Detecting and Locating Stress in Urban Settings

Daniel Namaki Ghaneh, Emanuele Respino, Gianmaria Saggini, Niccolò Settimelli
d.namakighaneh@studenti.unipi.it, e.respino1@studenti.unipi.it, g.saggini1@studenti.unipi.it, n.settimelli@studenti.unipi.it

ABSTRACT

In modern society, stress has become one of the main issues affecting both mental and physical health. The ability to detect and manage stress levels can help individuals to improve their daily life. In response to the problem, this project developed ChillIn, an Android mobile application designed with the intent of tracking and localizing stress in an urban environment. The application exploits the potential of wearable devices and their sensors to track stress within urban environments, correlating the data acquired with geographic locations: by aggregating the stress data of different individuals, the application can highlight areas of a city that tend to induce stress. Through its user-friendly interface and real-time stress monitoring features, the project offers a perspective on urban dynamics, making it possible to identify stress hotspots: this presents opportunities for urban planning initiatives aimed at improving community well-being and enhancing overall quality of life, offering an opportunity for promoting mental health awareness and stress mitigation strategies.

1 Introduction

Stress is one of the principal causes that affects quality of life in urban areas. The urban life often exposes individuals to stressors, originating from factors such as noise pollution, social and economic pressures. As a consequence, stress detection in urban settings has taken the attention of researchers.

Most of the research in the literature uses wearable devices to collect physiological signals from the users. This enables the development of various algorithms, going from range-based [2] to rule-based [1] ones.

Additionally, some researchers tried to localize the stress with GPS coordinates [1]. This approach enables to pinpoint a general point of stress in a city, and thus highlighting the most stressed areas. Urban planners can gain invaluable insights into the dynamics of urban stressors, allowing them to formulate targeted interventions and strategies aimed at alleviating stress.

Our project proposes an intuitive interface to visualize stress hotspots on a map, allowing the user to keep track of their individual stress and also the overall stress of their city. The stress is calculated using both range-based and rule-based algorithms presented in other research papers.

2 Architecture

The application is composed of two modules: the wearable and the handheld one. An overview of the architecture can be seen in Figure 1. The overall process is the following:

1. The wearable device collects the sensor's data and sends them to the device through a synchronization channel.
2. The handheld device sends the raw data to the database.
3. The python server gets the raw data from the database to analyze the stress.
4. Then it sends the derived data to the database.
5. The handheld device retrieves raw and derived data from the database to show them to the user.

Each of these steps will be discussed in detail in the following sections.

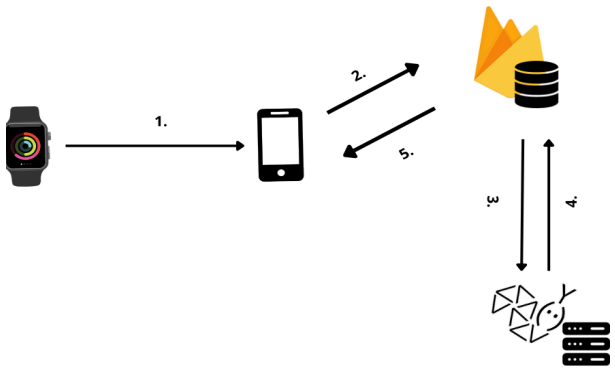

Figure 1: scheme of the application

2.1 Sensors sampling

The wearable app uses 3 sensors: EDA (ElectroDermal Activity), skin temperature and heart rate. The user can use the simple interface on the watch, which has a switch to start the sampling of the sensors.

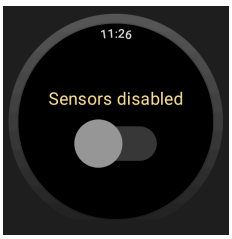

Figure 2: wearable app interface

The EDA signal can be divided into two parts: tonic and phasic components. The latter one results from sympathetic neuronal activity and thus is fundamental to detect stress. A filtering phase is fundamental to remove noise and divide the EDA signal in the tonic and phasic components. After a first filtering of the signal with a low-pass filter, the phasic part is extracted using a high-pass filter.

