COMP3516 Group Project

Yuemin Yu
yuyuemin@connect.hku.hk
"Mobile Computing is All You Need"
Hong Kong SAR, China

Entong He
u3584443@connect.hku.hk
"Mobile Computing is All You Need"
Hong Kong SAR, China

Xiaoyuan Han
u3584493@connect.hku.hk
"Mobile Computing is All You Need"
Hong Kong SAR, China

Zhejun Jiang
jasonj03@connect.hku.hk
"Mobile Computing is All You Need"
Hong Kong SAR, China

ABSTRACT

This report presents our system on the group project of HKU COMP3516, Data Analytics for IoT, 2024-25 Sem 2. In the system, we successfully implement required functionalities **on board**, including Channel State Information (CSI) collection via buffer, real-time motion detection and breathing rate estimation, efficient data transmission over wireless Message Queuing Telemetry Transport (MQTT) network and real-time web-application based visualization. Codes are available at https://github.com/YuyueminAustin/COMP3516-WiFi, permission granted upon request. A demo video is of free access at https://drive.google.com/file/d/1mEntCWfMyoapCzwKAsR8ixso2U6_kSmq/view?usp=sharing.

1 INTRODUCTION

WiFi Channel State Information (CSI) has been proved to be capable of capturing the human activity within indoor environment, due to the interference and superposition of the electromagnetic waves of the WiFi signals. Moreover, it is shown that more accurate sensing of humans can be done by leveraging the multipath effects [2].

Meanwhile, many available WiFi sensing systems are based on Intel 5300 network adapters, which may require extensive labour for hardware setup. In this project, we use ESP32, a lightweight WiFi CSI tool that is easier to use. Our contributions are following: **1)** we implement CSI collection via hardware buffer instead of naive serial port connection; **2)** we set up a wireless MQTT network, which contains one broker for hosting the data, one client on ESP32 board for publishing the data, and one on laptop for subscribing the data; **3)** we implement motion detection and breathing rate estimation *on board* in real time; **4)** we visualize the data we receive via web-based frontend application.

2 TECHNICAL PART

2.1 Hardware setups

As mentioned, we implement all of the data collection, processing, and computation steps on board.

2.1.1 *CSI collection.*

Buffered data retrieval . In this project, we don't utilize the phase angle information of the CSI samples. Therefore, when we are processing the CSI data, we only keep the amplitude information. In other words, if we receive a CSI sample $[r_0, i_0, r_1, i_1, r_2, i_2, \ldots, r_n, i_n]$, since the real and imaginary parts on each subcarrier are paired, we store $[(r_j{}^2 + i_j{}^2)]$ for each $j \in [1, 2, \ldots, n]$ where n is the number of subcarriers. In this system, $n = 56 + 1 = 57$ (56 subcarrier pairs + 1 direct current (DC) component), and the original CSI array admits length $114 = 57 \times 2$.

Sliding window buffering with FIFO replacement. Due to the limited computational resources on board, we adopted downsampling schemes. The original sampling frequency is 100Hz, which lead to over 800 samples if we set the processing window time to be 8 seconds. However, it is impossible to store such a large matrix of floats with size 800×57 due to the resource constraint. We thus downsample with 25Hz sampling rate (implemented by simply discarding 3/4 of the collected samples). In order to save the CSI samples of the latest time window for further processing, we also implemented a FIFO scheme instead of directly using the buffer provided. Specifically, the process is illustrated in Algo. 1.

The processed CSI data is stored in several global variables in the C program to ensure in-time data retrieval, and is updated as the sliding window proceeds. The following are the variables utilized, which correspond to the raw CSI amplitude samples, the samples with mean components removed, and the selected subcarriers with *top-10* gain values (which will be covered in §2.3).

```
#define NUM_FRAMES 200 // # of buffered CSI arrays
#define NUM_SUBCARRIERS 57 // # of subcarriers
static float csi_matrix[NUM_FRAMES][
    NUM_SUBCARRIERS]; // 2D matrix to store CSI
    data
static float removed_mean_csi[NUM_FRAMES][
    NUM_SUBCARRIERS]; // 2D matrix to store mean-
    removed CSI data
```

Algorithm 1: FIFO replacement of CSI matrix.

Data: csi_matrix, $filled\_frames$, $TOTAL\_FRAMES = 200$
/* The number of frames can also be adjusted to hold different time window lengths. */
Initialization;
while *device running* **do**
  receive CSI frame F;
  if $filled\_frames < TOTAL\_FRAMES$ **then**
    save F to $csi\_matrix[filled\_rows]$;
    $filled\_frames$ ++;
  else
    for $frame$ *in* csi_matrix **do**
      move $frame$ one row upwards;
    end
    save F to the last row of csi_matrix;
  end
end

```
static float csi_top10_subcarriers_t[10][
    NUM_FRAMES]; // 2D matrix to store top 10
    subcarriers
```

Listing 1: Global variables for CSI data processing

2.1.2 MQTT network implementation. After the computation of the breathing rate and motion statistics, we use MQTT protocol to send the output data via the wireless connection between the ESP32 board and the broker. To be specific, the ESP32 board publish the result to the MQTT Broker instance on the host laptop, and another MQTT client on host laptop is subscribing the messages. The messages received by the clients are then visualized via the web application. The overall architecture of the system is illustrated in Fig. 1.

