ETH zürich

Bericht

P&S: BITS ON AIR

Präsentiert von: Thorbjörn Höllwarth

Yao Zhou

Beaufsichtigt von: Manuel Strahm

Recep Gül

Prof. Dr. Helmut Bölcskei

Mai 2020

Chair for Mathematical Information Science (MINS)

Inhaltsverzeichnis

Inhaltsverzeichnis 2
Einleitung 3
Theorie & Versuchsverlauf 4
Erster Nachmittag 4
Zweiter Nachmittag 6
Dritter Nachmittag 9
Vierter Nachmittag 10
Fünfter Nachmittag 11
Sechster Nachmittag 13
Resultate & Fazit 14

Einleitung

In diesem Bericht geht es um die digitale Nachrichtenübertragung und deren Aufbereitung über die Modulation, der Übertragung über Antennen und die dabei entstehenden Störungen, und über die Entzifferung via Demodulation. Ziel dabei ist es, neben dem Verstehen der Nachrichtenübertragung, auch essenzielle Grundkenntnisse im Umgang mit dem für Ingenieure wichtigen Programmiertool MATLAB[1] zu erlernen. Deshalb wird im Verlauf dieses Berichts öfters auf die Anwendung verschiedener Code-Snippets eingegangen.

Am Ende versuchen wir unser erworbenes Wissen unter realen Bedingungen zu testen und machen von einer Amateurfunk-Radiostation in Deutschland gebrauch.

Ferner wollen wir noch betonen, dass dieses P&S aufgrund der speziellen Situation, mit der wir alle im Frühjahr 2020 konfrontiert waren, via Zoom[2] im HomeOffice stattgefunden hat. Ob und wie gut diese alternative Arbeitsweise funktioniert hat, ist dem letzten Teil zu entnehmen.

[1] www.mathworks.com – zul. Abgerufen am 27.05.2020
[2] www.zoom.us – zul. Abgerufen am 27.05.2020

Theorie & Versuchsverlauf

Die Theorie wurde uns als Skript[3] zur Verfügung gestellt, sowie jeweils an jedem Nachmittag über Zoom erklärt und vorgeführt. In Gruppen unterteilt wurden wir in verschiedene Breakout-Rooms verteilt, wo es uns möglich war, den jeweils anderen Bildschirm mit dem Code zusehen und damit zu interagieren. Kamen Fragen auf, konnte ein Assistent durch «Hand heben» für die Klärung dazu geholt werden – auch dieser konnte ggf. auf den PC des jeweils anderen zugreifen und bei der Lösung helfen.

Im Vorbereitungsskript, welches bis zur ersten Lektion zu bearbeiten war, ging es darum, sich schon mal ein bisschen mit MATLAB vertraut zu machen und wichtige Grundoperationen wie

- Matrizen erstellen und mit definierten Werten füllen, wie Bspw. die Einheitsmatrix,
- Vektoren mittels Startwert, Schritt und Endwert zu generieren,
- Transponieren,
- Submatrizen zu erstellen,
- Matrixmultiplizieren und
- Ausgeben von Matrizen

zu erlernen.

Erster Nachmittag

Nach einer kurzen Einführung in MATLAB, versuchten wir am ersten Nachmittag einen Sender und einen Empfänger für unser digitales Kommunikationssystem zu programmieren.

Der Sender soll eine zufällige Bitfolge aus *N* Bits erzeugen, die wir mit folgendem Code realisiert haben. Wie oft eine 0 oder 1 erscheint, wird durch den Wert von p bestimmt, welcher zwischen 0 und 1 liegt.

Source:

```
function bitsequence = source(sequence_length,p)
    bitsequence = ( rand(1,sequence_length)-p )>0;
end
```

Um den Mittelwert und die Varianz zu erhalten, programmierten wir die Funktion **drain.m**, welche diese beiden Werte als Vektor zurückgibt:

Drain:

```
function q = drain(bitsequence)
    average = mean(bitsequence);
    variance = (std(bitsequence)).^2;
    %q = average;
    q = [average, variance];
end
```

Danach schrieben wir ein Programm **loop.m,** welches es ermöglicht, die Programme **source.m** und **drain.m** in einer Schleife *loopsize*-Mal laufen zu lassen. Es soll am Ende des Ablaufs, mittels

[3] https://www.mins.ee.ethz.ch/teaching/PPS/BoA/ – zul. Abgerufen am 27.05.2020

Histogramms, die ermittelten Mittelwerte grafisch ausgeben. Wir sehen, dass es einer Gausskurve ähnelt (Abbildung 1).