Saving all signal variations would result in a high battery consumption, and it would be useless for the purposes of the app. Therefore, signals are saved with a frequency of 1 Hz, also to facilitate the development of the stress detection algorithms. To see how data is saved and sent see section 2.2.

Each sample is associated with a location, using the *FusedLocationProvider* API. The location is taken with GPS, due to the need to know the precise location of the user. Despite the fact that we need to know the precise location, we decided to round the coordinates to the third decimal place. This was done to aggregate the stress data of the users in the same approximated location.

The location is updated dynamically, adapting to the speed of the user. That is, if the user goes faster, the app will update the location more frequently. The frequency of updating is calculated as the time to cross a *square* (i.e., the distance between two approximated locations) at the current speed.

In this manner, if the user stays still in the same location the app will reduce the power consumption, keeping at the same time precision when he moves.

2.2 Synchronization

To optimize energy consumption, we bundle 30 sensor samples into a single payload, resulting in a payload size of 1080 bytes (36 bytes per sample).

Each sample includes the following components:

- **Timestamp**: timestamp of the recorded data in milliseconds [8 Bytes]
- **EDA Sensor** Value: Electrodermal activity sensor value [4 Bytes]
- **Temperature** Sensor Value: The temperature sensor value in Celsius [4 Bytes]
- **Heart Rate Sensor** Value: The heart rate sensor value in beats per minute [4 Bytes]
- **Latitude**: latitude value of the position [8 Bytes]
- **Longitude**: latitude value of the position [8 Bytes]

The synchronization mechanism uses Google's Wearable Data Layer APIs to enable communication between the wearable device and mobile application. Specifically, it exploits Channel Client, which provides a flexible approach for data transmission and is useful for scenarios where there's the need to stream continuous data.

2.3 Firebase

In the development of the application, various services provided by Firebase Google were leveraged for efficient data management. Specifically, the following services were utilized:

1. **Authentication**: Firebase Authentication facilitated user registration and login within the application, ensuring secure access control.
2. **Firestore DocumentDB**: Firestore served as the document database for storing both raw data (RawData) directly collected from the watch and derived data (DerivedData), obtained by processing RawData within a dedicated Python server for each user account. Data was organized into collections, with each account having its own collection. Additionally, an additional *Map* collection was created to keep track of the stress level for each geographical position per hour, enabling real-time updates and visualization.

Figure 3: Map and account collection structures

3. **Real-Time Key-Value Firebase DB**: To handle the substantial volume of data and immediate operations required by the application and host server, the Realtime Database of Firebase was integrated. This key-value database facilitated efficient operations such as stress score calculation and continuous streaming of RawData from the database to the application. Two distinct buckets, *Map* and *account*, were utilized to organize data. The first one Temporarily stores the 30 samples cyclically sent from the smartwatch to the phone, optimizing query efficiency for retrieving the latest sampled data. The second one contains key information such as geographical coordinates and current stress levels of positions, crucial for data manipulation for HeatMap generation and other advanced functionalities.
4. **Communication between Application and Firebase:** The application was structured following the *Model-View-ViewModel and Service* (MVVMS) pattern. The Service module, subdivided into Service and Dao, facilitates communication with Firebase. Service handles

business logic and Firebase communication, while Dao manages data access operations.

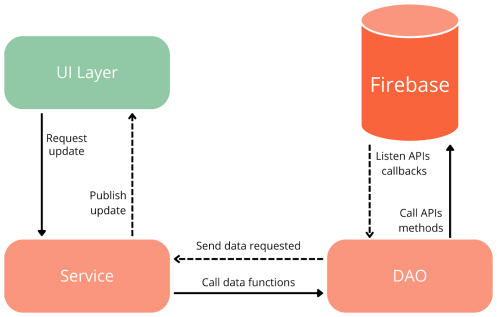

Figure 4: Firebase dialogue through services

2.4 Mobile app interface

The mobile interface is more complex than the wearable one, implementing many features, each one developed in a dedicated submodule.

