorithm which will be used to create a functional MATLAB program for the matched filter is listed below. This algorithm assumes the use of input signals and templates without any formal 'readings' of the necessary files. This algorithm just illustrates the functional pieces of the matched filter.

```
signal = signal %declaring signal
template = template %declaring template
length_template = length(template); %length of template
length_signal = length(signal); %length of signal
length_convolution = length_signal + length_template - 1; %length of convolution

for I = 0:lenth_signal %zeroing mf_output array
    mf_output(I) = 0;
end

```

```
11 for J = 0:length_signal %iteration loop for each index of input signal
12     for J = 0:length_template %iteration loop for each index of the template
13         mf_output(I + J) = mf_output(I + J) + (signal(I) * template(J));
14     end
15 end
```

This program effectively implements the matched filter. The iteration loop that occurs on lines 11 to line 15 can be difficult to understand. The inner loop is responsible for running through each index of the template doing three things. First the template response is scaled by multiplying it by the value of the input sample (see: signal(I) * template(J)), second the scaled impulse response is shifted the length of the template to the right by adding this number to the index used for the output signal (see: mf_output(I + J)), third mf_output must accumulate, sum, all the signals resulting from each sample in the input signal, therefor the new information must be added to the information that is already in the array (see: mf_output(I + J) = mf_output(I + J) + (signal(I) + template(J))).

2.6 Matched Filter Example Output

Applying the function detailed in section 4, MATLAB Processing, produces the outputs visualized below.

