

Final Project Report

18-758 Wireless Communications

Michael Nye (mnye)

System Design

Modulation

For my final project, I designed a system fairly similar to what was shown in lectures. For coding, I used a rate-1/2 convolution code with 4 states. After coding, the symbol bits are interleaved by a factor of 132 (chosen to be a large integral factor of my packet size). Again, testing of my system has demonstrated that this coding scheme sufficiently decouples error events and allows robust correction given my channel.

I use an 8-PSK modulation scheme. My choice of eight points was motivated by empirical measurements of the channel; I used the highest order constellation I could support without the noise causing symbol errors. I then used a hamming window pulse. The hamming window was chosen as it has better bandwidth properties than other FIR windows, but was easier to implement than a multisymbol pulse such as a raised cosine rolloff pulse. Testing showed that this pulse was sufficient to meet the project specifications.

Finally, I prepend my message with a single pilot sequence to facilitate equalization and timing synchronization. The sequence is created by generating a bit sequence. This sequence is a De Bruijn sequence, which creates a large variety in symbol ordering to create a unique sequence. This sequence is then modulated using a wide BPSK for easy detection.

Constant factors were chosen to be as large as possible while still meeting channel requirements and message size. Since I was fixed to rate-1/2 coding and 3 bits per symbol, this meant to fit my entire message (3036 pixels) in the allowed space, I was able to use at most 4 samples per symbol. I then expanded my pilot sequence to fill most of the remaining space.

Demodulation

My demodulator largely follows the reverse of my modulator. First off, I perform carrier recovery. To do so, I compute a DTFT on my received signal over a range of only low frequencies. I find the maximum amplitude frequency, and then divide by a complex sinusoid of the same frequency.

This is followed by timing recovery and equalization. I find the correlation between my pilot signal and the recovered signal. The maximum lag is the start of my pilot sequence. I can then extract the pilot sequence, and detect the pilot symbols. The channel equalization constant is then found by comparing the detected symbols to the known pilot symbols. Finally, the received signal is divided by this same factor to equalize.

At this point, I can window out only my message. I match filter the message, then sampling the result and perform hard detection of the coded bits. After deinterleaving these detected bits, I find the minimum error path through my coding trellis to correct any bit errors in the coded bits. Finally, the corrected coded bits are decoded into my message bits and returned.

Analysis

My system largely follows the principles demonstrated in lectures. There are three main differences I chose to make to simplify the system.

The first difference was my choice of pilot. I chose to use the same pilot message for carrier recovery, timing recovery and equalization. My testing found that this didn't cause any degradation of performance, and allowed me to minimize the size of my header.

The second difference was my choice of window. While raised cosine rolloff windows are clearly superior to hamming windows in terms of bandwidth properties, I found it difficult to correctly design the window to prevent ISI. A hamming window was much simpler to implement and performed admirably for my purposes.

The final difference was in how I performed carrier recovery. While we were encouraged to use a BPSK pilot sequence, I found this didn't provide sufficient resolution to find the carrier offset frequency. I instead just used the entire received signal, and bandlimited the region I search for the peak in the frequency domain. This proved surprisingly effective, and so I stuck with it.

The other difficulties I ran into were simply implementation details. Until this project, I had never used the MATLAB `'` operator to transpose complex data. I learned that this was a hermitian transpose, but it lingered in dark corners of my code base, and took quite a while to track down the errors. I also spent a lot of time determining exact offsets for my sampling times. This complexity was simply an artifact of using convolution with MATLAB vectors, and the resulting offsets it introduces into my data.

