Several Applications of Wireless Signals Using Channel State Information (CSI)

Yuning Mao
IEEE Honored Class
Shanghai Jiao Tong University
morningmoni@sjtu.edu.cn

Yuting Jia
IEEE Honored Class
Shanghai Jiao Tong University
hnxxjyt@sjtu.edu.cn

Zhenfeng Shi
IEEE Honored Class
Shanghai Jiao Tong University
XXX@sjtu.edu.cn

***Abstract*—Wireless signals (e.g., WiFi) are almost everywhere nowadays. However, the research of wireless signals using channel state information (CSI) has just started. In this project, we try several applications by leveraging the channel state information (CSI). We implement an end-to-end system which can be used to control electronic devices (e.g., laptop) by gesture recognition. We also do experiments of indoor localization using CSI. Our system leverages the WiFi signals using off-the-shelf network interface card (Intel 5300). It achieves an average accuracy of XXX % for a classification of 4? typical gestures and XXX% for localization of 5 different tables in the lab.**

***Keywords*—Gesture Recognition, Indoor Localization, Wireless Signals, Signal Processing, Classification**

I. Introduction

As the users shift from traditional PCs to mobile devices and their expectation for new methods of interaction increases, there are increasing demands for novel human-computer interfaces (HCI) through which the users can control various applications. Among those recently proposed methods, gesture recognition has gained much popularity. There are now successful commercial devices such as XBOX Kinect and Leap Motion. However, the cost of these commercial devices is relatively high and it also takes efforts to install and set up them. In addition, these devices only support in-sight gesture recognition since they are vision-based. Researchers also developed ways to move the sensors onto the body so as to bolster out-of-sight scenarios, and yet wearing sensors itself is inconvenient and infeasible in many cases.

Nowadays, wireless signals are almost everywhere. They are used for communication, remote control and so on. However, the research of doing gesture recognition by leveraging wireless signals just started in 2013. It partially results from the fact that there are plenty of challenges when it comes to the use of wireless signals. Wireless signals are usually unstable and sensitive to environmental changes, there could be issues because of medium contention, multipath interference, noises caused by other objects and so on. collecting wireless signals and processing them is hence challenging. Recently proposed WiFi-based systems are based on analyzing the changes caused by human motion in the characteristics of the wireless signals, such as the received signal strength indicator (RSSI) or detailed channel state information (CSI). We consider this a new direction of research and hence did some experiments by leveraging the CSI. The CSI is helpful for many purposes, such as gesture recognition, indoor localization, keystroke recognition, human activity detection and so on. In this paper, we propose an end-to-end gesture recognition system and implement a demo using off-the-shelf network interface card (Intel 5300) and *CSI Tools* [1].

Different from previous methods, we use machine learning techniques to support classification of different gestures and the accuracy of the classification can hence increase as the users provide more samples. We also support user-defined gestures. All the users need is to perform a gesture several times and then the system can store the pattern and recognize the gesture afterwards. In addition, We did experiment of indoor localization, or rather, classifying five tables in the lab. The results show that...

The rest of this paper is organized as follows. The background is introduced in Section II. The system design is illustrated in Section III. Section IV demonstrates the evaluation of the system by multiple experiments. The discussion concerning limitations, division of work, and future work is presented in Section V. Section VI concludes the paper.

II. Background

A. Channel State Information (CSI)

Modern WiFi devices that support IEEE 802.11n/ac standard typically consist of multiple transmit and multiple receive antennas and thus support MIMO. Each MIMO channel between each transmit-receive (TX-RX) antenna pair of a transmitter and receiver comprises of multiple subcarriers. These WiFi devices continuously monitor the state of the wireless channel to effectively perform transmit power allocations and rate adaptations for each individual MIMO stream such that the available capacity of the wireless channel is maximally utilized [2]. These devices quantify the state of the channel in terms of CSI values. The CSI values essentially characterize the Channel Frequency Response (CFR) for each subcarrier between each transmit-receive (TX-RX) antenna pair. Let M_T denote the number of transmit antennas, M_R denote the number of receive antennas and S_c denote the number of OFDM subcarriers. Let X_i and Y_i represent the M_T dimensional transmitted signal vector and M_R dimensional received signal vector, respectively, for subcarrier i and let Ni represent an M_R dimensional noise vector. An $M_R \times M_T$ MIMO system at any time instant can be represented by the following equation.

$$Y_i = H_i X_i + N_i, i \in [1, S_c] \tag{1}$$

In the equation above, the $M_R \times M_T$ dimensional channel matrix H_i represents the Channel State Information (CSI) for the subcarrier i. Any two communicating WiFi devices estimate this channel matrix H_i for every subcarrier by regularly transmitting a known preamble of OFDM symbols between each other [3]. For each Tx-Rx antenna pair, we can attain CSI values for S_c = 30 OFDM subcarriers of the 20 MHz WiFi Channel by using CSI Tools and Intel 5300 network interface card. This leads to 30 matrices with dimensions $M_R \times M_T$ per CSI sample.