2.2 Motion detection algorithm

The WiDetect motion detection algorithm is based on the statistical electromagnetic model proposed by [2]. For conciseness, we omit the details of the EM model, but focus on the motion detection based on the motion statistics.

2.2.1 Preprocessing CSI with motion. Suppose the CSI data collected is indexed by $H(t, f)$, while we omit other parameters in the dataframe. The corresponding power response $G(t, f) = |H(f, t)|^2$ (c.f. §2.1.1).

We use the sample auto-covariance function $\hat{\gamma}_G(\tau, f)$ where τ is the lag (c.f. Eqn. (2) in [2]), which is defined as

$$\hat{\gamma}_G(\tau, f) = \frac{1}{T} \sum_{t=\tau+1}^{T} (G(t-\tau, f) - \overline{G}(f))(G(t, f) - \overline{G}(f)),$$

where T is the number of samples (within a window).

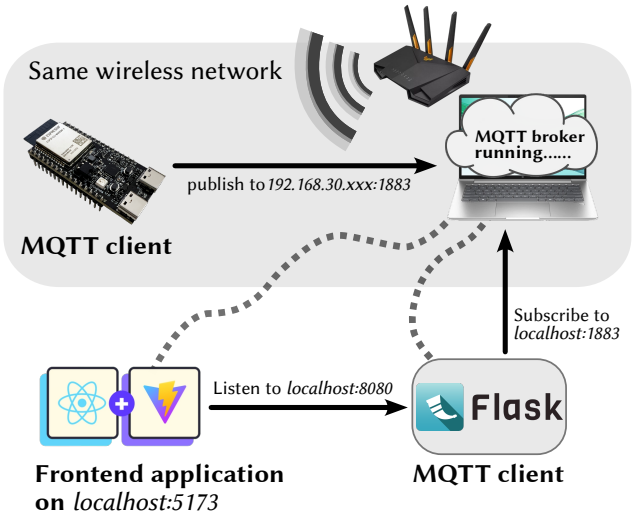

Figure 1: The overall architecture of the web-based system.

2.2.2 Detection rule . The ACF of $G(t, f)$ characterized by the lag $\tau > 0$ is defined as

$$\rho_G(\tau, f) = \frac{\hat{\gamma}_G(\tau, f)}{\hat{\gamma}_G(0, f)}.$$

Whenever the calculation of $\rho_G(\tau, f)$ fails (*i.e.*, $\hat{\gamma}_G(0.f) = 0$), we discard this subcarrier. The motion detection is performed by hypothesis testing. Denote the set of subcarriers as $\mathscr{F}$, for each $f \in \mathscr{F}$, we define the **motion statistic** as

$$\phi(f) = \rho_G\left(\frac{1}{F_s}, f\right).$$

To perform the following **hypothesis testing**:

$$H_0: \quad \hat{\phi}(f) \sim \mathcal{N}(-1/T, 1/T), \quad \forall f \in \mathscr{F};$$
$$H_1: \quad \hat{\phi}(f) \xrightarrow{\tau \to 0} \eta(f) > 0, \quad \forall f \in \mathscr{F},$$

we use the motion statistic estimator

$$\hat{\psi} = \mathbb{E}\left[\hat{\phi}(f)\right] = \frac{1}{|\mathscr{F}|} \sum_{f \in \mathscr{F}} \hat{\phi}(f).$$

The decision rule for motion (1) or static (0) is given by $\mathbb{I}\left\{\hat{\psi} > \eta\right\}$ for an appropriate choice of parameter η.

2.3 Breathing rate estimation algorithm

Our breathing rate detection (in breathing rate per minute (BPM)) algorithm resembles the conventional approach, with modifications to eliminate subcarriers with ***inferior*** data.

2.3.1 Weighting with maximum-ratio combination. Given the power response $G(t, f)$, we hope to distinguish those subcarriers with the most effective data according to some

specific metric. Following [1], we utilize the **Maximum-Ratio Combination** (MRC) to combine the subcarrier data.