Splash module. Simple module that implements the initial splash screen. It allows a faster application usage if the user has not logged out yet, exploiting the *auth* Firebase feature.
Access module. It implements screens and logic to manage login, registration and password recovery phases. An account is needed to access and use the application.
Home module. The home module implements all the functionalities used in the app when the user is logged in, including a simple setting screen. In particular:
Monitor. The monitor screen (Fig. 5) displays the main graphs associated with the user: it allows visualizing a report of the physiological data sensed and the stress levels computed, day by day. Since the UI should display a lot of data, we decided to aggregate them in intervals of 5 minutes, to avoid an excessive computational load of the UI. For activity data aggregation, we simply computed the mean of the values, while for stress data we calculated a weighted mean to give more importance to higher stress values.

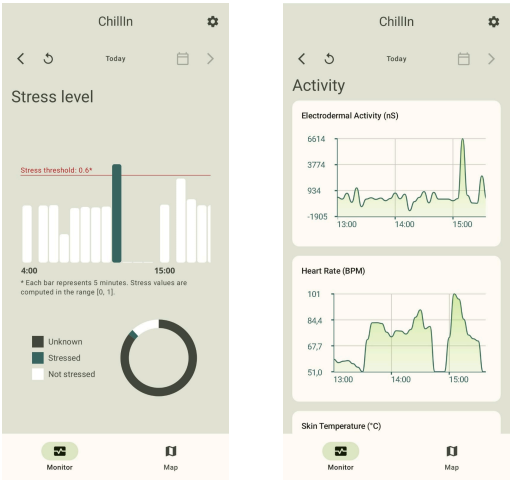

Figure 5: wearable app interface

Map. The map screen (Fig. 6) provides a heatmap displaying the stress areas in different hours and days that the user can set. In particular, it displays the stress points, for which it shows the maximum and minimum values: the value of a hotspot is the number of stress moments detected at that coordinates and hour of the day.

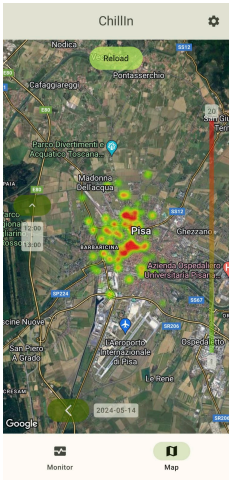

Figure 6: map interface

2.4.1 UI Implementation

The application has been developed using *Android Compose*, a modern toolkit that provides a declarative approach to generate UI interfaces.
Each screen is made up of composable functions that consume data provided by the *View Model*. This one is also responsible for executing actions and updating the *UI state* in response to the events of *UI elements*. The UI State maintains all the data that have to survive through recompositions (Fig. 7).
The *Data Layer* is implemented by services that can be accessed and exploited by the View Model: it allows retrieval of cloud data

through an interface, ensuring high levels of modularization and separating the *UI Layer* from the real implementation of the lower level.

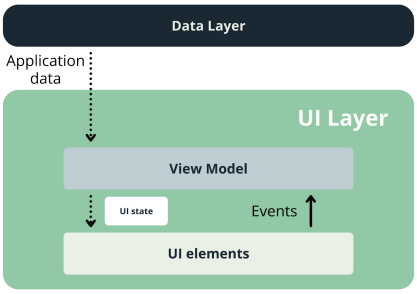

Figure 7: Interaction schema of the application elements

For the heatmap implementation, we exploited the *Google Maps SDK for Android* through an API, which provides a view that then is inflated into the Compose architecture.

3 Stress detection algorithms

The core logic of the application has been implemented in Python, defining two different algorithms to detect and classify stress perceived from the user. Namely, a range based and a rule based approach have been followed to classify the moments of stress.