B. Related Work

There are a myriad of work for device-free gesture recognition. Some are based on Received Signal Strength Indicator (RSSI), some on Channel State Information (CSI).

[4] claims itself to be the first to do gesture recognition using wireless signals. They presented a system named WiSee, which extracts minute Doppler shifts from wide-band OFDM transmissions to enable whole-home sensing and recognition of human gestures. There is another work *Allsee* [5] which leverages wireless signals furthermore and applies the method into RFID tags and power-harvesting sensors.

Another category is RSSI-based gesture recognition, which leverages the signal strength changes caused by human motion. However, due to the low resolution of RSSI values provided by commercial devices, the performance of these kinds of methods is usually considered relatively low. For instance, the accuracy is 56% over 7 different gestures in [6]. Nevertheless, [7] claims that they achieve 96% when evaluating the system using a multi-media player application. they also claim that they only need standard WiFi device, unlike *Wisee* which uses USRP in their experiment.

III. System Design

A. Overview

In this section, we describe the whole process and flow of work. It consists of data collection, data processing and model training. For data collection, we introduce the hardware and software tools and our work above them. For data processing, we discuss signal filtering methods we implemented. For model training, we illustrate how we select the features and model.

B. Data Collection

We use a Lenovo R400 laptop equipped with an Intel 5300 network card as the receive side of wireless signals. We measure the CSI of received packets using the CSI Tool at a sampling rate of 1000 HZ. For each segment of data, the CSI is represented in 30 channels, as a figure below:

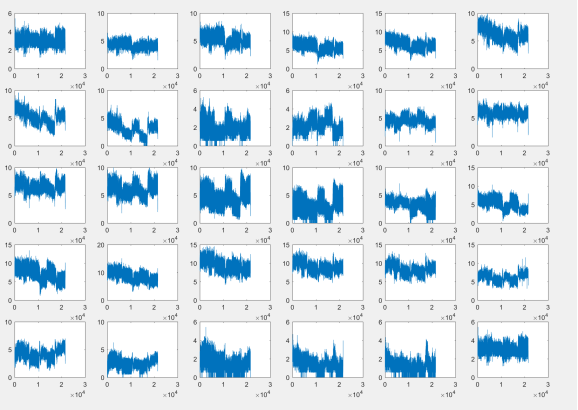


Fig. 1. Raw signal in 30 channel of CSI

From the above figure, we can see that the trend of the signal is obscure. Therefore, we need to preprocess the signals to attain signals with high SNR before we can do feature selection and model training.

The data format we can acquire is as follows [1]:

1) **timestamp_low** is the low 32 bits of the NIC's 1 MHz clock. It wraps about every 4300 seconds, or 72 minutes. This field was not yet recorded in the sample trace, so all values are arbitrary and always equal 4.
2) **bfee_count** is simply a count of the total number of beamforming measurements that have been recorded by the driver and sent to userspace. The netlink channel between the kernel and userspace is lossy, so these can be used to detect measurements that were dropped in this pipe.
3) **Nrx** represents the number of antennas used to receive the packet by this NIC, and Ntx represents the number of space/time streams transmitted. In this case, the sender sent a single-stream packet and the receiver used all 3 antennas to receive it.
4) **rssi_a, rssi_b, and rssi_c** correspond to RSSI measured by the receiving NIC at the input to each antenna port. This measurement is made during the packet preamble. This value is in dB relative to an internal reference; to get the received signal strength in dBm we must combine it with the Automatic Gain Control (AGC) setting (agc) in dB and also subtract off a magic constant. This process is explained below.
5) **perm** tells us how the NIC permuted the signals from the 3 receive antennas into the 3 RF chains that process the measurements. The sample value of [3 2 1] implies that Antenna C was sent to RF Chain A, Antenna B to Chain B, and Antenna A to Chain C. This operation is performed by an antenna selection module in the NIC and generally corresponds to ordering the antennas in decreasing order of RSSI.
6) **rate** is the rate at which the packet was sent, in the same format as the rate_n_flags defined above. Note that the antenna bits are omitted, as there is no way for the receiver to know which transmit antennas were used.
7) **csi** is the CSI itself, normalized to an internal reference. It is a $N_{tx} \times N_{rx} \times 30$ 3-D matrix where the third dimension is across 30 subcarriers in the OFDM channel. For a 20

MHz-wide channel, these correspond to about half the OFDM subcarriers, and for a 40 MHz-wide channel, this is about one in every 4 subcarriers. Which subcarriers were measured is defined by the IEEE 802.11n-2009 standard.

The CSI Tool provides supplementary programs written in Matlab to preprocess the signals.

It uses the following command to store the CSI into a binary file.

```
sudo linux-80211n-csitool-supplementary/netlink/log_to_file csi.dat
```

Then it uses Matlab to read the binary file and does the preprocessing.

```
csi_trace = read_bf_file('sample_data/log.all_csi.6.7.6');
plot(db(abs(squeeze(csi).')))
legend('RX Antenna A', 'RX Antenna B', 'RX Antenna C',
       'Location', 'SouthEast');
xlabel('Subcarrier index');
ylabel('SNR [dB]');
```

Which can display the SNR of three different antennas on subcarrier level.