MATLAB Implementation

On the following pages, I have attached the code used to implement my communications system. There are three files. The first is `constants.m`, and contains just a set of constants referenced in my modulator and demodulator. The second is `generate transmit signal.m`, and contains a function that given a bit sequence will return a transmit signal. Finally, `decode received signal.m` takes in a received signal and performs recovers the original message in the transmit signal.

```

1 %
2 % constants.m
3 %     Defines the set of constants to be used for the comm system
4 %
5
6 % Constants defined by the radio
7 Fs = 2e6;      % transmit rate of USRP in Hz
8 maxL = 10000; % max samples in output signal
9
10
11 % Transmit constants
12 T = 4; % samples per symbol
13 B = 3; % bits per symbol
14 L = 3036; % packet size in bits
15
16 % Coding constants
17 coded = true; % enable
18 R = 2; % coded bits per data bit
19 interleaveA = 132; interleaveB = R*23; % Factors of interleaving
20
21 if ~coded, R=1; end;
22
23
24 % Hamming pulse for modulation. Normalize to unit energy
25 pulse = hamming(T)';
26 pulse = pulse/norm(pulse);
27
28
29 % De Bruijn sequence of order 5 for timing sequence
30 % Using a BPSK modulation
31 pilotBits = [0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, ...
32             0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1];
33
34 pilotT = 20;
35 pilotPulse = ones(1,pilotT/2); pilotPulse = pilotPulse/norm(pilotPulse);
36
37 pilot = upsample(2*pilotBits-1,pilotT);
38 pilot = conv(pilot,pilotPulse);
39
40
41 % Channel constants for simulation
42 SNR = 10; % dB
43 Ex = 1; % expected symbol energy
44 sigN = Ex / 10.^(SNR/10); % noise variance
45
46 maxdelay = 500; % max delay before transmit in samples
47 atten = [40 60]; % factor of attenuation during transmit

```

```

1  function x = create_transmit_signal(bits, plots)
2      % Load constants
3      constants;
4
5      % Fill variables
6      if nargin < 2
7          plots = false;
8      end
9
10     % Pad bits to full length
11     if(length(bits) > L)
12         error('Provided packet exceeds max packet size');
13     end
14     N = length(bits);
15     bits = [bits zeros(1,L-N)];
16
17
18     % Display message, pulse and pilot
19     if plots
20         figure(1); clf(1);
21         subplot(4,1,1);
22         stem(bits(1:N));
23         title('Uncoded message');
24     end
25
26
27     % Generate coded bits from the input
28     % Uses a 4-state rate 1/2 convolutional code
29     if coded
30         codedbits = zeros(1,R*length(bits));
31         state = [0 0];
32         for ii=1:length(bits)
33             % Get the current data bit
34             bit = bits(ii);
35
36             % Add the coded bits depending on the state
37             if isequal(state, [0 0])
38                 codedbits(2*ii-1 : 2*ii) = xor(bit, [0 0]);
39             elseif isequal(state, [0 1])
40                 codedbits(2*ii-1 : 2*ii) = xor(bit, [1 0]);
41             elseif isequal(state, [1 0])
42                 codedbits(2*ii-1 : 2*ii) = xor(bit, [1 1]);
43             else % isequal(state, [1 1])
44                 codedbits(2*ii-1 : 2*ii) = xor(bit, [0 1]);
45             end
46
47             % Update the state
48             state = [state(2) bit];
49         end
50
51         % Interleave the coded bits
52         codedbits = reshape(codedbits,interleaveA,interleaveB).';
53         codedbits = codedbits(:).';
54     else
55         codedbits = bits;
56     end
57
58     if plots
59         subplot(4,1,2);
60         stem(codedbits(1:R*N));
61         title('Coded bits');
62     end

```

```

63
64 % Map coded bits to symbols
65 % Uses 16PSK
66 M = 2^B;
67 ints = bi2de(reshape(codedbits,B,length(codedbits)/B).').';
68 syms = exp(2*pi*1j*ints/M);
69
70
71 % Expand in time and convolve with pulse sequence
72 x = upsample(syms,T);
73 x = conv(x,pulse);
74
75
76 % Place pilot sequence at head
77 x = [pilot x];
78
79 if plots
80     subplot(4,1,3); hold on;
81     plot(imag(x),'g'); plot(real(x));
82     plot(pilot,'c');
83     title('Transmit signal');
84     legend('x^Q', 'x^I', 'Pilot', 'Location','NorthWest');
85     stem(0,'marker','none');
86
87     spec = fftshift(fft(x));
88     subplot(4,3,10:11); hold on;
89     plot(linspace(-pi,pi,length(spec)),20*log10(abs(spec)+.01));
90     title('Transmit spectrum');
91     xlabel('\omega'); ylabel('Spectral power (dB)');
92
93     subplot(4,3,12); hold on;
94     plot([-2 2],[0 0], 'k');
95     plot([0 1e-10], [-2 2], 'k');
96     scatter(real(syms), imag(syms), 'bx');
97     axis([-1.5 1.5 -1.5 1.5]); axis square;
98     title('Signal space');
99     xlabel('x^I'); ylabel('x^Q');
100 end
101
102 % Normalise x to maximize power
103 x = x/max(abs(x));
104
105 % Error if x is too large
106 if(length(x) > maxL)
107     error('Computed message exceeds maximum message size');
108 end
109 end