Loop

```
function loop(loopsize, N, p)
    averages = drain(source(N,p));
    storage = zeros(1,2);

     for i = 1:loopsize
         storage = drain(source(N,p));
         averages (i) = storage(1);
     end

    histogram(averages,1000);
    %gauss(mean(averages),std(averages));
end
```

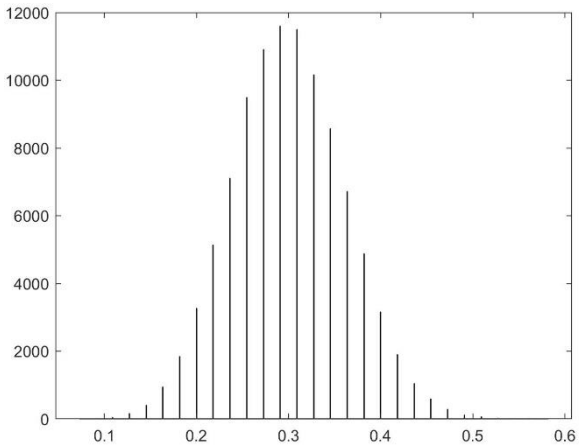

Abbildung 1: Histogramm von 100000 Bitsequenzen [MATLAB]

Als nächstes galt es, einen Kanal zu implementieren, um den gewünschten Programmfluss zu erreichen, wie er in Abbildung 2 dargestellt ist.

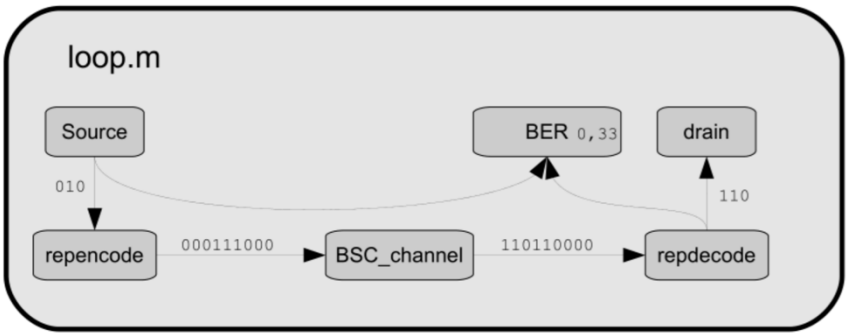

Abbildung 2: Graphische Darstellung des gesamten Programmflusses [Skript]

Dies erreichten wir mit folgendem Code:

Repencode:

```
function repetedsource = repencode(source,N)
  repetedsource = repelem(source,N);
          %repetedsource = repmat(source,N,1); %if matrix is needed
```

Repdecode:

```
function decoded = repdecode(bit_seq,N)
  decoded1 = reshape(bit_seq,[N,length(bit_seq)/N]);
```

BSC_channel:

```
function channel_bit_sequence = BSC_channel(bit_seq,q)
  bit2 = source(length(bit_seq),1-q);
  channel_bit_sequence = xor(bit_seq,bit2);
```

CalcBER:

```
function BER = calcBER(source, sent)
  N = length(source);
  BER = (1/N)*(sum(mod((sent - source),2)));
```

Zweiter Nachmittag

Am zweiten Nachmittag arbeiteten wir hauptsächlich mit der Modulation und der Demodulation.

Bei der Modulation wird eine Bitfolge in ein kontinuierliches Signal umgewandelt, wobei 0 und 1 jeweils eine unterschiedliche Trägerfrequenz (f_0 oder f_1) haben. Wir müssen auch eine Symboldauer τ_S wählen, welche angibt, wie viele Zeiteinheiten für die Übertragung eines Symbols (1 Bit) verwendet wird. Diese Art der Modulierung nennt man Frequency Key Shifting (FSK).