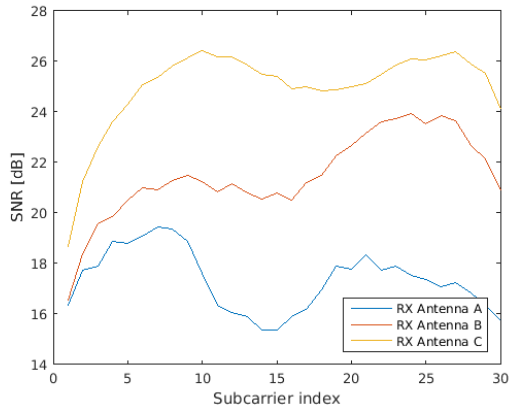


Fig. 2. The SNR of three antennas

To adjust the process above to our own purpose, we modified **log_to_file.c** to allow it to stop logging at certain time. We also used C to rewrite the functions in **read_bf_file.m** which were originally written in Matlab. MORE DETAIL HERE @JYT.

C. Data Processing

We use the amplitude of CSI from 30 subcarriers as the raw input. Then we do low-pass filtering and moving average filtering in turn.

1) Low-pass Filtering: Our raw data is sampled at a sampling rate of 1000 HZ, which is relatively high since the duration of typical human gestures is greater than hundreds of milliseconds. Therefore, we run CSI in the wireless signals through a low-pass filter to smooth out the fast varying noise while keeping the variations caused by gestures intact.

In our design, we use a low-pass filter with the following parameters:

Parameter	value
Sampling Frequency	1000 Hz
Passband Frequency	80 Hz
Stopband Frequency	100 Hz
Passband Ripple	1 dB
Stopband Attenuation	60 dB

TABLE I. PARAMETERS FOR LOW-PASS FILTER

The frequency response is shown as follows:

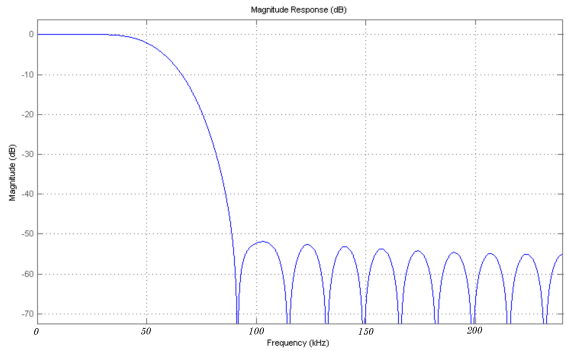


Fig. 3. Frequency Response

The result of passing raw signals to this filter is as follows:

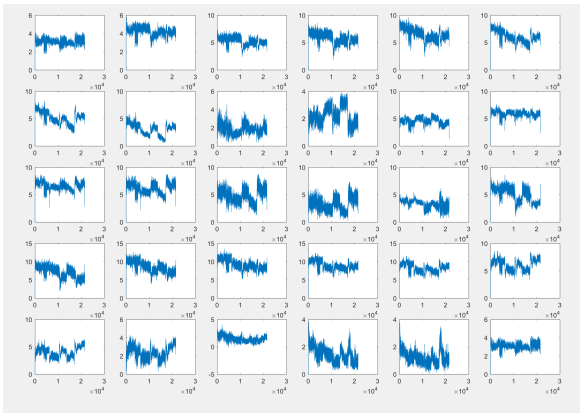


Fig. 4. Result of low-pass filter

Notably, we have implemented such low-pass filter both in Matlab and Python to meet our requirement. In Matlab, we directly use a build-in Butterworth filter but adjust the parameters to the needed ones. Matlab provides a more powerful and convenient method based on *fdesign* to design filters, compared to the traditional way like *fdatool*. As is illustrated above, only 3 lines of code can accomplish filter implementation.

Algorithm 1 Low-pass filter implement in MATLAB

```
function y=LowPassFilter(x)
    h = fdesign.lowpass(Fpass, Fstop, Apass, Astop, Fs)
    Hd = butter(h, 'MatchExactly', match)
    y = filter(Hd, x)
end
```

However, Matlab is slow to start and run. And since the model we use is written in Python, we turn to the implementation of python version. In Python, the implementation of such a low-pass filter is slightly harder than it in Matlab. We use an open-source package for mathematics, science and engineering called *Scipy*. In this package, it includes a toolbox called *Scipy.signal* which contains some filtering functions, a limited set of filter design tools and a few B-spline interpolation algorithms for one- and two-dimensional data.

In detail, the filter is implemented as a direct II transposed structure as follows:

$$a[0]*y[n] = b[0]*x[n]+b[1]*x[n-1]+...+b[nb]*x[n-nb]$$
$$-a[1]*y[n-1]-...-a[na]*y[n-na]$$

using the following difference equations:

```
y[m] = b[0]*x[m] + z[0,m-1]
z[0,m] = b[1]*x[m] + z[1,m-1] - a[1]*y[m]
...
z[n-3,m] = b[n-2]*x[m] + z[n-2,m-1] - a[n-2]*y[m]
z[n-2,m] = b[n-1]*x[m] - a[n-1]*y[m]
```

where m is the output sample number and $n = max(len(a), len(b))$ is the model order.

