In the primary submission directory, there are mainly 5 Matlab files: *wifitransmitter.m* for the transmission function, *wifireceiver.m* for the primary receiver function, *vitDecode.m* function to perform Viterbi decoding on a given sequence, *test.m* for testing and *kalman\_filter.m* for a simple Kalman filtering implementation.

My receiver first does some vanity checks and tries to catch the preamble in the received signal. The way to do it is by utilizing a sliding window which is the expected preamble sequence and slides it through on every possible interval of the input sequence to compute the total Euclidian distance between each sequence pair. By selecting the minimal distance index we can say that this is the most possible place where the preamble hides. After that I decode the message length, since it is exactly the following 64 data points after the preamble and there’s no turbo encoding performed on the length bits, it is easier to decode. After getting the message length, I can thus infer the length of the packet because every 16 characters makes up for 128 bits, which is the length of a symbol in the transmission part, and it is made sure that bits are appended so that the information bits are a multiple of 128 bits at the beginning. Knowing the preamble position and the length of packet, I truncate the sequence to eliminate the padding at the beginning and end of the packet and pass it to Layer #2.

In my second layer, I reverse the OFDM by performing inverse FFT on every 64 data points and pass it to Layer #3. In Layer #3 I perform the 4-QAM demodulation using the *qamdemod* function, which provides certain tolerance for noise and drop the preamble sequence.

The demodulated sequence now becomes binary bit sequence due to the 4-QAM demodulation and is sent to the next layer. In Layer #4 the sequence is performed with a specific version of Viterbi decoding in the *vitDecode* function. This function can only decode sequence with the Trellis generated by *poly2trellis(3,[7,5])* in Matlab and is not a general Viterbi decoding function. The procedure generally mimics the shortest path algorithm mentioned in Recitation 1, in which I first compute weight matrix to represent branch metrics, then compute cost matrix to represent the cost of reaching each state, after that trackback to find shortest state path, and finally generate the original bits utilizing the state path.

Finally the sequence is sent to Layer #5 to undo interleaving. I first generate an rev-interleaving matrix, just like the Interleave matrix in the transmission side, and apply this sequence permutation on every 128 bits. At last I can get the character representation by combining the length of the message, the start position of the message and regenerate the whole message character by character.

I want to talk a bit about this receiver’s anti-noise performance. During my test, the receiver can perform well on a SNR down to -15 dB, and after -16 to 18 dB there is a significant performance drop down, so I would say it can handle input SNR down to -15 to -16 dB. In my opinion, the mechanism behind it is that firstly the *qamdemod* function provides an anti-noise demodulation to generate binary bit sequence. As long as the noise is under a certain limit, the generated bits would not flip. Even though some bits do get flipped, the sequence content can also be fixed in the *vitDecode* function when finding the most likely sequence. I also tried to perform Kalman filtering on the received signal, but there does not seem to be any performance improvement as far as I can see, so in the end I gave up the Kalman filtering and sticked to the original design.

There are no special libraries needed besides the communication toolbox of Matlab, and the receiver can be tested by changing and running the code in *test.m.*