Um FSK zu implementieren, schrieben wir den folgenden Code:

Modulate:

```
function y = modulate (b)
    figure('Name','Measured Data');
    %setVariables;
    load ('data.mat') %for tau's

    %PLOT1: BitSequenz (plots values of b in discrete form )
    subplot(5,1,1) %creates subplot
    seq=ranSeq %implementet randomSignal
    x = [1:1:length(seq)]
    plot (x,seq,'o'); %OK

    %PLOT2: diskrete Bitsequenz (Bit wird fortgesetzt)
    subplot(5,1,2);
    xx = [1:1:tauS*length(seq)];
    largeB = repelem(seq,tauS);
    plot(xx,largeB,'o');
```

```
    %PLOT3: Trägermodulation x1
    subplot(5,1,3);
    trager0 = 1* cos((2*pi*xx)/tau0);
    plottrager0 = trager0 .*(1-largeB);
    plot(xx,plottrager0);

    %PLOT4: Trägermodulation x2
    subplot(5,1,4);
    trager1 = cos((2*pi*xx)/tau1);
    plottrager1 = trager1.*largeB;
    plot(xx,plottrager1);

    %PLOT5: moduliertes Signal
    subplot(5,1,5);
    y = plottrager0 + plottrager1;
    plot(xx,y);

    save ('data.mat','y','-append'); %Moduliertes Signal wird für
  spätere Demodulation gespeichert (ohne noise)

    fprintf("modulate finished")
end
```

Bei der Demodulation wird durch Korrelation des empfangenen Signals mit den Trägerfrequenzen die Information vom hochfrequenten Signal extrahiert. Die Korrelation der empfangenen Werte mit den beiden Trägerfrequenzen f_0 und f_1 ist definiert als:

$$z_i[n] = \int r(t) \cdot x_i(t) \cdot p(t - n\tau_S)dt,$$

wobei

$$p(t) = \begin{cases} 1, & t \in [0, \tau_S) \\ 0, & t \notin [0, \tau_S). \end{cases}$$

Falls wir

$$\int \cos(a \cdot t) \cdot \cos(b \cdot t)dt$$

berechnen, sehen wir, dass dieses Integral gleich 0 ist, wenn a ungleich b ist. Das Resultat ist nur ungleich 0, wenn a = b.

In MATLAB eingebettet ergab das den folgenden Code:

Demodulate:

```
function bhat = demodulate(r)
    figure('Name','Measured Data2');
    load ('data.mat');
    r=y;
    hold on;

    %PLOT1: The Signal
    subplot(7,1,1);
    plot(r);
```

```
    %PLOT2: Multiplikation mit x_1
    subplot(7,1,2);
    xx = [1:1:length(r)];
    trager0 = 1* cos((2*pi*xx)/tau0);
    multi0 = r.*trager0;
    plot(xx,multi0);

    %PLOT3: Korrelation mit x1
    subplot(7,1,3);
    z01 = r.*trager0;
    z00 = sum(reshape(z01,tauS,length(z01)/tauS));
    x = [1:1:length(z00)];
    plot(x,z00,'o');

    %PLOT4: Multipliklation mit x_2
    subplot(7,1,4);
    trager1 = 1*cos((2*pi*xx)/tau1);
    multi1 = r.*trager1;
    plot(xx,multi1);

    %PLOT5: Korrelation mit x2
    subplot(7,1,5);
    z10 = r.*trager1;
    z11 = sum(reshape(z10,tauS,length(z10)/tauS));
    plot(x,z11,'o');

    %PLOT6: Multipliklation mit x_2
    subplot(7,1,6);
    diffZ = z11 - z00;
    plot(x, diffZ, 'o');

    %PLOT7: Detektierte Bitsequenz
    subplot(7,1,7);
    detekt_seq = (diffZ > 0 );
    bhat = (diffZ > 0 );
    plot(x,detekt_seq,'o');

end
```

Nachdem wir diese beiden Programme ausgeführt hatten, erhielten wir 12 Plots, die jeden Schritt der Modulation/Demodulation visualisieren (Abbildung 3).