To boost the ACF effectiveness, we use the motion statistics (c.f. §2.2.2) as the weight. Specifically, each subcarrier $f \in \mathcal{F}$ is assigned a weight $w_f = \phi(f)/\sum_{f' \in \mathcal{F}} \phi(f')$. Finally, the data on all subcarriers is thus combined to obtain the **MRC ACF** $\rho_G^{\mathrm{MRC}}(\tau)$:

$$\rho_G^{\mathrm{MRC}}(\tau) = \sum_{f \in \mathcal{F}} \frac{\phi(f)}{\sum_{f' \in \mathcal{F}} \phi(f')} \hat{\gamma}_G(\tau, f).$$

Finally, by routinely detecting the first peak index $\tau_{\mathrm{peak}} \in [T]$ of $\rho_G^{\mathrm{MRC}}(\tau)$, the estimated BPM is given by

$$\widehat{\mathrm{BPM}} = 60 \times \frac{F_s}{\tau_{\mathrm{peak}}}.$$

Notably, since the typical BPM of human resides in the interval $[8, 30]$, we can narrow the feasible region of peak finding to $[\tau_{\min}, \tau_{\max}] \cap [T]$, where $\tau_{\min} = \lfloor 60 \times \frac{F_s}{30} \rfloor$ and $\tau_{\max} = \lceil 60 \times \frac{F_s}{8} \rceil$.

In our program, we use a `find_first_peak` function parameterized by several identities to provide adaptive peak finding for signals with different shapes. Its prototype (in `app_main.c`) is given as follows:

```
int find_first_peak(const float *ACF, float
    min_height, int min_width, float prominence,
    int sampling_rate, int window_length);
// ACF: input MRC ACF data, min_height: minimum
    data value at the peak, min_width: radius of
    the neighbor to examine whether a tentative
    peak is the maximum value in it, prominence:
    minimum prominence of the peak, sampling_rate:
     used to determine tau_min and tau_max,
    window_length: size of the window to use to
    smooth the data prior to finding the peak.
```

Listing 2: Our peak finding function prototype.

2.3.2 Improve performance by truncation. In our on-board test, computing $\rho_G^{\mathrm{MRC}}(\tau)$ can be expensive. Moreover, we observe that some subcarriers yield negligible (even negative) motion statistics, which makes it ridiculous to combine them into the MRC ACF. To this end, we consider a "killing two birds with one stone" settlement: truncating the ACF data by only keeping those with the highest M motion statistics. We denote the corresponding subcarrier set as $\mathcal{F}_M \subseteq \mathcal{F}$. In our code implementation, we apply a bubble sort (since $|\mathcal{F}|$ is small here) on the ACF matrix according to $\rho_G(1, f)$, and replicate the operations on the index set $[|\mathcal{F}|]$ to find out the top M subcarriers. Unless otherwise stated, we fix $M = 10$ in all the on-board tests (c.f. Listing 1).

3 EVALUATION

3.1 Evaluation result

We conducted the motion detection and breathing rate estimation algorithms on board. The devices are deployed on the table in the computer lab at Room 311, Haking Wong Building. For details, see Figure 2.

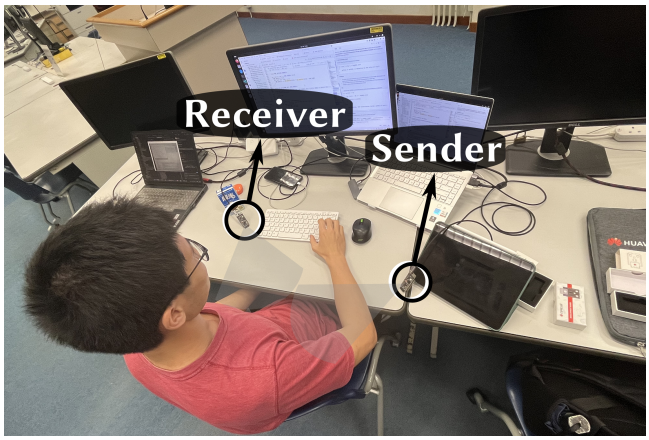

Figure 2: Setup of our on-board test.

The evaluation result is shown in Table 1. We observe that motion detection is comparatively more robust, whereas breathing rate estimation is hindered by poor data quality, sometimes resulting in undetected peaks. The data collection procedure is the main burden of the performance. Future work includes resolving this problem by adjusting the device calibration subject to the testing circumstance geometry.

Table 1: Overall evaluation result.

Evaluation Dataset	Result
Motion detection	> 80% (outliers removed)
Breathing rate estimation	MAE ≈ 2 BPM

3.2 Benchmark result

All benchmark results are computed by running the on-board algorithms written in C on a Windows laptop with the input parsed by Python programs. Codes are available at https://github.com/HeEntong/COMP3516-Group-Project-Local-Test-Code.

3.2.1 Motion test. We conduct the motion detection on the whole benchmark dataset with sampling rate F_s being 100Hz, and set the window size to be 200, the step size to be 25, the lag $\tau = 1$ (such that it corresponds to exactly $1/F_s$ in real time), and the threshold $\eta = 0.4$. The algorithm reports an

Table 2: Test result of motion detection. First block: ground truth = 'motion'; Second block: ground truth = 'static'.