The rational transformation function describing this filter in the z-transform domain is:

$$Y(z) = \frac{b[0]b[1]z^{-1}+...+b[nb]z^{-nb}}{a[0]+a[1]z^{-1}+...+a[na]z^{-na}}X(z)$$

The pseudocode is as follows:

Algorithm 2 Low-pass filter implement in Python

```
def LowPass(cutoff, fs, order=5)
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff,
                  btype='low', analog=False)
    return b, a
def LowPassFilter(x, cutoff, fs, order=5)
    b, a = LowPass(cutoff, fs, order=order)
    y = lfilter(b, a, x)
    return y
```

2) Moving Average Filtering: Another serious problem is that minor changes in various environmental factors such as user location, distance between router and receiver, and objects in vicinity, can have an impact on the absolute WiFi channel amplitude. The amplitude changes we are interested in for gesture recognition, however, are independent of these absolute values. To extract these changes and remove bias from the absolute values, we filter the samples by calculating a windowed average of the channel samples (averaged over 300 ms, which is 300 points based on our sampling rate) from the low-pass filtered samples in the previous step.

The following figure shows the result of the moving average filter.

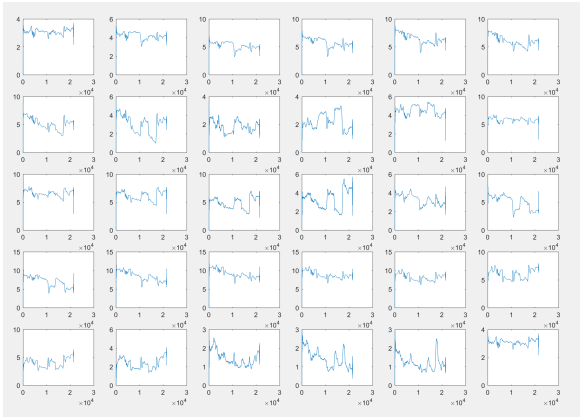


Fig. 5. Result of moving-average filter

From the above figure, we can clearly see the trend of the signal comparing to the raw data.

Similar to the low-pass filter, we also implement two moving average filter on Matlab and Python.

In MATLAB, we use a build-in filter which filters the input data, x, using a rational transformation function defined by the numerator and denominator coefficients b and a, respectively.

Here we design

```
b=ones(1, moving_points)/moving_points
a=1
```

In this case, the filter is identical to a moving average implementation.

The pseudocode is as follows:

Algorithm 3 Moving average filter implement in MATLAB

```
function y=MovingSmoothing(x)
    moving_points = 301;
    y = filter(ones(1, moving_point)/moving_point, 1, x);
end
```

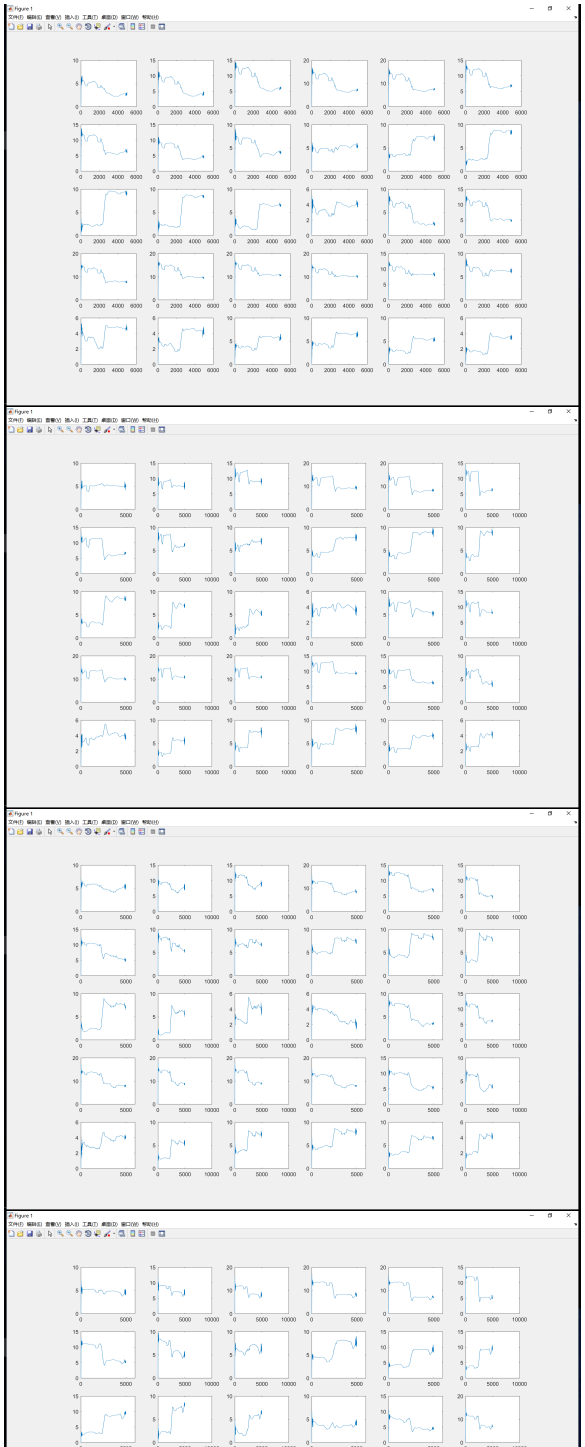
In Python, no such build-in function can be call to fulfill the filtering. Therefore, we implement it by ourselves, which has a complexity of $O(n)$.

