Group Project: Wi-Fi Sensing via ESP32-C5

Bai Junhao
bjh2001@connect.hku.hk
helloworld
Hong Kong

Long Qian
u3638055@connect.hku.hk
helloworld
Hong Kong

SHI Xianjie
u3638000@connect.hku.hk
helloworld
Hong Kong

Wei Shuang
u3641481@connect.hku.hk
helloworld
Hong Kong

1 INDIVIDUAL CONTRIBUTION

Table 1: Individual Contribution

Name	UID	Contribution Statement
Bai Junhao	3036382909	25%, implemented on-board algorithms, processed data, designed the webpage and overall design.
Long Qian	3036380559	25%, compiled materials, produced the final document, transferred data and designed the webpage.
Shi Xianjie	3036380004	25%, implemented real-time breathing rate estimation, generated graphs.
Wei Shuang	3036414817	25%, implemented real-time motion detection, generated graphs.

2 OVERALL RESULT

2.1 Evaluation Result

Table 2: Overall Evaluation Result.

Evaluation Dataset	Result
Motion Detection	Accuracy: 100(%)
Breathing Rate Estimation	median MAE: 0.85 BPM

2.2 Test Result

2.2.1 Breathing Rate Test. Here we plot three figures, Fig.1-3, of our estimated breathing rate, whose titles are the three test files.

2.2.2 Motion Test. Here are shown the results (1 for motion detected, 0 for no) in the table 3.

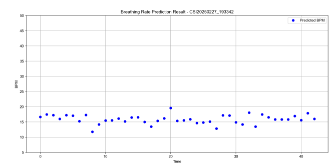


Figure 1: Estimated Respiration for 193342.csv

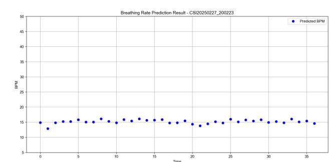


Figure 2: Estimated Respiration for 200223.csv.

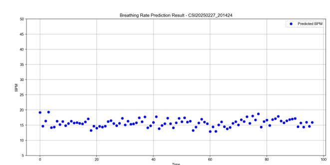


Figure 3: Estimated Respiration for 201424.csv.

Table 3: Test Result of Motion Detection.

File Name	Result	File Name	Result
205713.csv	1	205723.csv	1
205733.csv	1	205803.csv	1
205822.csv	1	205834.csv	1
205845.csv	1	205855.csv	1
205906.csv	1	205928.csv	1
205943.csv	1	205958.csv	1
210036.csv	1	210911.csv	0
210928.csv	0	210942.csv	0
211010.csv	0	211023.csv	0
211035.csv	0	211055.csv	0
211107.csv	0		

3 SYSTEM DESIGN AND ANALYSIS

3.1 CSI Collection

Screenshots for this part of implement can be refered to in Appendix

3.1.1 Target setting.

3.1.2 Find the port (COM6 for sender, COM5 for receiver).

3.1.3 Build and flash (Sender).

3.1.4 Build and flash (Receiver).

3.1.5 Connected to WiFi AP.

3.1.6 Power up the sender, then the receiver gets CSI data.

3.2 Data Transmission

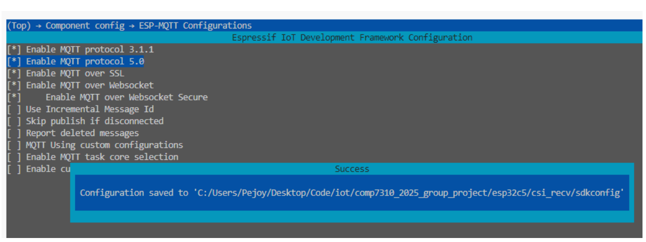


Figure 4: MQTT5

3.2.1 modify idf.py menuconfig to set MQTT5 Protocol .