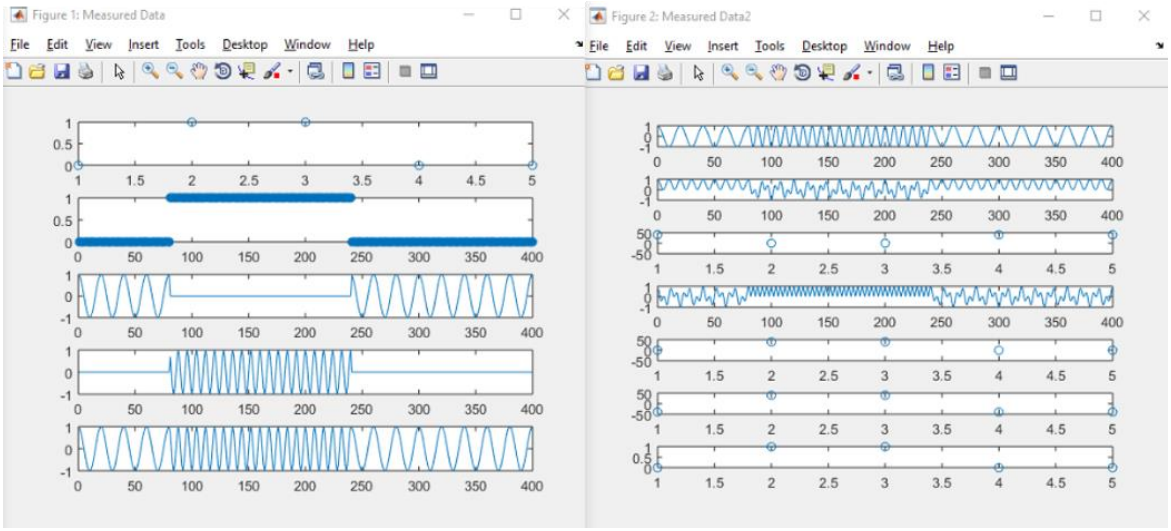

Abbildung 3: Entwicklung der Modulation (links) und der Demodulation (rechts) [MATLAB]

Dritter Nachmittag

Am dritten Nachmittag nutzten wir die von MATLAB bereitgestellte Funktion zur Kreuzkorrelation (**xcorr**) und wandten sie auf die Synchronisation an. Wir haben uns dabei mit zwei verschiedenen Typen von Synchronisation auseinandergesetzt: die Symbolsynchronisation und die Rahmensynchronisation.

Bei der Symbolsynchronisation beschäftigen wir uns mit dem Problem, dass das Signal Rauschen enthält, welches zu falsch verschobenen Symbolen führt. Mit der Funktion **sync.m** versuchten wir, den Offset zwischen dem ursprünglichen Signal und dem empfangenen Signal mittels Kreuzkorrelation zu finden.

Sync:

```
function offset = sync(a,b);
  %PRE: a & b nur mit Werten (1,-1)
  x = xcorr(a,b);                          % x vectorLength is
  =2*length(a) -1
  [m, index ] = max(x);
  offset = index - length(a) + 1;
end
```

Bei der Rahmensynchronisation berücksichtigen wir das Rauschen, welches als Störung bei diversen Übertragungsarten, sei es über Antennen oder auch bei der Aufnahme über das Mikrofon, immer dazukommt. Die Rahmensynchronisation ist ein Mechanismus, der den Beginn des tatsächlich übertragenen Signals trotz des Vorhandenseins von Rauschen identifiziert.

Vierter Nachmittag

Die Aufgaben am vierten Nachmittag bestanden darin, eine Bitfolge zu modulieren, diese über die eigenen PC-Lautsprecher abzuspielen, über das Mikrofon wieder aufzunehmen und als Datei in MATLAB anzuzeigen (Abbildung 4). Dies realisierten wir mit folgendem Code:

Record sound:

```
function recDataWithNoise = recordsound(SignalToSend)
    load ('data.mat')    %Fs, nBits, nChannels,  recTime
%Audiorecorder:
    recObjWithNoise = audiorecorder(Fs,nBits,nChannels);       %recObj
<-- recorded data in HERE

%Signal Modulieren:
    %generate Testsignel unmodulated
    %original Testsound unmodulated
        signalToModulate=SignalToSend
    %Modulate Sound
        moddedSoundData=modulate(signalToModulate);

%play and record Sound

    fprintf("Start recording\n")
        record(recObjWithNoise);
        playsound(moddedSoundData);     %play the modded sound
        stop(recObjWithNoise);
    fprintf("Stop recording...\n")

%Plot RecObj:
    figure('Name','Received Signal');
    y = getaudiodata(recObjWithNoise);
    plot(y);

%Return:
    recDataWithNoise=recObjWithNoise;
    fprintf("recordsound finished \n")
end
```