File Name	Result	File Name	Result
204932.csv	100%	205301.csv	100%
205318.csv	100%	205330.csv	100%
205343.csv	100%	205356.csv	100%
205406.csv	100%	205439.csv	80%
205451.csv	100%	205504.csv	100%
205516.csv	100%	205526.csv	100%
205539.csv	100%	205553.csv	100%
205607.csv	100%	205624.csv	100%
205637.csv	100%	205645.csv	100%
205654.csv	100%	205704.csv	100%
203408.csv	100%	204824.csv	100%
210608.csv	100%	210623.csv	80%
210645.csv	100%	210747.csv	100%
210802.csv	100%	210814.csv	100%
210840.csv	100%	210854.csv	100%

accuracy of 98% on the "static" benchmarks and 99% on the "motion" benchmarks. Accuracy is reported in Table 2.

3.2.2 Breathing rate test. We conduct the breathing rate estimation on the downsampled benchmark with 25Hz sampling rate (take 1 out of every 4 consecutive data frames). During the data parsing procedure, we observe that the provided signal is noisy, so we apply a windowed filter with width 4 on the CSI amplitude data prior to ACF calculation. Moreover, as the number of estimations of our program might deviate from that of the provided ground truth, we use the optimal time horizon alignment with the minimum MAE. Estimation and ground truth are reported in Figure 3 and Figure 4.

4 CONCLUSION

In this project, we implemented an on-board WiFi sensing system using ESP32 to capture human motion and breathing rates via Channel State Information (CSI). By leveraging real-time algorithms for motion detection (> 80% accuracy) and breathing rate estimation (MAE ≈ 2 BPM) and efficient MQTT-based data transmission based on buffered data retrieval, we demonstrated an end-to-end IoT solution with real-time web visualization. While challenges such as data quality and computational constraints impacted the performance, the system provides a foundation for future enhancements, including improved calibration and noise reduction.

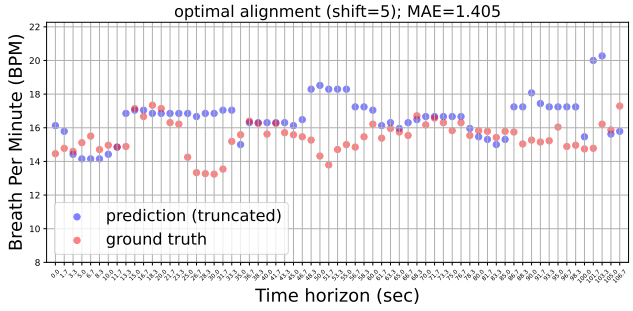

Figure 3: Estimated respiration for 191018.csv. Peak detection parameters: `min_height` = 0.01, `min_width` = 4, `prominence` = 0.08, `window_length` = 4.

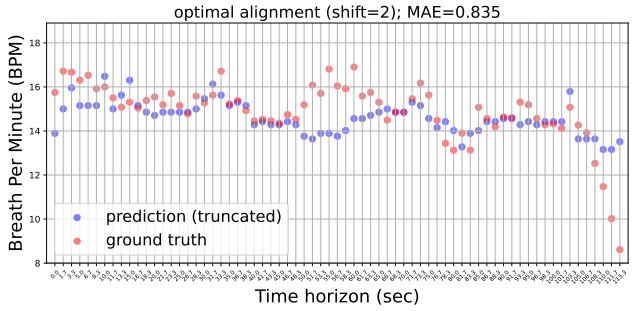

Figure 4: Estimated respiration for 193124.csv. Peak detection parameters: `min_height` = 0.02, `min_width` = 2, `prominence` = 0.1, `window_length` = 12.

ACKNOWLEDGMENT

E. He would like to express his heartfelt thanks to Y. Yu, who worked throughout the night without sleeping to set up the hardware. The project would not have been so successful without his leadership and efforts. E. He acknowledges the fifth group project collaboration with Y. Yu.

REFERENCES

[1] Xiaolu Zeng, Beibei Wang, Chenshu Wu, Sai Deepika Regani, and K. J. Ray Liu. 2022. WiCPD: Wireless Child Presence Detection System for Smart Cars. *IEEE Internet of Things Journal* 9, 24 (Dec. 2022), 24866–24881. https://doi.org/10.1109/jiot.2022.3194873

[2] Feng Zhang, Chenshu Wu, Beibei Wang, Hung-Quoc Lai, Yi Han, and K. J. Ray Liu. 2019. WiDetect: Robust Motion Detection with a Statistical Electromagnetic Model. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 3, Article 122 (Sept. 2019), 24 pages. https://doi.org/10.1145/3351280