```

```

1 function bits = decode_received_signal(y, plots, len)
2     % Load constant factors and prep to display
3     constants;
4
5     % Fill variables
6     if nargin < 2
7         plots = false;
8     end
9     if nargin < 3
10        len = L; %#ok
11    end
12
13
14    % Display received signal
15    if plots
16        figure(2); clf(2);
17        subplot(4,1,1); hold on;
18        plot(imag(y),'g'); plot(real(y));
19        legend('y~Q', 'y~I');
20        title('Raw received signal');
21        stem(0,'marker','none');
22
23        figure(3); clf(3);
24        subplot(2,1,1); hold on;
25        spec = fftshift(fft(y));
26        plot(linspace(-pi,pi,length(spec)),20*log10(abs(spec)));
27        title('Signal Spectra');
28        xlabel('\omega'); ylabel('Spectral power (dB)');
29    end
30
31
32    % Perform carrier recovery, using DTFT to estimate frequency offset
33    ws = linspace(-pi/1000,pi/1000,1500);
34    [~,I] = max(abs(dtft(y, ws)));
35    y = y .* exp(-1j*(1:length(y))*ws(I));
36
37
38    % Determine offset
39    [corrs,lags] = xcorr(y,pilot, 500);
40    [~,I] = max(abs(corrs));
41    delta = lags(I);
42
43    if plots; figure(2); scatter(delta,0,'r.');
```

```

44    end
45
46    % Match filter and sample the pilot to equalize
47    p = y(delta+1 : delta + length(pilot));
48    p = filter(pilotPulse(end:-1:1), 1, p); %#ok
49    zs = p(pilotT/2:pilotT:end);
50    ps = 2*pilotBits - 1;
51    eq = (ps*transpose(zs))/(ps*ps');
52
53
54    % Drop the offset, pilot and trailing end
55    L = min(L,len)*R/B; % number of symbols to read in
56    delta = delta + length(pilot);
57    y = y(delta+1 : delta + T*L);
58
59    if plots
60        subplot(4,1,2); hold on;
61        plot(imag(y),'g'); plot(real(y));
62        legend('y~Q', 'y~I');
```

```

63         title('Windowed signal, post-carrier recovery');
64         stem(0,'marker','none');
65     end
66
67
68     % Equalize
69     y = y/eq;
70
71     if plots
72         subplot(4,1,3); hold on;
73         plot(imag(y),'g'); plot(real(y));
74         title('Equalized signal');
75         legend('y~Q', 'y~I');
76         stem(0,'marker','none');
77     end
78
79
80     % Match filter, grab inphase and quadrature components, sample
81     y = conv(y, pulse(end:-1:1)); %#ok
82     yi = real(y);      yq = imag(y);
83     zi = yi(T:T:end); zq = yq(T:T:end);
84
85     if plots
86         subplot(4,1,4); hold on;
87         plot(yq, 'g'); plot(yi);
88         stem(T:T:length(y), zq, 'cx');
89         stem(T:T:length(y), zi, 'ro');
90         title('Filtered signal');
91         legend('y~Q', 'y~I', 'z~Q', 'z~I');
92         stem(0,'marker','none');
93
94         figure(3);
95         subplot(2,1,1); hold on;
96         spec = fftshift(fft(y));
97         plot(linspace(-pi,pi,length(spec)),20*log10(abs(spec)),'r');
98         legend('Received spectrum','Filtered spectrum');
99
100        subplot(2,1,2); hold on;
101        plot([-2 2],[0 0], 'k');
102        plot([0 1e-10], [-2 2], 'k');
103        scatter(zi, zq, 'bx');
104        scatter(real(exp(2*pi*1j*(0:(2^B-1))/2^B)), ...
105                imag(exp(2*pi*1j*(0:(2^B-1))/2^B)), ...
106                'ro', 'MarkerFaceColor','r');
107        axis([-1.5 1.5 -1.5 1.5]); axis square;
108        title('Equalized signal space');
109        xlabel('z~I'); ylabel('z~Q');
110    end
111
112
113    % Hard detect the coded symbols from MPSK
114    M = 2^B;
115    angs = angle(zi + 1j*zq);
116    decs = mod(round(angs*M/(2*pi)),M);
117    detectedbits = de2bi(decs,B).';
118    detectedbits = detectedbits(:).';
119
120
121    % Correct the coded bits using viterbi
122    if coded
123        % Deinterleave the received bits
124        detectedbits = reshape(detectedbits,interleaveB,interleaveA).';