Play sound:

```
function playedData= playsound(moddedSoundData)        %soundData =
modulated Signal
    fprintf("playsound Start \n")

    load ('data.mat')

    %audioplayer
    player = audioplayer(moddedSoundData,Fs,nBits);        %creates an
audioplayer object for signal Y, using sample rate Fs. The function
returns a handle to the audioplayer object, player.
    play(player);
    pause(2);
    stop(player);

    fprintf("playsound finished \n")
end
```

Die für das Testsignal verwendete Bitfolge war:

```
signalToSend=[1,0,1,1,0,0,0,1,0,1,1,0,0,0,1,0,1,1,0,0,0,1,0,1,1,0,0,0
,1,0,1,1,0,1,0,1,1,1,1,0,0,0,1,0,1,1,0,0,0,1,0,1,1,0,0,0,1,0];    %for
testing
```

Und ergab folgende aufgezeichnete Audiodatei:

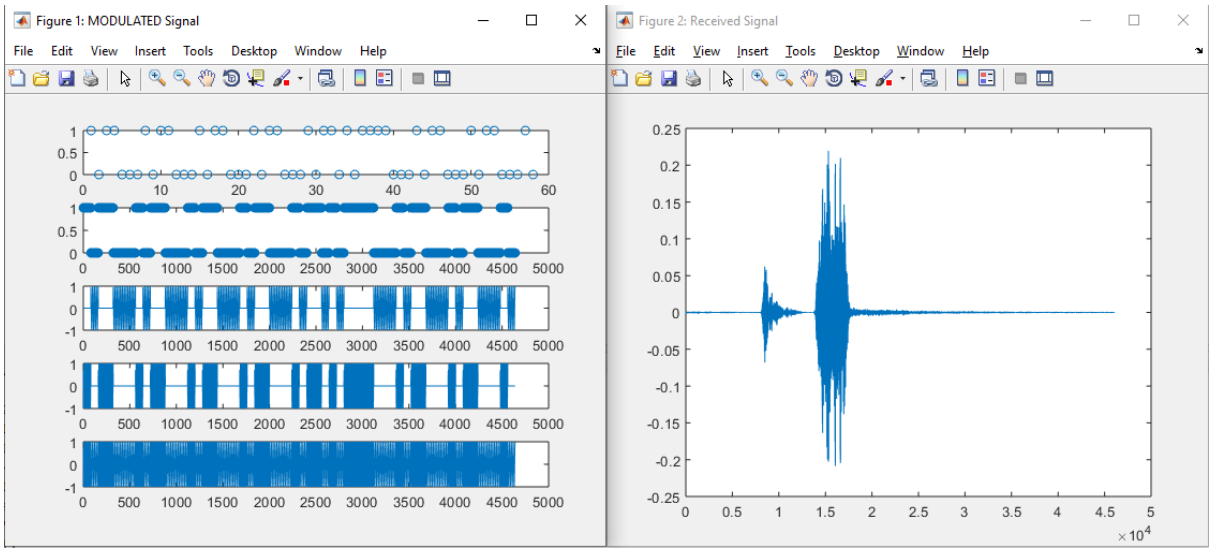

Abbildung 4: Bild der vom Mikrofon empfangenen Audiodatei [MATLAB]

Fünfter Nachmittag

Am fünften Nachmittag stand die komplexe Korrelation im Zentrum.

Diese ist definiert als:

$$\varphi_i = \lim_{T\to\infty} \frac{1}{T} \int_{-\infty}^{\infty} x(t) e^{2\pi j f_i t}\, dt \qquad i = 0,1$$

Für komplexe Funktionen können wir die **xcorr** Funktion von MATLAB aufgrund fehlender Implementation für komplexe Werte nicht benutzen. Jedoch aber können wir mit der Euler-Identität und den trigonometrischen Identitäten das obige Integral in einen realen Teil und einen imaginären Teil trennen. Auf diese Weise ist es uns möglich, die Korrelationen für die beiden Teile getrennt zu berechnen.