The pseudocode is as follows:

3) Some Example Result of Preprocessing: In this section, we illustrate some figures of the result of preprocessing procedure described as above.

Algorithm 4 Moving average filter implement in Python

```
def MeanMovingSmoothing(x, windowSize):
    y = np.zeros(x.size)
    for i in range(0, windowSize // 2):
        y[i] = x[i]
    for i in range((x.size - windowSize // 2), x.size):
        y[i] = x[i]
    summation = x[:windowSize].sum()
    y[windowSize // 2] = summation / windowSize
    for i in range(windowSize // 2 + 1
                   , x.size - windowSize // 2):
        summation = summation -
                   x[i - windowSize // 2 - 1]
                   + x[i + windowSize // 2]
        y[i] = summation / windowSize
    return y
```



The above 4 figures come from 4 independent identical gestures 'PUSH'. Clearly, we can see that these signals share a very high similarity in corresponding channels.

D. Feature Selection

After processing, the data we collect is a time series signal, and we cannot use it as feature vector to de training because there are also noise or useless information in these signals and the length of the series may have different length. So we have to extract some useful and equivlent length feature from these signals.

From the figure of the signal processed:

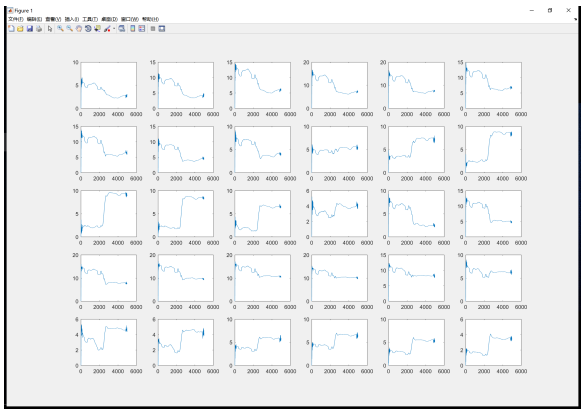


Fig. 7. Signal Processed

Intuitively and Easily, we can find that the continuous increase and decrease are very obvious and strong feature. So at first we try to extract some features relevant to continuous increase and decrease. But From Figure III-D, we can find that, even thought we have process and filtered data, there are also many vibrations. In these figure, red line means strength increasing while blue line means decreasing. We can find that there are many segments which look like a continuous increasing or decreasing segment, but actually they are not. So we have to do more filter first, or we cannot get the real and correct segment. What's more, because of the limitation of filtering, the first and last 150 points have not been filtered, so we can see that they vibrate vary much. And we need to throw them, or they will affect the feature selection process.

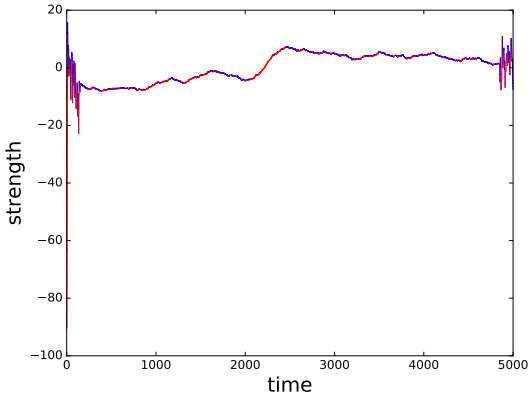


Fig. 8. Increase or Decrease

At first, we threw the first and last 10 percent of datas. And then, in this filter process, we choosed a easy but useful method. We conbine each 50 points into 1 point. In this case, the signal becomes more smooth and has less vibration. Figure III-D shows the curve of filtered data. Although there are also some little vibrations, but it's much easy for us to remove such little vibration.