3.1 Range Based Algorithm

The Range Based Algorithm [1] is employed to adaptively obtain a range of values in which the heart rate should fall over time to prevent user stress situations. This approach is based on a Bayesian method called Gibbs sampling. The algorithm iteratively estimates the mean and variance of the data, seeking to approximate the data distribution to a normal distribution. The algorithm's hyperparameters, such as the prior mean and variance, were estimated using a grid search algorithm on a dataset provided by the paper. The algorithm was adapted to work in a data streaming fashion and utilizes a sliding window to dynamically estimate the desired range. The estimation range, called “binterval”, is updated whenever new data arrives.

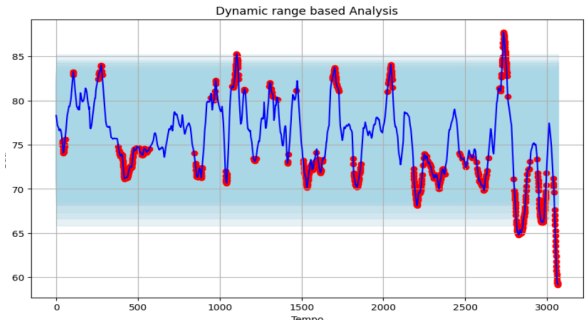

Figure 8: How the range algorithm works with the paper test dataset

3.2 Rule Based Algorithm

The Rule Based Algorithm [2] is another technique used to compute the stress score of a user based on raw data from the Real Time Database. This algorithm considers the data from the EdaSensor and SkinTemperature of the patient along with the timestamp. Using five specific rules, the stress score is calculated for each instance of the patient, considering the values of the next 30 samples. The first four rules are applied to each sample, and the score is calculated as the weighted average of these rules. The fifth rule imposes a maximum limit of one stress moment every 10 seconds. The algorithm was evaluated by comparing the results with stress labels present in the dataset provided by the paper, achieving an accuracy of 70% and a sensitivity of 85%. The obtained results indicate that the algorithm is particularly sensitive in detecting stress moments. Finally, the estimation range and the stress score are sent to Firestore for monitoring and analysis purposes.

3.3 Heatmap Generation

The process of gathering data to construct a collection Map for identifying hotspots entails the utilization of both *Firestore* and the *Realtime* Database within the Firebase platform. This strategic decision was made to mitigate potential resource constraints inherent in our subscription plan, which could have been exacerbated by frequent accesses to Firestore. Operationally, the Map module is invoked every 30 seconds by the *main_routine.py* script. Initially, data is relayed to the Realtime Database to ensure timely updates. Subsequently, on an hourly basis, the *map_routine.py* script transmits this accumulated data from the Realtime Database to Firestore, merging it with existing datasets. At each 30-second interval, the Map instance receives 30 samples of *DerivedData*, each containing attributes such as *stress_score, timestamp, latitude, and longitude*. Notably, a threshold of **0.6** on the **stress_score** scale (ranging from 0 to 1) is empirically established to identify **Moments of Stress (MOS)**. Upon detecting a MOS, the script extracts the date (formatted as YYYY-MM-DD) and hour (ranging from 0 to 23) from the timestamp, which is represented in milliseconds. Consequently, the MOS event is appended to the transient collection within the Realtime Database, indexed by the corresponding latitude, longitude, date, and hour coordinates.

4 Testing

We had a short testing phase at the end of the app development, to evaluate in broad terms the overall power consumption of the wearable application and its accuracy in stress detection.
For the tests we used:

- *Redmi Note 8 Pro* smartphone (Android 11), in which we installed the mobile application module;

- *Google Pixel Watch 2* wearable (Wear OS 4.0), in which we installed the wearable module to sense data.

The tests consisted of using the application while traveling in the urban contexts of Pisa and Livorno, both by car and on foot. The user then described his trip, underlining the stress levels perceived during the activities done.
In addition, we tracked the wearable battery percentage between the start and the end of the tasks.

