EE5505 Final Project Report - Research on Coding and Modulation with Compression Methods

Tianming Cui

May / 2019

Introduction

In this project, I am trying to find some compression algorithms with corresponding coding and modulation methods that can be used to transfer some particular curve data in different wireless channels.

We have a wireless remote controlled vehicle generates a large amount of data and need to send the data to the control center, but since it may stay far away from the control center and may need to move into some complex environments, sometimes the wireless channel can have a really bad condition. We do not need to transmit the data one hundred percent accurately, but we still need to make sure the quality of transferred information is acceptable even with a bad wireless channel. And obviously, the faster the better.

This project is mainly based on experiment, and in this report I will discuss the experiment results and make a conclusion.

Preprocess on the data

The raw data generated by the sensors on the vehicle is a long set of integer numbers(through time), which can be plotted as following:

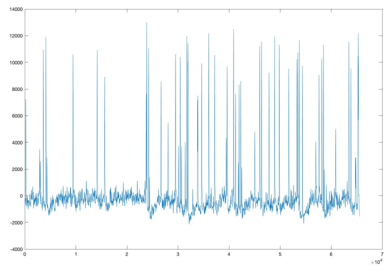
Figure 1: Raw data

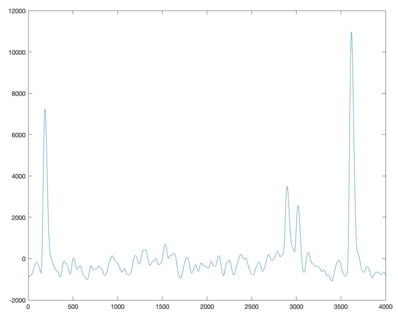
Figure 2: Partial enlarged

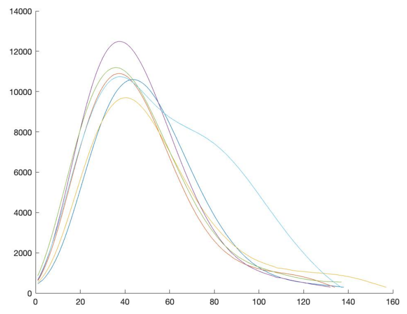
Figure 3: Filtered data

The vehicle generates high curves when it detected something special, while at most of the time, the sensor is just recording environment noises.

So it will be a good idea to extract those high curves which contain a large amount of real information, and only transfer them if we are not able to transfer all the data.

The algorithm of this extraction is: When detect the curve begin to going up, record the index of the current point. Therefore, between each two recorded index, we got a complete concave down curve. Then, examine every curve we got, if it is significantly(more than 5x) larger than nearby curves, we extract it as an important curve. In the real-time system where the data are generating continuously, this extraction algorithm actually tells the transmitter when it needs to begin or finish transmitting. If currently the vehicle is generating noise data points, then not to transfer these noisy points will greatly save the battery life.

After filtering, there are totally 8 curves, with total length. This preprocessing actually greatly reduced the data size. In the following parts of this report, I will mainly discuss the yellow curve(curve#2) from the filtered data.

Basic Coding and Modulation

To transfer some data with a wireless channel, we need to first design some coding and modulation methods.

Take a look into one of the extracted curves(the yellow one) as an example,

#Point	1	2	3	...	37	38	...	132	133	134
-	614	800	1018	...	10894	10892	...	372	330	286

We can find that the curve consists a set(totally 134) of integer points, with a range from 286 to 10894. If we want to be able to represent every possible integer value within 286 to 10894, we need 10608 different codes in our coding system, which equals to 14 binary bits(16384). If we want to use MQAM modulation here, one point can be directly modulated to one 16384-QAM point, but MQAM modulation with such a high M cannot be robust with a bad wireless channel. Since we already know the working environment of our transmitter is usually not so good, we'd better choose a stable and robust modulation method like 16-QAM.