```

```

125     detectedbits = detectedbits(:).';
126
127
128     % Create empty trellis
129     oldtrellis = struct('errors', {0, Inf, Inf, Inf}, ...
130                        'bits', {[], [], [], []});
131     newtrellis = struct('errors', {0, 0, 0, 0}, ...
132                        'bits', {[], [], [], []});
133
134     % Score the whole trellis
135     for ii=1:2:length(detectedbits)
136         bits = detectedbits(ii:ii+1);
137
138         % State 00
139         error1 = oldtrellis(1).errors + sum(bits ~= [0 0]);
140         error2 = oldtrellis(3).errors + sum(bits ~= [1 1]);
141         if error1 < error2
142             newtrellis(1).errors = error1;
143             newtrellis(1).bits = [oldtrellis(1).bits 0 0];
144         else
145             newtrellis(1).errors = error2;
146             newtrellis(1).bits = [oldtrellis(3).bits 1 1];
147         end
148
149         % State 01
150         error1 = oldtrellis(1).errors + sum(bits ~= [1 1]);
151         error2 = oldtrellis(3).errors + sum(bits ~= [0 0]);
152         if error1 < error2
153             newtrellis(2).errors = error1;
154             newtrellis(2).bits = [oldtrellis(1).bits 1 1];
155         else
156             newtrellis(2).errors = error2;
157             newtrellis(2).bits = [oldtrellis(3).bits 0 0];
158         end
159
160         % State 11
161         error1 = oldtrellis(2).errors + sum(bits ~= [1 0]);
162         error2 = oldtrellis(4).errors + sum(bits ~= [0 1]);
163         if error1 < error2
164             newtrellis(3).errors = error1;
165             newtrellis(3).bits = [oldtrellis(2).bits 1 0];
166         else
167             newtrellis(3).errors = error2;
168             newtrellis(3).bits = [oldtrellis(4).bits 0 1];
169         end
170
171         % State 10
172         error1 = oldtrellis(2).errors + sum(bits ~= [0 1]);
173         error2 = oldtrellis(4).errors + sum(bits ~= [1 0]);
174         if error1 < error2
175             newtrellis(4).errors = error1;
176             newtrellis(4).bits = [oldtrellis(2).bits 0 1];
177         else
178             newtrellis(4).errors = error2;
179             newtrellis(4).bits = [oldtrellis(4).bits 1 0];
180         end
181
182         % Make the new trellis the old trellis and continue
183         oldtrellis = newtrellis;
184     end
185
186     % Select the lowest error path

```



```

187     [~,I] = min([oldtrellis.errors]);
188     codedbits = oldtrellis(I).bits;
189
190
191     % Decode the found bits
192     bits = zeros(1,length(codedbits)/R);
193     state = [0 0];
194     for ii=1:length(bits)
195         % Get the current coded bits
196         cbits = codedbits(2*ii-1 : 2*ii);
197
198         % Find the data bit depending on the state
199         if isequal(state, [0 0])
200             bits(ii) = isequal(cbits, [1 1]);
201         elseif isequal(state, [0 1])
202             bits(ii) = isequal(cbits, [0 1]);
203         elseif isequal(state, [1 0])
204             bits(ii) = isequal(cbits, [0 0]);
205         else % isequal(state, [1 1])
206             bits(ii) = isequal(cbits, [1 0]);
207         end
208
209         % Update the state
210         state = [state(2) bits(ii)];
211     end
212
213
214     if plots
215         errors = find(codedbits ~= detectedbits);
216
217         figure(4); clf(4);
218         subplot(3,1,1); hold on;
219         stem(detectedbits);
220         title('Hard detected bits');
221
222         subplot(3,1,2); hold on;
223         stem(codedbits); stem(errors, codedbits(errors), 'r');
224         legend('Correct bits', 'Corrected bits');
225         title('Coded bits after error correction');
226
227         subplot(3,1,3); hold on;
228         stem(bits);
229         title('Decoded bits');
230     end
231     else
232         bits = detectedbits;
233     end
234
235
236     % Return only the requested symbols
237     bits = bits(1:len);
238 end

```

Operational Figures

On the following pages, I have attached sample figures demonstrating my the functioning of my system. I have shown a higher-than-average noise sample demonstrating that even under harder transmit criteria, I receive less than a .3% error rate.