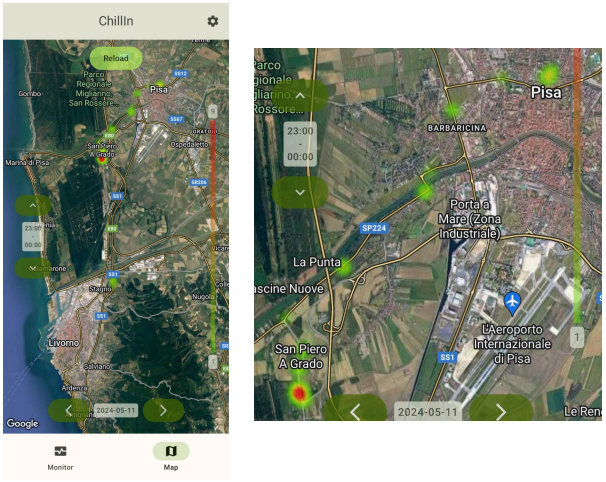

Figure 9: Heatmap results during a task in the test phase

Power consumption. Primarily, tests allowed the detection and fixing of some bugs that caused a rapid decrease in the battery level in the wearable device. In this way, we observed that, during a task that started at 12:45 and ended at 15:30, the battery power of the wearable decreased from 92% to 76%.
Stress detection. Furthermore, tests showed the detection of some stress hotspots during both types of travel, which overlapped the stress moments declared by the user at the end of the task (Fig. 9).

5 Conclusion

This project represents a step forward in addressing the pressing issue of stress management in modern society: through the utilization of a wearable device, improved by AI algorithms, the application allows users to monitor their stress levels in real time, providing useful insights into the impact of the urban environment on mental well-being.
Indeed, the real improvement offered by the application is the ability to correlate stress data with geographical location. This provides not only a better understanding of stress triggers for the user, but it also diffuses important information on stress hotspots within cities.
Overall, ChillIn has the potential to be useful in several fields, from public health and personal wellness, to urban planning and community development, as it could also be included in employer wellness programs.

Further improvements. Some improvements can be applied to reach better application performance and overall quality:

- *Power Consumption.* The introduction of a cache in data sent from the smartphone to the database would optimize the power consumption of the handheld device.
- *Graceful degradation.* The application should be tolerant to sensor availability, limitations to permissions given by the user and internet connectivity issues. To increase the compatibility with wearable devices, a solution could be the application of the range-based algorithm instead of the rule-based one, in the case that the heart rate sensor is the only sensor available.
- *New functionalities.* Keeping aggregated data about coldspots too, providing a better evaluation of stress points. Introducing a user-dedicated hotspots map which displays only the user hotspots.
- *GDPR Compliant.* Manage the sensitive data, such as location, according to privacy policy.
- *Malicious user tolerance.* Introducing a system to detect and exclude malicious users and improper data generation, to avoid heatmap generation issues.
- *Algorithm accuracy.* Some further works could aim to improve the stress detection accuracy by testing new algorithms using different datasets.
- *Interoperability.* The application could be developed to support a larger range of operating systems. Currently, the data layer implemented limits the availability to Wear OS. In addition, a Swift version of the mobile application could extend the support also to iOS devices.

REFERENCES

[1] Kyriakou, K.; Resch, B.; Sagl, G.; Petutschnig, A.; Werner, C.; Niederseer, D.; Liedlgruber, M.; Wilhelm, F.H.; Osborne, T.; Pykett, J. Detecting Moments of Stress from Measurements of Wearable Physiological Sensors. *Sensors* **2019**, *19*, 3805. https://doi.org/10.3390/s1917380

[2] Iqbal, T.; Simpkin, A.J.; Roshan, D.; Glynn, N.; Killilea, J.; Walsh, J.; Molloy, G.; Ganly, S.; Ryman, H.; Coen, E.; et al. Stress Monitoring Using Wearable Sensors: A Pilot Study and Stress-Predict Dataset. *Sensors* **2022**, *22*, 8135. https://doi.org/10.3390/s22218135