3.2.2 Set Wifi frequency to 2.4GHz.

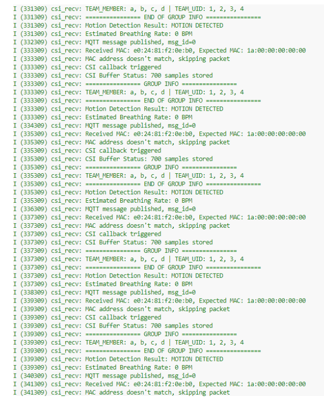


Figure 5: Sent Messages

3.2.3 It managed to send messages after getting the IP address.

3.3 Motion Detection

For the implementation of motion detection, the most intuitive guess is that the intensity of Received Signal Strength Indicator (RSSI) will fluctuate greatly during motion activities, which is also a feature mentioned in the course materials.

Based on the above idea, we conducted a statistical analysis on the given motion data set and static data set. It can be found that for the motion state, the standard deviation range of RSSI is 1.61 to 6.21, which is significantly different from 0.0 to 0.49 for the static state. Therefore, in the dataset-oriented task, we used 1.0 as the threshold of the RSSI standard deviation to distinguish between motion and static states.

Our prediction accuracy on the training set is 100

3.4 Breathing Rate Estimation

Before implementing the real-time algorithm, we first wrote the code in Python on a PC and verified the algorithm logic through evaluation.

The script used to calculate the MAE for the data in the `evaluation` folder and the other script to process CSI data in the `test` directory and plot the related figures. They should be placed at the same directory level as the `benchmark` folder, the required Python libraries and their corresponding versions are shown in below:

numpy: 1.26.4
pandas: 2.2.1
matplotlib: 3.7.0
scipy: 1.10.0.

```python
# Clean data and convert to complex numbers
csi_str = str(csi_str).strip('[]"')
elements = [float(x) for x in csi_str.split(',')
    if x.strip()]

```

```python
# Validate data integrity
if len(elements) % 2 != 0:
raise ValueError(f"CSI data length error:
    currently {len(elements)} elements")

# Construct complex pairs and compute amplitude
magnitudes = [abs(complex(real, imag))
  for imag, real in zip(elements[::2], elements
    [1::2])]
return magnitudes
```

The above describes the processing of the `data` column in the CSI file. For each row, the real and imaginary parts are extracted in pairs to calculate the magnitude, and the results are stored in the `magnitudes` matrix. The size of the `magnitudes` matrix is M×N, where M is the number of rows in the CSI file, and N is 117 — which is the length of each `data` entry (len = 234) divided by 2.

```python
for time in range(0, len(csi_amplitudes), 1):
  left_time = add_seconds(base_time, time)
  right_time = add_seconds(left_time, window_size)
  l_win = find_timeidx(timestamps, left_time)
  r_win = find_timeidx(timestamps, right_time)
  if r_win == -1:
    break
  window = csi_amplitudes[l_win:r_win]

  fs = int(len(window) / window_size)
  bpm = estimate_breathing_rate(window, fs)
  pred_bpm[right_time] = bpm
```

The above is the main loop function for the sliding window. The timestamps array stores the timestamp column from the CSI file, and the base_time variable is set to the first value of timestamp. In the loop, left_time is calculated by adding the current loop index to base_time, which means the window slides forward with a step size of 1. The corresponding right_time is equal to left_time plus the window length of 15 seconds. The find_timeidx function helps locate the index of the first occurrence of a given time in the timestamps array, allowing us to extract the current window (window) and compute its sampling rate fs.