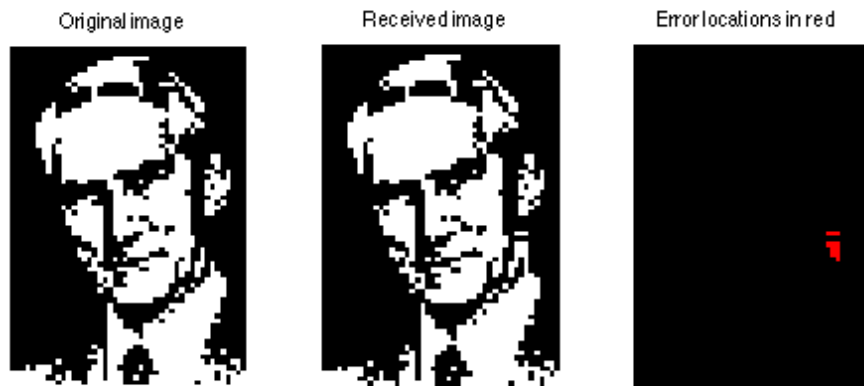


Figure 1: Comparison of transmitted and received images

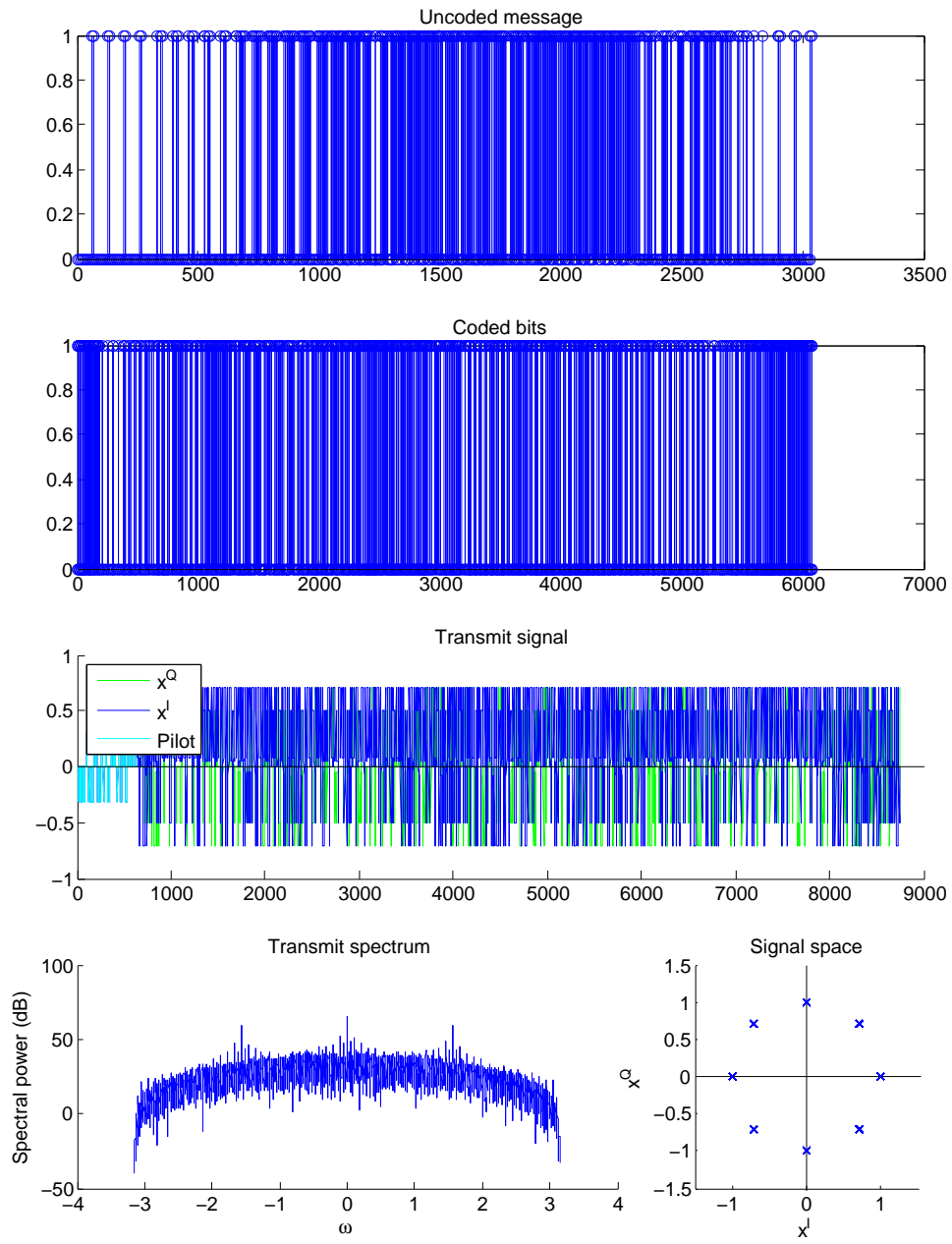


Figure 2: Generated transmit signal

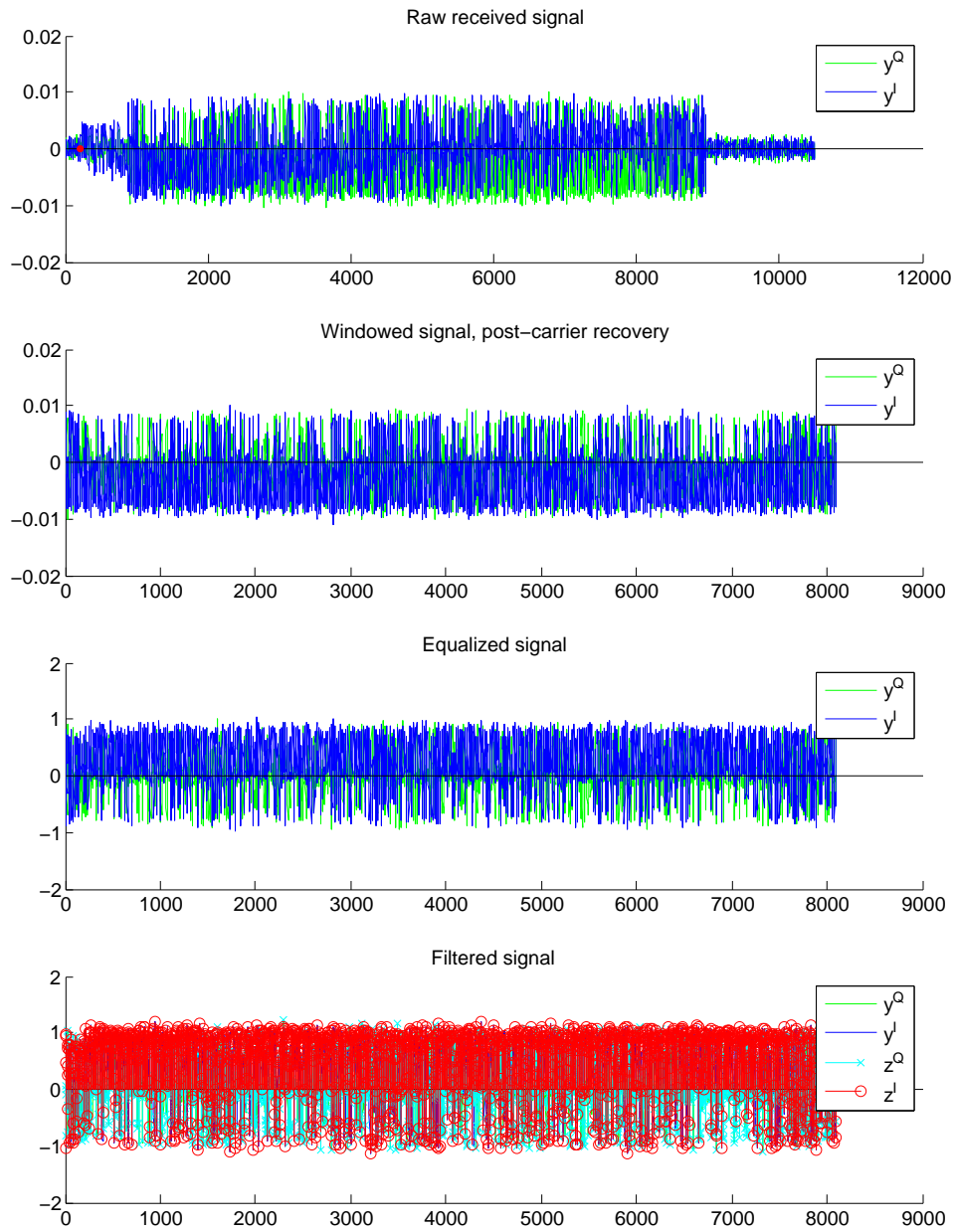