$$z_i = \lim_{T\to\infty} \left(\cos(\theta) \underbrace{\frac{1}{T} \int_{-\infty}^{\infty} \cos(2\pi f_k t) \cos(2\pi f_i t)\, dt}_{=1,\ \text{falls } i=k} + \sin(\theta) \underbrace{\frac{1}{T} \int_{-\infty}^{\infty} j \sin(2\pi f_k t) \sin(2\pi f_i t)\, dt}_{=1,\ \text{falls } i=k} \right)$$

Decide Correlation:

```
function symbolDecision =
decideCorrelationStudents(samplesf0,samplesf1,fsample,fsymbol,f0,f1,s
amplesPerSymDur)
```

```
        tSymbol=1/(fsymbol);
        tStart=0;
        tEnd=tSymbol;
        tSchrittweite=tSymbol/samplesPerSymDur;
        t = [tStart:tSchrittweite:tEnd];

        %Correlate z0:
        corr0cos=xcorr(samplesf0,cos(2*pi*f0*t));
        corr0sin=xcorr(samplesf0,sin(2*pi*f0*t));
        z0= abs(corr0cos+i*corr0sin);

        %Correlate z1:
        corr1cos=xcorr(samplesf1,cos(2*pi*f1*t));
        corr1sin=xcorr(samplesf1,sin(2*pi*f1*t));
        z1= abs(corr1cos+i*corr1sin);

        %Create H-vector:
        H=z1-z0;

        %H-Vektor normieren:
        hMax=max((z1),(z0));
        H_Norm=H./hMax

        %J-Vektor halbieren:
        length(samplesf0)
        length(H_Norm)
        H_Norm_Half=H_Norm(length(H_Norm)/2:end)
        length(H_Norm_Half)
        symbolDecision = H_Norm_Half;
    end
```

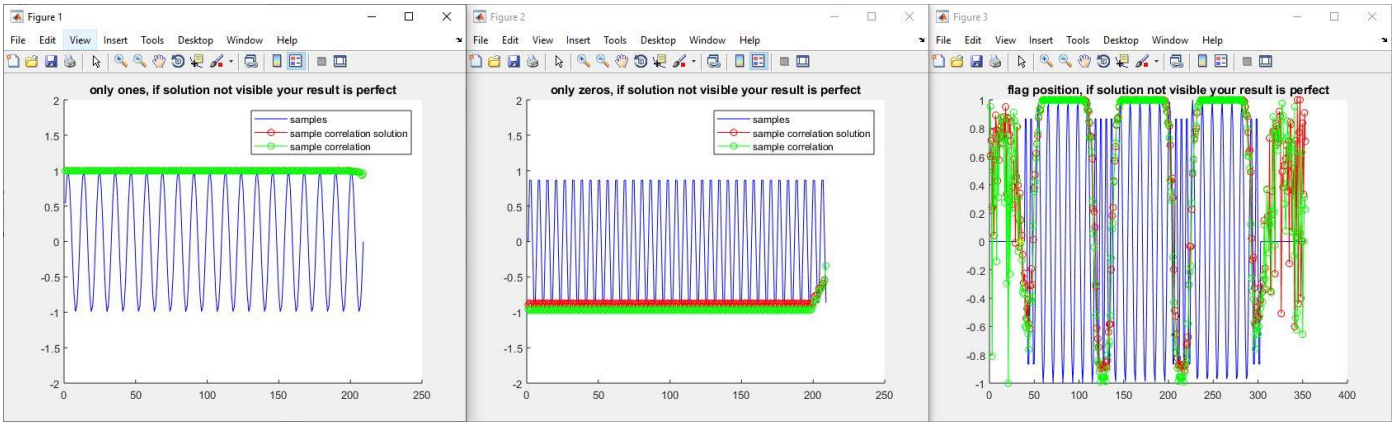

Abbildung 5: Ergebnisse von Tests und Flag-Position [MATLAB]

Sechster Nachmittag

Am sechsten und somit letzten Nachmittag versuchten wir, APRS Nachrichten in Echtzeit zu empfangen. Dazu machten wir die uns empfohlene Amateurfunk-Radiostation in Friedrichshafen[4] zu Nutze.