pred_bpm is a dictionary. After calculating the BPM for the current window, we store the result using right_time as the key. This makes it convenient to match the corresponding ground truth BPM from the GT file later, facilitating further analysis and processing.

```python
def bandpass_filter(signal, fs):
    cutoff_freq = [12/60, 22/60]
    # Breathing rate frequency band: 12-22 BPM
    sos = butter(1, cutoff_freq, 'band', fs=fs,
    output='sos')
    return sosfiltfilt(sos, signal)

def autocorrelation(signal, fs):
    signal = signal - np.nanmean(signal)
    # Remove DC component
    signal = np.nan_to_num(signal)

    filtered = bandpass_filter(signal, fs)
    norm_signal = filtered / (np.std(filtered) + 1
    e-9)

    corr = np.correlate(norm_signal, norm_signal,
    mode='full')
    return corr[len(corr)//2:]

def estimate_breathing_rate(csi_matrix, fs):
    bpm_list = []

    for i in range(csi_matrix.shape[1]):
    # Iterate over subcarriers
        signal = csi_matrix[:, i]
        acf = autocorrelation(signal, fs)
        peaks, _ = find_peaks(acf, prominence
    =0.01)

        if len(peaks) == 0:
            continue

        period = peaks[0] / fs
        bpm = 60 / period
        bpm_list.append(bpm)

    # Fuse multiple subcarriers using the median
    if bpm_list:
        return np.median(bpm_list)
    else:
        return np.nan
```

The above describes the function used to calculate the breathing rate from a windowed signal. The main function is estimate_breathing_rate(), where the input parameter csi_matrix has a size of m×n, with m representing the window length (window_size, set to 15 in this project) and n representing the number of subcarriers, which equals 117.

Within the main function's loop, each subcarrier in the windowed signal is processed one by one. For each subcarrier, the autocorrelation() function is called. This function performs DC offset removal, applies band-pass filtering (the filter range is [12/60, 22/60], which aims to suppress signal interference outside the 12 BPM to 22 BPM range), and finally normalizes the filtered signal before computing its autocorrelation.

After obtaining the autocorrelation result, peak detection is performed on the signal. The index of the first detected peak is used to calculate the corresponding BPM value for that subcarrier, and the result is added to bpm_list.

In the end, the bpm_list contains all BPM values estimated from the subcarriers within the window. The median of bpm_list is selected as the final result of multi-subcarrier feature fusion, representing the estimated BPM for the current signal window.

```
# Calculate evaluation metrics
abs_sumdiff = np.abs(aligned_pred - aligned_gt)

results.append({
  'File': csi_file.name,
  'MAE': np.nanmean(abs_sumdiff),
  'Data points': len(aligned_pred),
  'Maximum error': np.nanmax(abs_sumdiff)
})
```

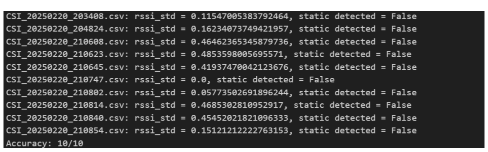
The above is the code for calculating the MAE value of each CSI file, where `aligned_pred` and `aligned_gt` represent the predicted and ground truth BPM arrays that have been aligned based on timestamps.

The final results of the evaluation can be referred to in Figure 8

4 RESULTS VISUALIZATION

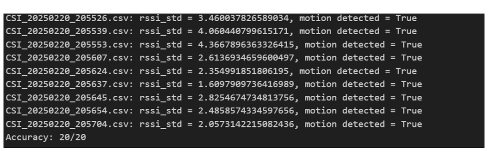
4.1 Motion Detection

The results of Motion Detection can be found in file .ipynb we submitted. Here, we provide some screenshots for reference.



```
CSI_20250220_203408.csv: rssi_std = 0.11547005383792464, static detected = False
CSI_20250220_204824.csv: rssi_std = 0.16234073749421957, static detected = False
CSI_20250220_210608.csv: rssi_std = 0.46462365345879736, static detected = False
CSI_20250220_210623.csv: rssi_std = 0.4853598005695571, static detected = False
CSI_20250220_210645.csv: rssi_std = 0.41937470042123676, static detected = False
CSI_20250220_210747.csv: rssi_std = 0.0, static detected = False
CSI_20250220_210802.csv: rssi_std = 0.05773502691896244, static detected = False
CSI_20250220_210814.csv: rssi_std = 0.4685302810952917, static detected = False
CSI_20250220_210840.csv: rssi_std = 0.4545202182109633, static detected = False
CSI_20250220_210854.csv: rssi_std = 0.1512121222276315, static detected = False
Accuracy: 10/10
```