The detailed progress is shown below:

First, convert each point into binary bits. Since the values of the data are within the range [286,10894], the converted binary code could be at least 9 bits and at most 14 bits. Before the converting, the transmitter could not know the actual length of converted code, so we need to padding the binary codes by adding 0s at the end of the code to make sure all of them are 14-bits long, and could be correctly decoded by the receiver.

Then, consider about our modulation method. If we want to use 16-QAM here to fit a noisy wireless channel, then each 16-QAM point can carry 4 binary bits.

To transmit 14-bits values, we have some different methods here: combine 8 values together and transfer them with 7 16-QAM points, or simply transfer one value with one 16-QAM point. These methods have different advantages and disadvantages. Make a combination of several data points then coding and transmit them together can speed up the transmitting progress, since it reduces the total size of modulated points by 12.5%. And this kind of combination will not influence the accuracy, because each binary bits before and after modulation are independent, the error rate of each bit(and therefore each data point) has not changed here. But transfer each data point individually also have a huge advantage: in a real-time system, the data is generated and transmitted at the same time. Although we use an already completely generated dataset in our experiment, in the actual application, the remote control vehicle is a real-time system. If we send several data points together, it means we need to wait until all these data points are generated and then send them, which makes a significant delay. So here I choose to represent each data point with one 16-QAM point, to make sure the transmit delay is acceptable.

Here we may take the value 10086 as an example, in our design, it was first converted to a 14-bit binary code, 1001 1101 1001 10, then we padding extra 0s at the end, make it be able to divide into 4 bit parts, then it becomes 1001 1101 1001 1000. Then each of these 4-bit part will be modulated with 16-QAM, and transmit to the receiver side. Then at the receiver side, every time it received 4 16-QAM points, the receiver does demodulate on these points, get four 4-bit binary codes. By combining these 4-bit codes together, the receiver can rebuild the 16-bit binary code and decode it into a decimal value.

The following figure shows the performance of this transmitting design.

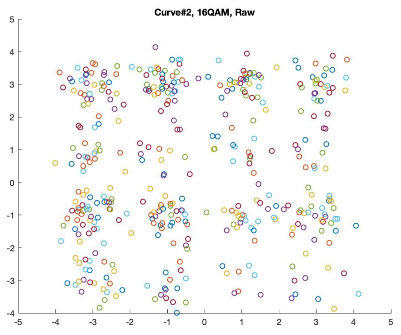

Figure 4: Modulated points

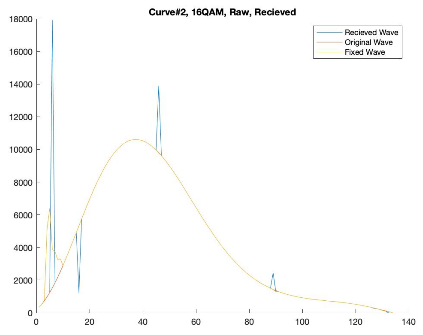

Figure 5: Transferred Figure

The figure at the left side shows the distribute of received modulated points. The points are spread over the space, since the signal has passed an AWGN channel with SNR=15. Some areas have more points than other areas, since we did padding during coding, so some bits are more likely to be or fixed at 0.

The figure at the right side shows the received curve. There are three different curves in the plot: the red line shows the original curve(generated by the vehicle); the blue line shows the received curve, and the yellow line shows the fixed curve.

We can see that due to the noisy wireless channel, some data was transmitted incorrectly, made some bad points in the received curve. So here some algorithms can be used to fix these bad points are needed. A simple approach is: use a low-pass filter on the received signal. Which can greatly reduce the influence of these high frequency noise, but after testing, I used some approach with better performance:

My design here is, first, check all points on the received curve, if one point is significantly(0.7x smaller or 1.5x larger) different with the previous point, then it will be tagged as a bad point which needs to be fixed. The fixing method is based on the continuity of the original curve: since the curve is continuous, one point should be closed to the average value of its nearby neighbors. We abandon those bad points, and using the average value of their nearby points to replace them. After doing this, we got the fixed curve, which looks smooth and more similar to the original data. This fixing algorithm will be used in all following experiments, since it can work with different situations.