Figure 3: Signal received from radio, and various processing

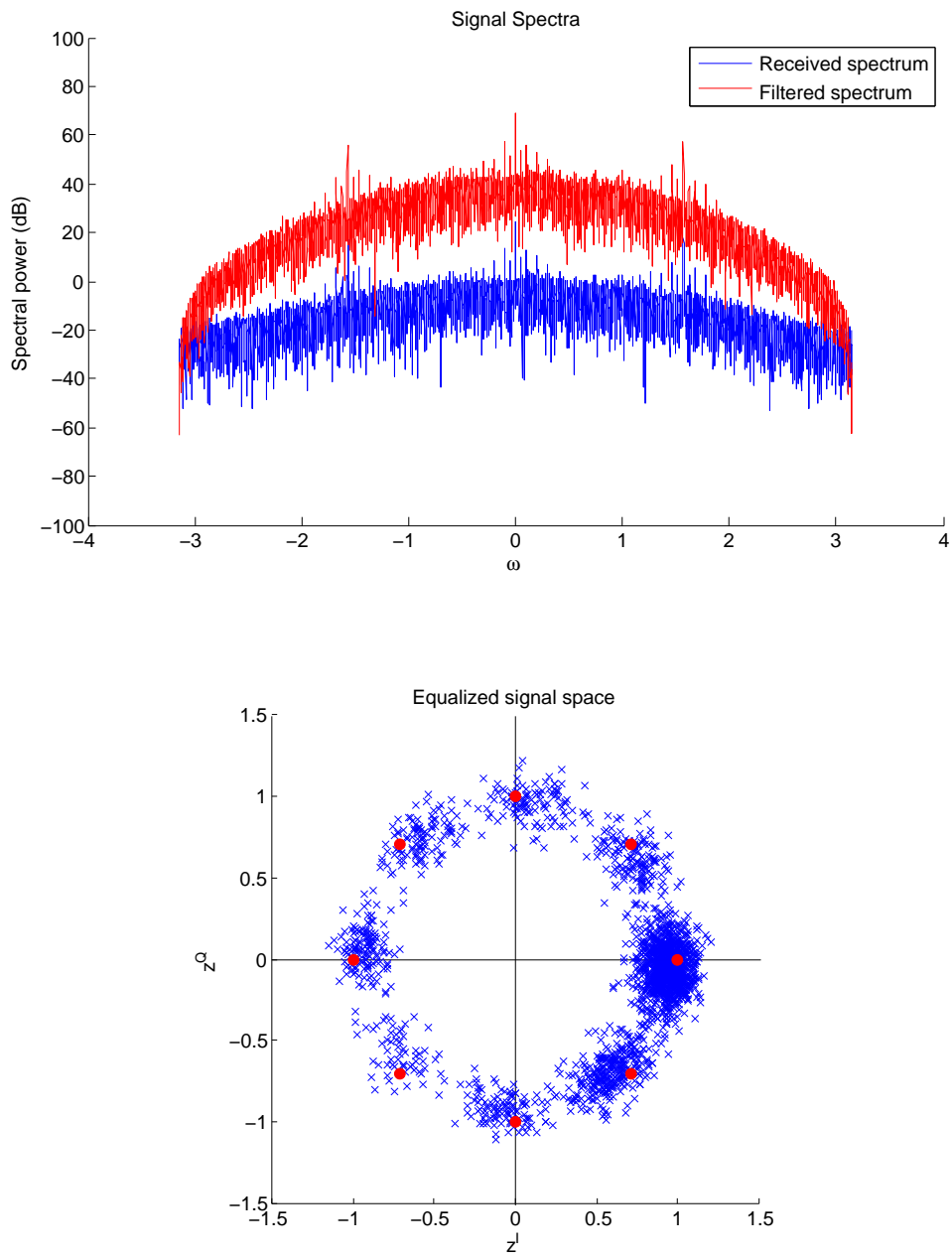


Figure 4: Received