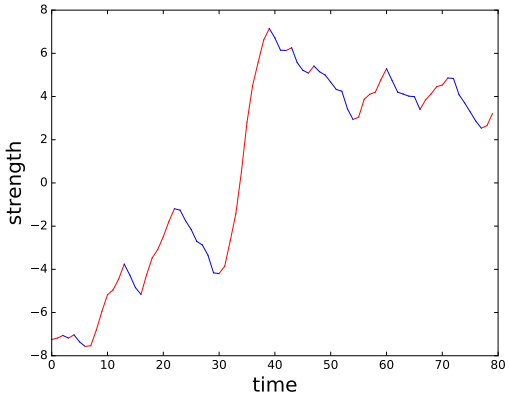


Fig. 9. Increase or Decrease of Filtered Data

Now, we can start to extract features from the data filtered just now. First, there are some continuous increase and decrease, so maybe the amplitude of increase and decrease contains some useful information. We have count the maximal increase and decrease, which showed in Figure III-D. From this figure, we can find that the gesture of 'zoomout' can be distincted from other gestures totally. And most of 'zoomin' are also different to other gestures. But for the remaining three types of gesture, we can not seperate them from others.

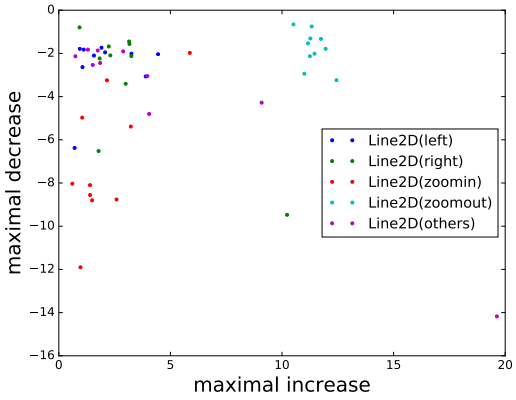


Fig. 10. Maximal Increase and Decrease

After this, we also tried some other features:

1) the maximal continuous change and it's duration, showed in Figure 1

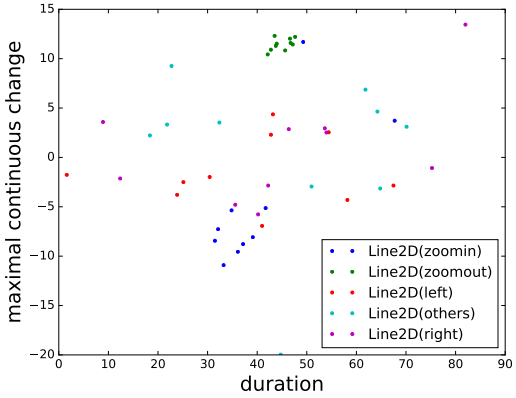


Fig. 11. Max Change to Duration

2) Number of value greater than $(max + mean)/2$ and number of value less than $(min + mean)/2$, showed in Figure 2

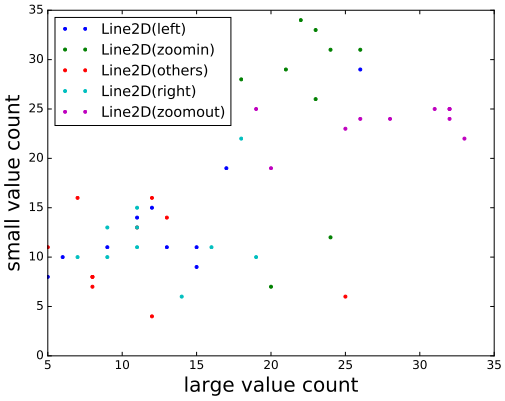


Fig. 12. Large Count to Less Count

3) max over mean and min over mean, showed in Figure 3

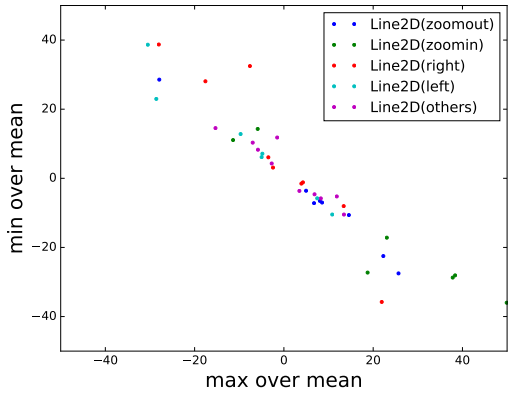


Fig. 13. Max over Mean to Min over Mean

4) mean and varance, showed in Figure 4

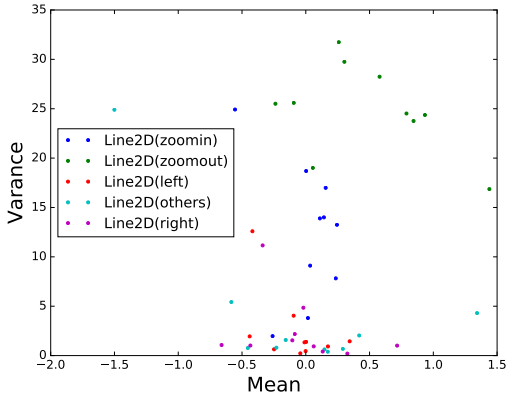


Fig. 14. Mean to Varance

5) number of increase, showed in Figure 5

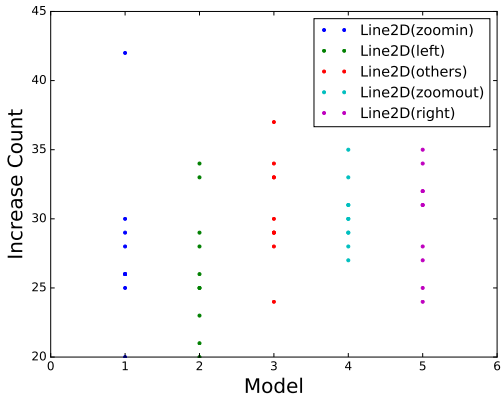


Fig. 15. Increase Count

6) average increasing per increasing change and average decreasing per decreasing change, showed in Figure 6