Common Compression Approaches

Sampling

Algorithm

The term Sampling in the signal processing field means generating a discrete-time signal from a continuous-time signal, which is widely-used in audio processing.

However, in this project, the original data is already a discrete-time wave, therefore, to be more precisely, we are actually doing re-sampling here, which can also be referred to sample-rate conversion. The reason for doing so is, by reducing the sample rate, we can generate a smaller subset of the original data, which still contains most of the information from the original data. There are many different (re)sampling methods created by former researchers, all of them are aim to represent the original data better for different situations.

In our particular case, the data need to be sent are smooth curves, and it is hard to say which parts of the curve are more important(since we have already done some extraction). So it will be a good idea to simply do the re-sampling evenly across the curve, by choosing one point within each continuously K points. We do this re-sampling at the transmitter side to reduce the datasize by eliminating some points, and then do a reverse re-sampling at the receiver side to regenerate these lost points from the points transferred.

If we want to reduce the data size by a half, then we need to choose the K value as two, means in the newly generated subset, we represent two points with one new point. There are totally 4 different methods here: randomly choose one point in each two points; choose the first point in each two points choose the second point in each two points; take the average of each two points.

If we use randomly choose points, there would be a problem, we lost more information here: the information which tells us which points are chosen. So it is a bad idea to do so since we want to lose less information as possible.

Taking the average of each two points can be a good approach, but if we want to regenerate lost points at the receiver side by calculating the average number of nearby points, then doing the same thing at the receiver side will be an unnecessary move.

Looking at the receiver side, after the whole transmitting progress is done, we got a received curve with half size of the original curve. Then we need to regenerate lost points from what we received. Since the curve is smooth, a possible solution is, between each two points, we regenerate one new point by taking the average of these two points.

Test Result

The processed curve is shown with the purple line, which is quite close to the original data. Some parts of the regenerated curve seem not as smooth as the original one due to the linear regenerated method(taking average).

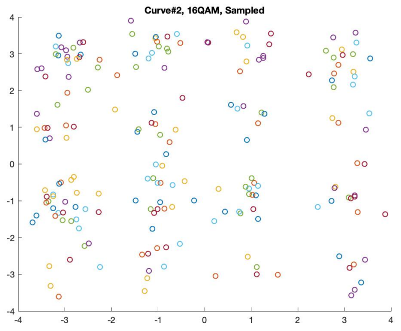

Figure 6: Modulated points

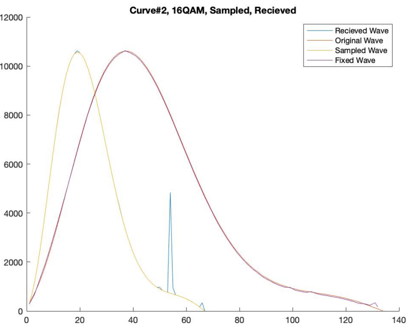

Figure 7: Transferred Figure

The processed curve looks similar to the original curve, they are totally overlapped to each other. But when the channel is even worse, as shown below, the performance will become rather bad(comparing to not doing re-sample).

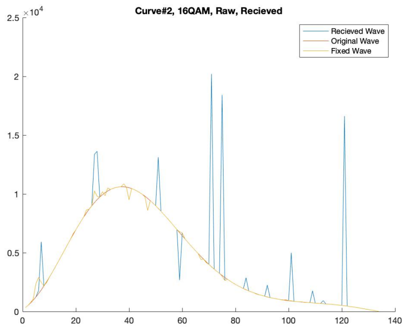

Figure 8: Without Sampling

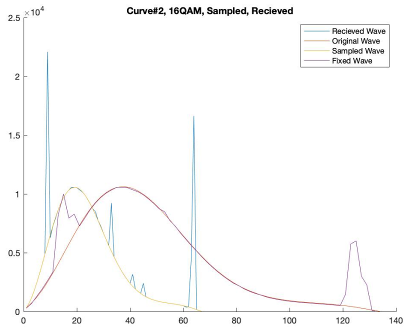

Figure 9: With Sampling

The reason might be: since we lost half of the information by doing re-sampling, the receiver actually cannot have enough information to detect and fix the bad points. Some bad points are not fixable and will influence the whole area when the receiver tried to fix other points nearby.