Figure 6: Static detection



```
CSI_20250220_205526.csv: rssi_std = 3.460037826589034, motion detected = True
CSI_20250220_205539.csv: rssi_std = 4.060440799615171, motion detected = True
CSI_20250220_205553.csv: rssi_std = 4.3667896363326415, motion detected = True
CSI_20250220_205607.csv: rssi_std = 2.6136934659600497, motion detected = True
CSI_20250220_205624.csv: rssi_std = 2.354991851806195, motion detected = True
CSI_20250220_205637.csv: rssi_std = 1.6097909736416989, motion detected = True
CSI_20250220_205645.csv: rssi_std = 2.8254674734813756, motion detected = True
CSI_20250220_205654.csv: rssi_std = 2.4858574334597656, motion detected = True
CSI_20250220_205704.csv: rssi_std = 2.0573142215082436, motion detected = True
Accuracy: 20/20
```

Figure 7: motion detection

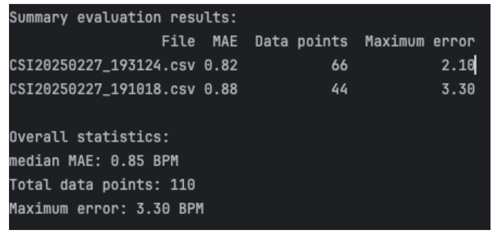
4.2 Breathing Rate Estimation

The MAE values for the two files are 0.82 and 0.88 respectively, both lower than 0.94. In addition, we provide the average breathing rate in bpm of each file, they are shown below:

193342.csv Mean BPM: 15.89;

200223.csv Mean BPM: 15.21;

201424.csv Mean BPM: 15.77.

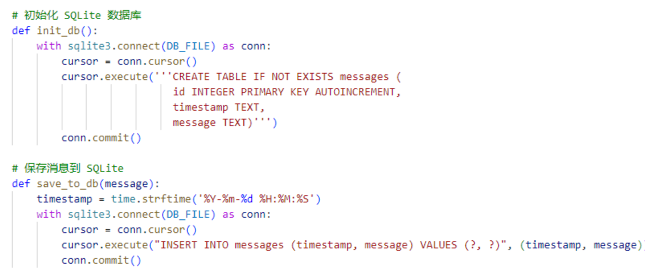


```
Summary evaluation results:
                    File  MAE  Data points  Maximum error
CSI20250227_193124.csv 0.82           66           2.10
CSI20250227_191018.csv 0.88           44           3.30

Overall statistics:
median MAE: 0.85 BPM
Total data points: 110
Maximum error: 3.30 BPM
```

Figure 8: Final evaluation results

4.3 Data Visualization



```
# 初始化 SQLite 数据库
def init_db():
    with sqlite3.connect(DB_FILE) as conn:
        cursor = conn.cursor()
        cursor.execute('''CREATE TABLE IF NOT EXISTS messages (
                          id INTEGER PRIMARY KEY AUTOINCREMENT,
                          timestamp TEXT,
                          message TEXT)''')
        conn.commit()

# 保存消息到 SQLite
def save_to_db(message):
    timestamp = time.strftime('%Y-%m-%d %H:%M:%S')
    with sqlite3.connect(DB_FILE) as conn:
        cursor = conn.cursor()
        cursor.execute("INSERT INTO messages (timestamp, message) VALUES (?, ?)", (timestamp, message))
        conn.commit()
```