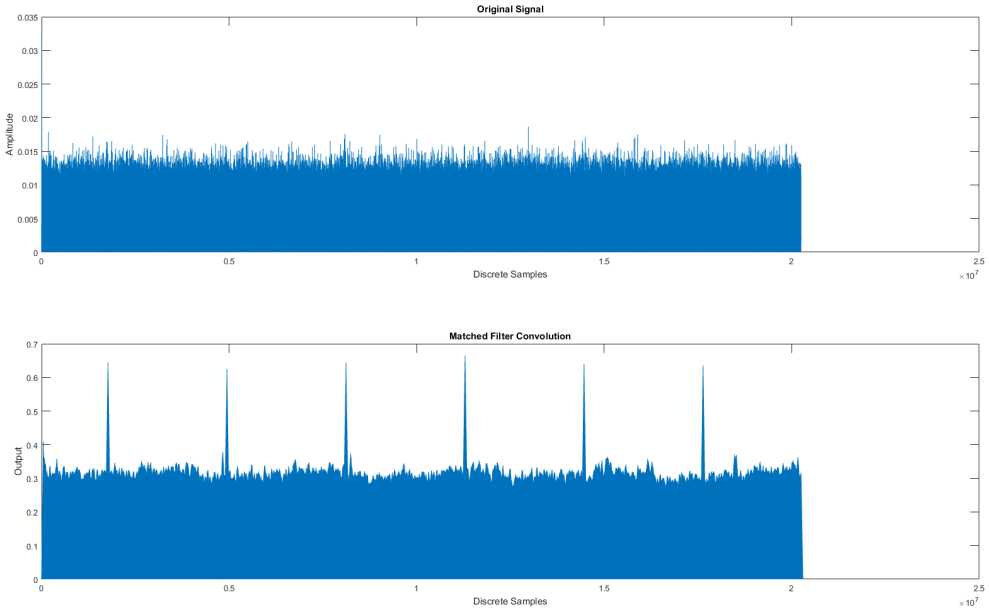

Figure 3: Showcasing visibly buried pulses being detected by the matched filter

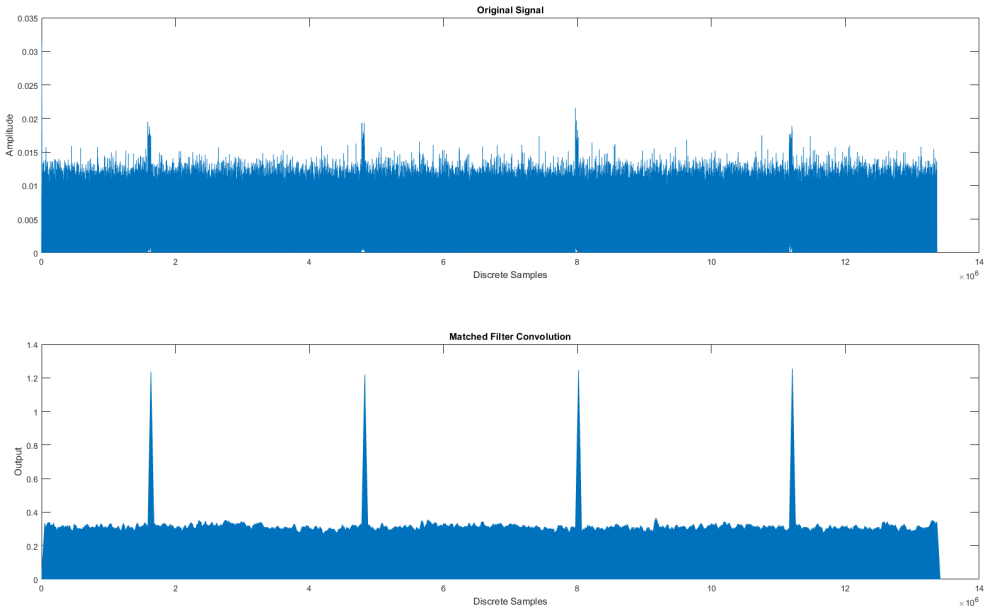

Figure 4: Showcasing visible pulses being detected by the matched filter

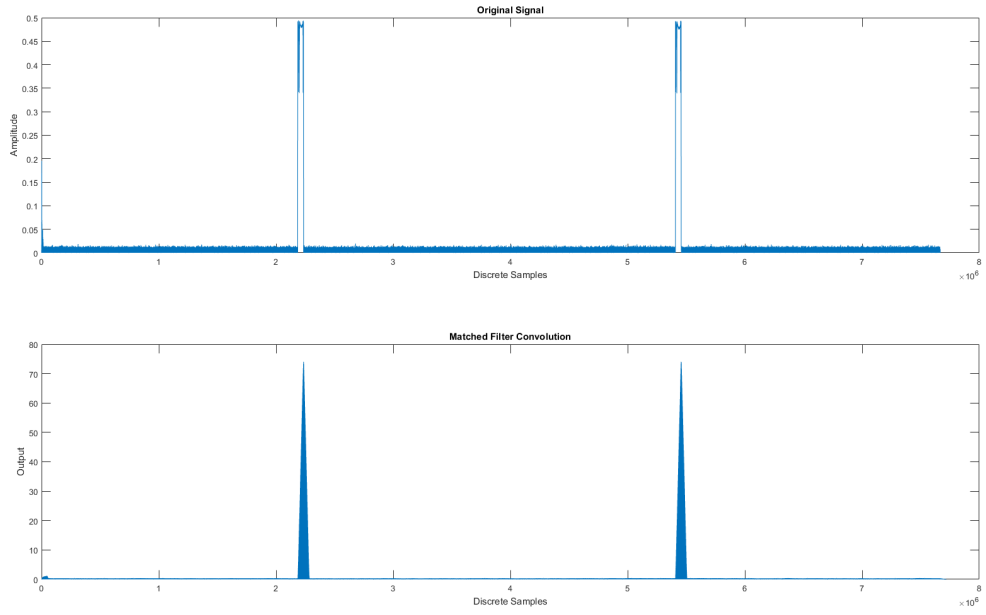

Figure 5: Showcasing visible pulses being detected by the matched filter

2.7 References

[1]S. W. Smith, "The Scientists and Engineers Guide to Digital Signal Processing," pp. 107-115, June 1997.

3 Methods/Procedures

3.1 Installing Necessary Software

3.1.1 MATLAB

To install MATLAB reference mathwork's installation guides (see: https://www.mathworks.com/support/install-matlab.html). Installing MATLAB requires a valid software license, which can be obtained by purchasing the product from the MathWorks Store (see: https://www.mathworks.com/store/) or downloading a product trial.

3.1.2 Octave

If MATLAB is not available to the user, Octave is a free open source numerical computing environment with similar syntax to MATLAB. To install Octave on a Windows machine you will want to download the Windows binaries with corresponding source code (see: https://www.gnu.org/software/octave/download.html) and navigate to the following link (see: https://ftp.gnu.org/gnu/octave/windows/) and select the user's preferred installation method. The "octave-4.2.1-w64-installer.exe" is recommended (as of 3/5/2017, this is Octave's newest stable version).

All software installations are now complete for calculating SNR.

4 MATLAB Processing

After opening MATLAB navigate to the directory in which the I/Q data file was saved. In order for the user to use or begin processing the I/Q data for pulse detection. The function used for this is included below.

The following MATLAB command would save `matched_filter_output` to your workspace as a float variable containing the discrete-time signal with the matched filter output.

```
matched_filter_output = matched_filter_output('file_sink_tutorial', template);
```

The desired function is detailed below for reference:

```
function Y = matched_filter_output (filename, template);
  signal = abs(read_complex_binary(filename)); %reading IQ file
  signal = abs(signal);

    function v = read_complex_binary (filename, count) %read IQ function
        m = nargchk (1,2,nargin);
        if (m)
        usage (m);
        end

        if (nargin < 2)
        count = Inf;
        end

        f = fopen (filename, 'rb'); %opening raw binary file
        if (f < 0) %checking for data
         v = 0;
        else
        t = fread (f, [2, count], 'float'); %reading as a float
```

```
        fclose (f); %closing file
        v = t(1,:) + t(2,:)*i; %assuming IQIQIQ structure of binary file
        [r, c] = size (v); %creating matrix to match IQ file size
        v = reshape (v, c, r); %reshaping matrix to correct format
        end

    end

len_signal = length(signal); %length of signal
len_template = length(template); %length of template
len_total = len_signal + len_template - 1; %total length of convolution

for I=1:len_total %zero output array loop
    Y(I)=0;
end

%the number 100 in the loops below can be changed to ease processing power
for I=1:100:len_signal %iterate for each point in input singal
    for J=1:100:len_template %iterate for each point in template
        Y(I + J) = Y(I + J) + (signal(I) * template(J)); %convolution
    end
end

hold on
subplot(2,1,1)
plot(real(signal))
hold on
%plot(imag(signal))
title('Original Signal')
xlabel('Discrete Samples')
ylabel('Amplitude')
subplot(2,1,2)
plot(real(Y));
hold on
%plot(imag(Y(len:end)))
title('Matched Filter Convolution')
xlabel('Discrete Samples')
ylabel('Output')
hold off

Y = Y;
end
```

Essentially this function calls a nested function that opens the desired file, designates that is raw binary, reads the raw binary file as a float, closes the file, then parses the data assuming it is structured like I1Q1I2Q2I3Q3... until the end of the file is reached. Then the function reshapes it as desired. What is left is an output in complex form with I being the real part and Q being the imaginary part in its complex representation.

The function then brings to fruition equation (1) and plots the matched filter output in comparison to that of the original signal showcasing the maximized SNR visually.

5 Recommendations

Comparisons of the various equivalent approaches detailed in the introduction to the matched filter would hopefully allow for a more robust understanding of what is going on with a matched filter.