So here we can make a conclusion: doing re-sampling can significantly speed up the transmit, but it will not be a good method if we have really bad channels.

Reduce Resolution

Algorithm

If we do re-sampling on some data, we lost part of the information along the time axis. There is another compression method that do a similar thing, but abandons some information along the value(y) axis instead of time(x) axis.

In the raw data, we have a minimum unit of 1, means each integer within the range [1, 16384] can be represented by a corresponding independent code.

If we reduce the resolution to the unit of N, means coding all values with [Nk,Nk+1] as a same code k, we can reduce the total number of possible codes by N times.

Test Result and Conclusion

Here we reduce the resolution by 32x. This can be done by doing ceil(original/32) at the transmitter side and add a 32x multiplier at the receiver side.

By doing so, we reduced 16384 different possible code to 332. Which can be represented with 3 16-QAM points to get a 25% speedup, or 4 8-QAM points which can provide a higher performance with a noisy channel.

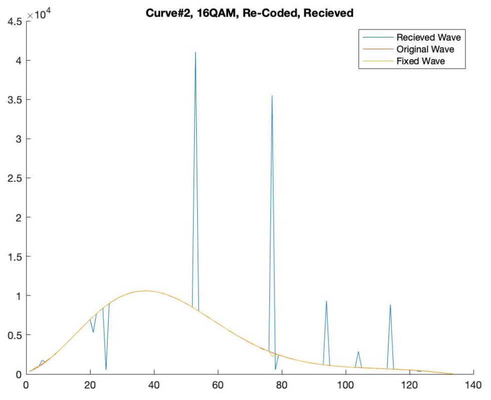

Figure 10: Modulated points

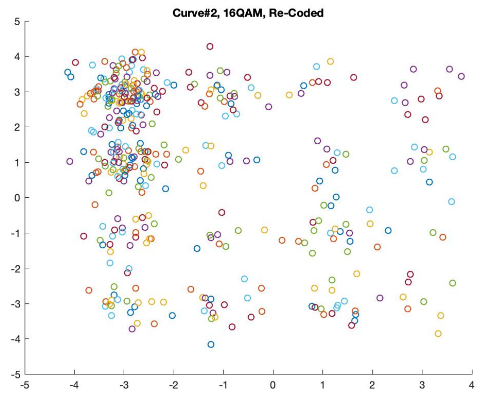

Figure 11: Transferred Figure

From the above figures, we can see that the fixed curve is still similar to the original curve while we obtain a 25% speedup here, if we reduce 4 16-QAM points per data to 3 16-QAM points.