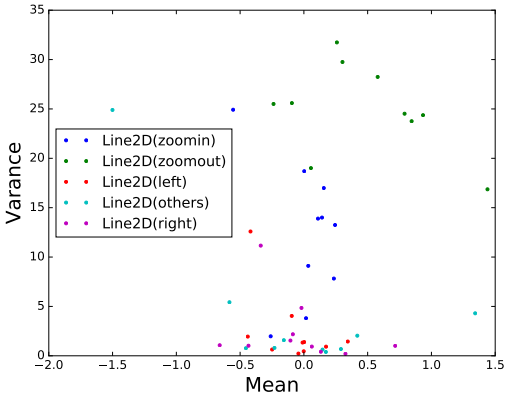


Fig. 16. Average Increase to Average Decrease

E. Model Training

From the figures showed above, we can find that many times we cannot seperate different gesture by linear classification. So finally we choosed SVM(Support Vector Machine) with Gaussian kernel.

1) SVM: In machine learning, support vector machines (SVMs, also support vector networks[1]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

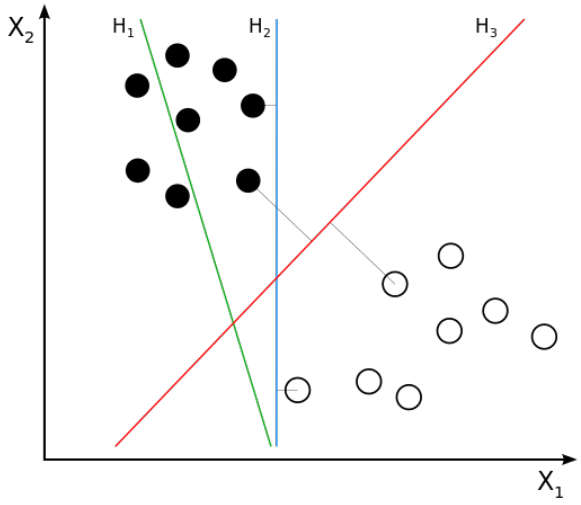


Fig. 17. SVM

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

2) Kernel Trick: The original maximum-margin hyperplane algorithm proposed by Vapnik in 1963 constructed a linear classifier. However, in 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick (originally proposed by Aizerman et al. [8]) to maximum-margin hyperplanes. [9] The resulting algorithm is formally similar, except that every dot product is replaced by a nonlinear kernel function. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. The transformation may be nonlinear and the transformed space high dimensional; although the classifier is a hyperplane in the transformed feature space, it may be nonlinear in the original input space.

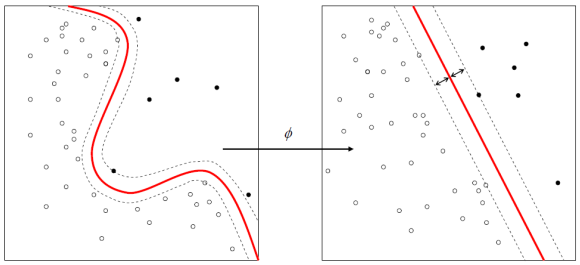


Fig. 18. Kernel Trick

It is noteworthy that working in a higher-dimensional feature space increases the generalization error of support vector machines, although given enough samples the algorithm still performs well. [10]

3) Cross Validation: In SVM, there are some hyper-parameters which need us to set manually. So we need to know which parameter can achieve the best performace. And Cross Validation is a pretty good way for us to get the best parameter setting.

Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (testing dataset). The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem), etc.

4) Model Persistance: The training process is a little slow. So if we want to predict the gesture in real-time, we cannot retrain the model once again. It means that we need to train out model first and then persist it as files so that when we need to use it, what should we do is just loading it and using it to predict.

5) Realization: To realize the training and persistance process, we choose to use 'python' and it's package 'sklearn'. We didn't realize it by ourselves because this part is not the focus of this course project, and the mature package has better performance and less bug than ourselves realization.

TABLE II. ACCURACY ON TRAINING SET AND TEST SET

	dataset1	dataset2	dataset3
cross validation	0.76	0.94	0.85
predicting test set	0.53	0.88	0.88

IV. EVALUATION

We have collected datas for three times. In each time, we set AP and computer to be different place, because we want to know that whether our algorithm will be sensitive to the location of AP and distance between AP and computer.

For these three different dataset, we split each one to be two parts, training set and test set. First we train our model by training set. In this process, we use cross validation to gain the best performance. And then we evaluate this model on test set. Table II shows the accuracy of cross validation and predicting on test set.