Abbildung 6: Amateurfunk-Radiostation in Friedrichshafen [Website]

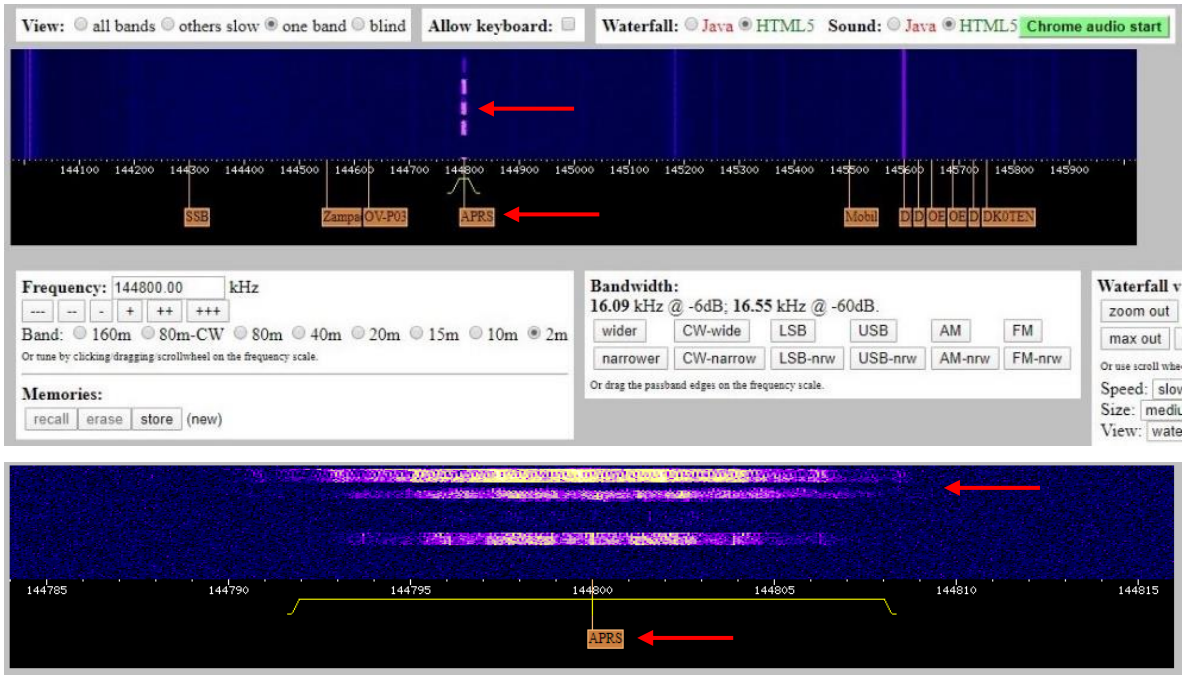

Abbildung 7: Die Webseite vom Sender in Friedrichshafen, eingestellt auf 2m Band und 144.800 MHz [Webseite]

Das empfangene Signal ist violett beim APRS-Signal in der Abbildung 7 erkennbar.

[4] http://dk0te.dhbw-ravensburg.de:8901 – zul. Abgerufen am 27.05.2020

Resultate & Fazit

In diesem P&S haben wir vieles über die digitale Nachrichtenübertragung und deren Zustandekommen erfahren. Die dafür benutzte Programmierumgebung MATLAB eignete sich dazu sehr gut. Selbst programmiert, erhielten wir Einblick in jeden Schritt, der von der Signalentstehung über die Modulation und der Übertragung, bis hin zur Symbol- und Rahmensynchronisation und der Demodulation notwendig ist, um das Ursprungssignal wieder zu erhalten.

Durch das Kennenlernen und permanente Ausprobieren und Umschreiben von Funktionen in MATLAB, haben wir durch das gesamte P&S hindurch essenzielle Programmiererfahrungen sammeln können, welche uns als Ingenieure lange begleiten und zu Gunsten kommen werden. Wichtige Arten im Umgang von Matrizen und Funktionen wurden dabei erlernt und grundlegende Abweichungen zu anderen Programmiersprachen festgestellt.

Die Alternative Unterrichtsweise über Zoom hat während der gesamten Zeit gut funktioniert und nahezu keine Probleme bereitet. Das interaktive ScreenSharing-Tool, sowie die Aufteilung in sog. Breakout-Rooms, hat ausnahmslos gut funktioniert und ermöglichte uns somit den reibungslosen Ablauf des Praktikums.

Als Erfolg verzeichnet, schliessen wir dieses P&S mit den erworbenen Kenntnissen ab und danken der Betreuung für die Unterstützung bei der Durchführung des P&S Bits on Air.