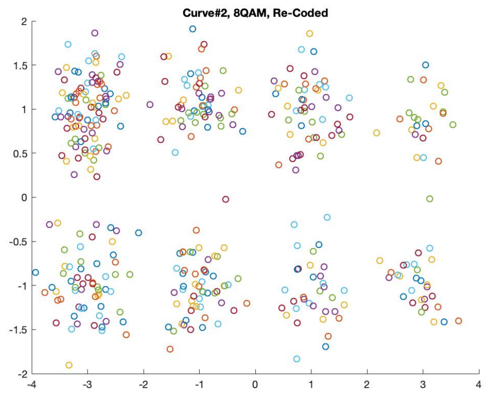

Figure 12: Modulated points

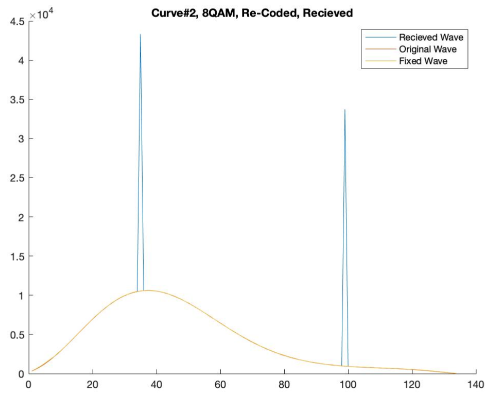

Figure 13: Transferred Figure

And here we can see if we care more about robustness instead of speed, using 8-QAM modulation with resolution reduction can obviously reduce the bad points while not increasing the total size of data transmitted.

So, we can see that by reducing the resolution of our coding algorithm, we can get different benefits if we want, by giving up some accuracy on the y-axis.

Curve-Fitting

Algorithm

It is obvious that the data we need to transfer here is some particular smooth curve with special patterns. We used this characteristic of the data to do the bad-point fixing and lost-point regeneration in the previous approaches. These algorithms improved the accuracy and robustness of the transmitted curve. However, these methods are still dependent on the transmitted data, since they only do modifications on the transmitted data.

But as we know there is some particular pattern in the curve, it will be possible to only transmit this pattern, and redraw the whole curve based on the pattern at the receiver side.

To find the pattern, we can try curve-fitting approaches.

Curve-fitting means find an equation that can well represent a curve. By fitting the curve with a pre-defined equation with several parameters, all data points on the curve can be represented by the equation. Once we find a good equation, then this equation(or its parameters) will be the only thing we need to transmit, which can greatly reduce the amount of data.

The detailed progress is shown as following: First, we need to define an equation that can well represent the curve we need to transfer. During the early stage of this project, I used $f(x) = ax + bx^2 + cx^3 + dx^4$ here, but after reading the manual of the sensor, I use following equation instead(which only make a little bit difference at the result): $y(x) = a(\frac{x+c}{b})^4 e^{-\frac{x+c}{b}}$

After the equation has been decided, we set some initial value for a,b and c. Based on these parameters, compute the y(x), compare it with the actual value on the curve(err = y(x) - actual(x)). And adjust a,b and c with this err. Repeat this progress, until the err is small enough. This can be done with different algorithms, like Gauss-Newton method, coordinate descent, etc. Here we choose the basic algorithm, gradient descent method to do the fitting.

The fitting result is quite close to the original curve. Since we only need to transfer 3 parameters here, we can use the most stable modulation method 2-QAM here, which can ensure the correctness of the transmit. Therefore, by using curve-fitting method here, we can not only obtain a 30x speedup, but also have better robustness with bad wireless channels.

When the channel is in really bad situation with SNR=8, we can compare the performance of curve-fitting approach and directly transmitting approach.