But if we mix these three datasets and repeat this process, The accuracy of cross validation and predicting test set dicrese to 0.42 and 0.34, which means that the signal we collect is very sensitive to the position of AP and distance between AP and computer. So it cannot be used in real word, because we cannot promise the position of AP and computer are same for each detection. And how to eliminate the influence of the position is our next direction. This problem is very hard to solve, but we will try our best to find a method to solve it.

V. DISCUSSION

In this section, we discuss what we learned from this project, in which division of work and future work are also covered.

A. Division of Work

Yuning Mao is the team leader. He took great efforts to the research and survey of papers in which CSI is used. He installed the hardware and set up the environment of CSI tool in Ubuntu 12.04. He tried things first and assigned tasks to the team members after having a basic understanding. He modified the program in CSI tool which was originally used to save CSI into binary file into a program which was then used to do gesture recognition. He is also responsible for most experiments and batch files.

Yuting Jia is in charge of the model, including feature selection, model selection, model training and so on. He also rewrote the C file which was originally called within *matlab* in the CSI tool so that we could leverage the CSI directly.

Zhenfeng Shi did the work of signal processing. He tried multiple methods of signal filtering and implemented them in both *matlab* and *python*.

B. Limitations and Future Work

Currently our system depends on the environment heavily and is sensitive to noise. In addition, we have to train different parameters of model in different scenarios. We think that there are a couple of reasons accounting for that. First, we couldn't find a quite and stable environment for data collection. The training data are hence with great noise, which is apparently undesirable. Second, the samples we collected are far from

enough compared to the researchers who have been working on this field for years. For instance, in [3], they obtained $10persons \times 30samples/person/classes \times 37classes = 11100$ samples. We only have around one percent of them in each experiment. After all we had to collect data after establishing the whole system, and there was only one week or so and the exams were around the corner... Future work directions are as follows. First, We need to collect more training data in a relatively stable environment. Second, we should try more advanced filtering methods via which we can hopefully eliminate random noise. Furthermore, we can try different models and different applications such as typing recognition and voice recognition, which are more challenging and exciting in the meantime!

VI. CONCLUSION

In this project, we did various experiments by leveraging channel state information (CSI) of WiFi signals using *CSI Tool* and Intel 5300 NIC. We install the Intel 5300 NIC into a lenovo R400 laptop and use external antennas to receive signals better. We process the CSI and use SVM to train models to classify four different gestures and acquire an average accuracy of ??%. In addition, we use CSI to localize tables in the lab and achieve an average accuracy of ??%. It's true that there are still flaws and limits in our system. But We indeed started from scratch without instructions from the experienced and did plenty of experiments. It's pretty cool to try and learn new stuff as awesome as this from the very beginning. We learned a lot from it.

ACKNOWLEDGMENT

We want to say thanks to Zhenyu Song, who let us know the CSI Tool and provide us with Intel 5300 NIC and laptop Lenovo R400. We also thank Prof.Jia for the WiFi router and instruction.

REFERENCES

[1] D. Halperin, W. Hu, A. Sheth, and D. Wetherall, "Tool release: gathering 802.11 n traces with channel state information," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 53–53, 2011.

[2] ——, "802.11 with multiple antennas for dummies," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 19–25, 2010.

[3] K. Ali, A. X. Liu, W. Wang, and M. Shahzad, "Keystroke recognition using wifi signals," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 2015, pp. 90–102.

[4] Q. Pu, S. Gupta, S. Gollakota, and S. Patel, "Whole-home gesture recognition using wireless signals," in *Proceedings of the 19th annual international conference on Mobile computing & networking*. ACM, 2013, pp. 27–38.

[5] B. Kellogg, V. Talla, and S. Gollakota, "Bringing gesture recognition to all devices," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 303–316.

[6] S. Sigg, U. Blanke, and G. Troster, "The telepathic phone: Frictionless activity recognition from wifi-rssi," in *Pervasive Computing and Communications (PerCom), 2014 IEEE International Conference on*. IEEE, 2014, pp. 148–155.

[7] H. Abdelnasser, M. Youssef, and K. A. Harras, "Wigest: A ubiquitous wifi-based gesture recognition system," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 1472–1480.

[8] M. A. Aizerman, E. A. Braverman, and L. Rozonoer, "Theoretical foundations of the potential function method in pattern recognition learning." in *Automation and Remote Control,*, no. 25, 1964, pp. 821–837.

[9] B. E. Boser and et al., "A training algorithm for optimal margin classifiers," in *PROCEEDINGS OF THE 5TH ANNUAL ACM WORKSHOP ON COMPUTATIONAL LEARNING THEORY*. ACM Press, 1992, pp. 144–152.

[10] C. Jin and L. Wang, "Dimensionality dependent pac-bayes margin bound," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1034–1042. [Online]. Available: http://papers.nips.cc/paper/4500-dimensionality-dependent-pac-bayes-margin-bound.pdf