signal spectra and signal space

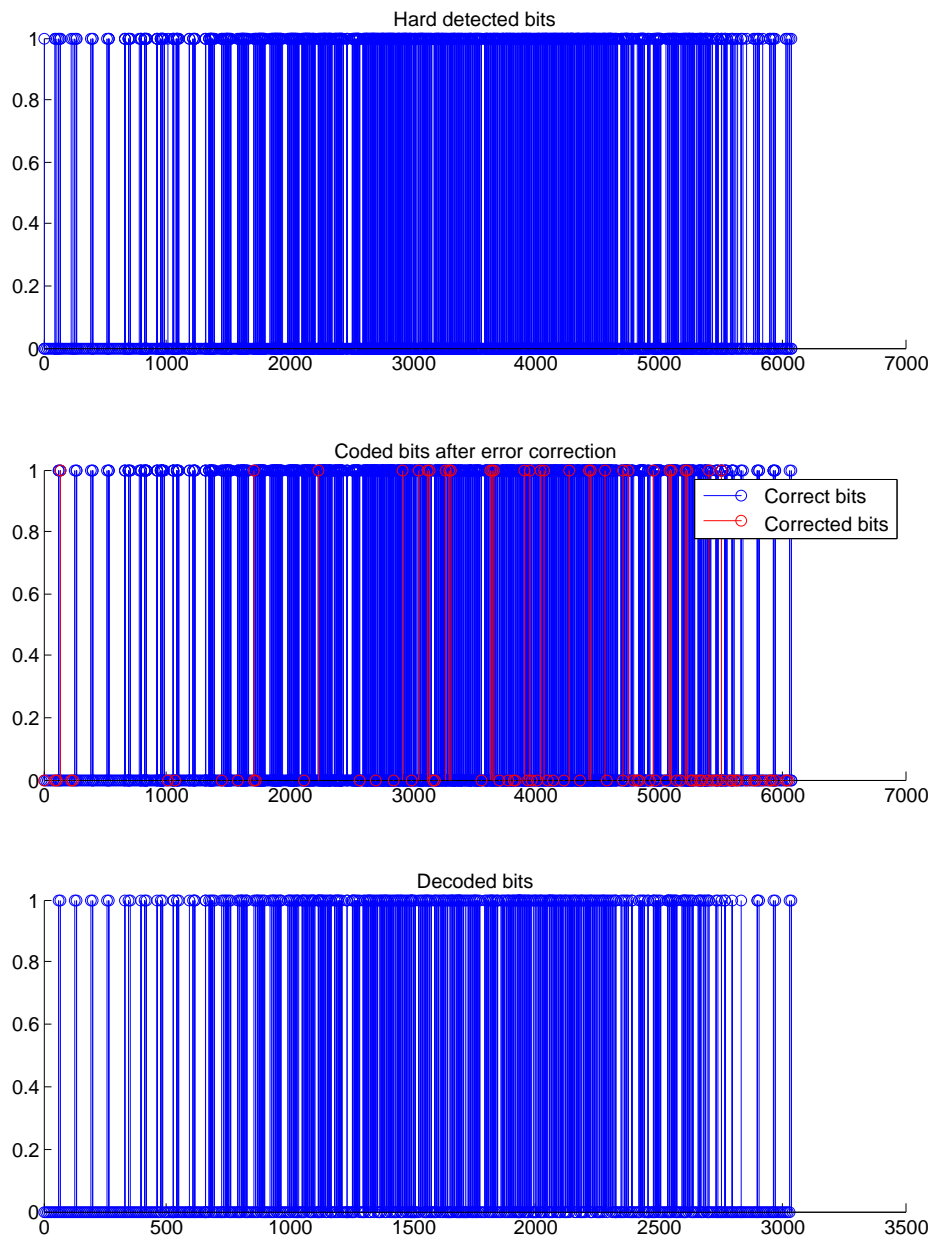


Figure 5: Received signal bits, and decoding