Figure 9: Get data and store them

4.3.1 *Get data from MQTT and store them in SQLite3.*

4.3.2 *Flask : messages port is used to search history.*

4.3.3 *Front end: Vue/D3.js.*

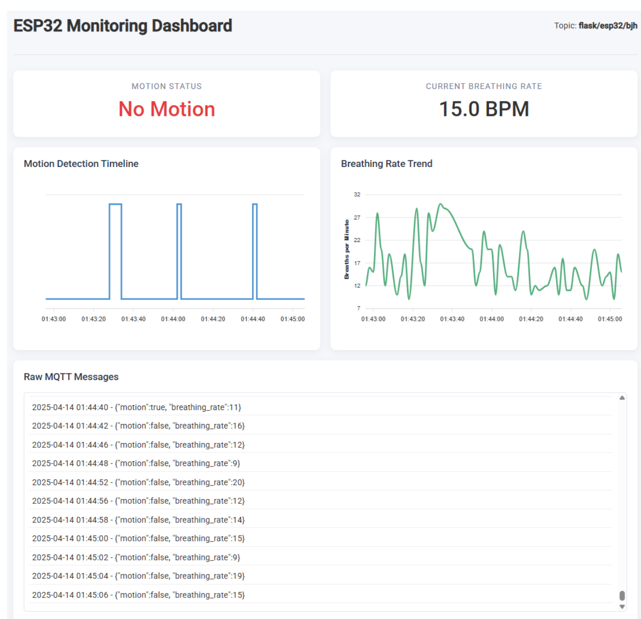


Figure 10: Dashboard

4.3.4 *Final Dashboard.*

5 APPENDIX

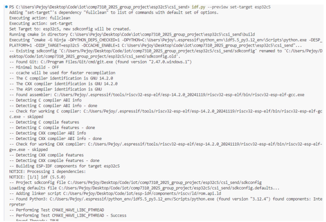


Figure 11: Target setting

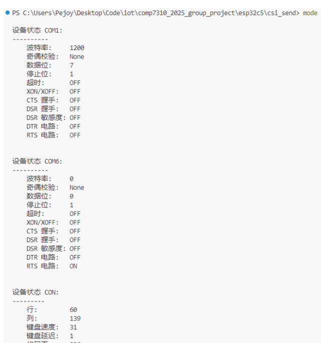


Figure 12: Find the port (COM6 for sender, COM5 for receiver)

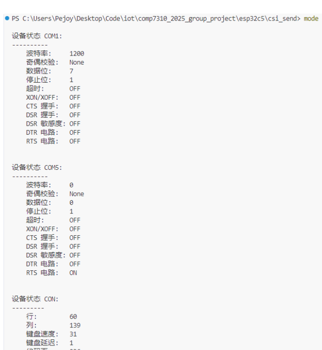


Figure 13: Find the port (COM6 for sender, COM5 for receiver)

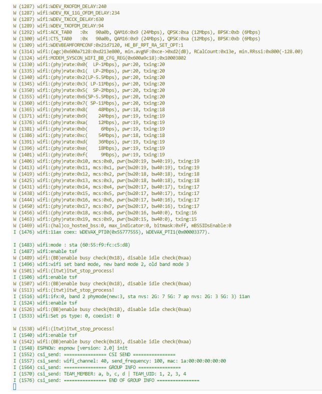


Figure 14: Build and flash (Sender)

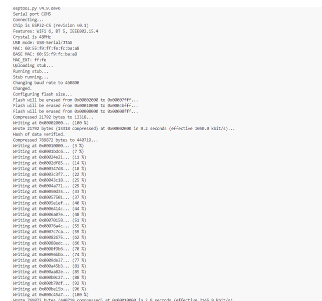


Figure 15: Build and flash (Receiver)

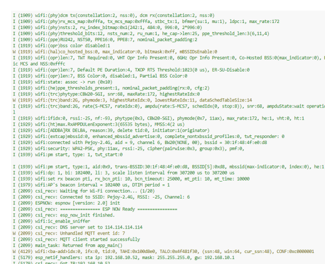


Figure 16: Connected to WiFi AP

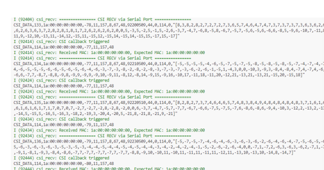


Figure 17: Power up the sender, then the receiver gets CSI data