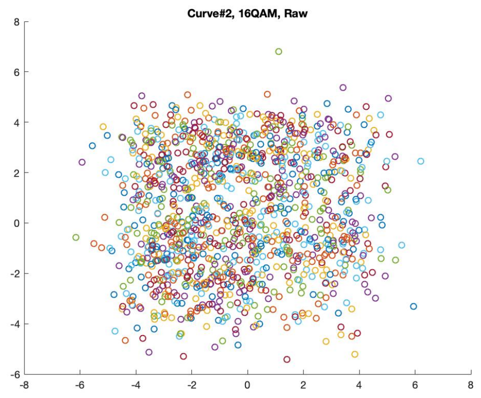

Figure 14: Modulated points, 16QAM

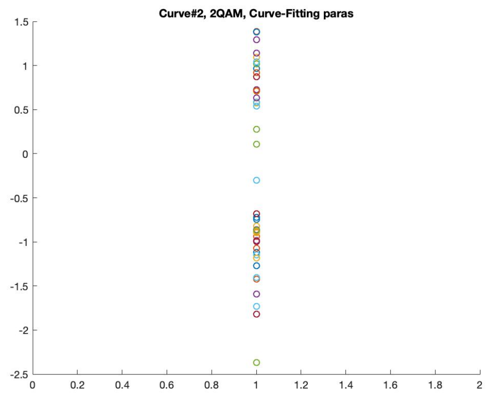

Figure 15: Modulated points, 2QAM

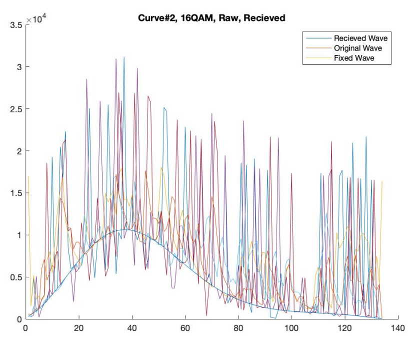

Figure 16: Received curve, directly transmitting

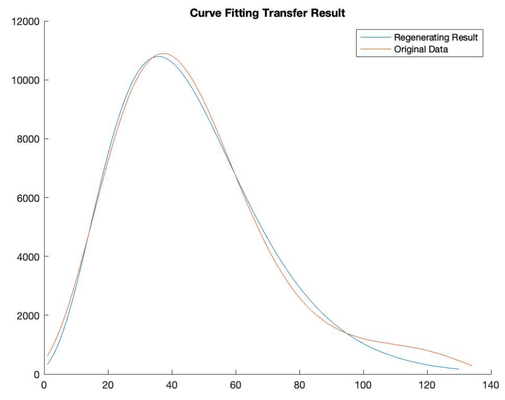

Figure 17: Received curve, curve-fitting

Discussion

From the experimental results, we can observe that with the curve-fitting approach, both of the transmitting speed and the robustness of the received signal are greatly improved.

However, there are still some issues about the curve-fitting approach: The first thing is, as we discussed in the previous parts in this report, the delay of the transmit system is an important aspect. If we want to do the curve-fitting at the transmitter side, we need to wait until the whole curve is generated, which makes a significant delay. Also, during the experiment, we were doing curve-fitting on a modern computer. For the remote control vehicle which is built with a simple microcontroller, the curve-fitting might be slow due to a high complexity of computation.

Also, in our experiment, we assume the curves have particular patterns, but in most of the cases, the patterns of the data are not predictable.

But the advantage of this approach is also obvious: it reduces the data size by nearly 30x for our test case. And no matter how long the original data is, curve-fitting approach can always reduce the data into fixed parameters.

Conclusion

In this report, we discussed several different approaches aim to compress the data size and speed up the transmitting. We noticed that, there are different aspects to measure the performance of each approach: speedup, accuracy, and delay.

Speedup can be represented by the reduction of total datasize, accuracy can be calculated by getting the sum of the error of each point(the curve-fitting result shows a really bad accuracy, about 10x than send the data directly without compression, but it is due to the phase difference between these two curves, their shapes are very similar), delay is hard to compute, it means when we can start to send the curve, how many related data do we need in the coding.

There are some trade-off among these different performances. Doing sampling and resolution reduction can obtain a good speedup without increase the delay, but we lost some accuracy. Doing curve-fitting can greatly speed up the transmitting, and get a better accuracy in a bad channel, but it introduces a large delay. Each method has its own benefits and limitations, so we need to choose the best approach based on the actual application.

Thanks for reading!

Bibliography

[1] GOLDSMITH, A. *Wireless Communications*. 2005.

[2] P.E. FRANDSEN, K. JONASSON, H. N., AND TINGLEFF, O. Unconstrained optimization, 3rd edition. *IMM, DTU* (2004).

[3] THE MATHWORKS, I. 16-qam with matlab functions. *https://www.mathworks.com/help/comm/gs/compute-ber-for-a-qam-system-with-awgn-using-matlab.html*.

[4] TIANMING CUI, H. Y. Curve-fitting algorithms. *From Lab assignments, haven't been published* (2018).

[3] [2] [1] [4]