UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Polygraph Test

Student :
Rafail MARIS
Stefania-Cristina MOZACU

Group : 30433, 30434

Teacher :
Radu DANESCU

January 19, 2026

Contents

1 Introduction

A polygraph system is typically designed to monitor physiological signals that may change under stress or emotional arousal. While real-world lie detection is a complex and controversial problem, measuring biometric reactions such as heart activity, skin conductance, and voice intensity can provide useful indicators of stress in controlled scenarios.

This project implements a real-time polygraph-inspired monitoring system using an **Arduino Uno** and an **ESP32** module. The device collects multiple signals, including **Electrocardiogram (EKG)**, **Galvanic Skin Response (GSR)**, and **sound level** from a microphone sensor. In addition, the system includes an **RFID authentication module** that controls access to live biometric measurements, as well as a **joystick input** used to simulate stress events for testing purposes.

The architecture is organized as an end-to-end pipeline. The Arduino is responsible for sensor acquisition and user authentication, then it transmits the collected data to a laptop through a Serial connection. A Python application running on the laptop performs an initial calibration stage to compute baseline values, then continuously evaluates incoming measurements and generates a deception/stress probability score. The processed data is streamed to a web-based dashboard in real time using Socket.IO, where the user can visualize the signals and system status. Finally, the computed score is forwarded to the ESP32, which transmits it to a mobile phone using **Bluetooth Low Energy (BLE)** notifications.

The main objective of this project is to demonstrate the design of a distributed embedded system that integrates sensor acquisition, real-time communication, online calibration, and interactive visualization. The final result is a functional prototype that can operate both in **simulation mode** (unauthenticated) and **live mode** (authenticated), enabling reliable demonstrations even when full hardware signals are not available.

2 Project Goals and Requirements

2.1 Functional Requirements

The system must satisfy the following functional requirements:

- **Multi-sensor acquisition:** The Arduino Uno must continuously read and process the following signals:
  - EKG signal (real or simulated)

  - Galvanic Skin Response (GSR)
  - Sound level from a microphone sensor

- **User authentication:** The system must authenticate the user using an RFID card reader. If the scanned card UID matches an authorized UID, the user is considered authenticated and the system enters live monitoring.
- **Operating modes:** The system must support two operating modes:
  - **Simulation mode (unauthenticated):** the EKG signal is generated internally using a predefined waveform model.
  - **Live mode (authenticated):** the EKG signal is acquired from the physical EKG sensor input.
- **Calibration stage:** After startup, the software must run an initial calibration period (baseline acquisition) for a fixed duration. During calibration, baseline statistics are computed for each sensor signal.
- **Stress / deception scoring:** After calibration, the system must compute a real-time probability score that estimates stress or possible deception, based on deviations from baseline values.
- **Real-time visualization:** The collected and processed data must be displayed on a web dashboard, including:
  - Live plots for EKG, GSR, and sound level
  - Authentication status indicator
  - Stress detection indicator
  - Deception probability percentage
- **Bluetooth score transmission:** The computed deception probability score must be periodically transmitted from the laptop to the ESP32 through a Serial connection, and then forwarded to a mobile device via BLE.

2.2 Non-Functional Requirements

In addition to the functional behavior, the system must satisfy the following non-functional requirements:

- **Real-time performance:** Sensor values and system status must be updated with low latency in order to provide a real-time monitoring experience on the dashboard.

- **Reliability and robustness:** The communication pipeline (Arduino → Python → Web Dashboard and Python → ESP32 → Phone) should remain stable during continuous operation. Temporary disconnections or invalid serial messages should not cause the entire application to crash.
- **Data consistency:** The transmitted sensor data must preserve the correct order and meaning of each measurement field (EKG, GSR, sound, authentication state, stress state), ensuring correct visualization and scoring.
- **Modularity:** The project should be structured in separate modules (Arduino firmware, Python server, ESP32 firmware, Web dashboard) to allow independent debugging, development, and future upgrades.
- **Usability:** The dashboard interface must be easy to understand, providing clear indicators and readable plots for non-technical users.
- **Portability:** The system should be able to run on a standard laptop without specialized hardware, using common libraries and frameworks (Flask, Socket.IO, Chart.js), and allowing quick deployment for demonstrations.
- **Scalability for future improvements:** The design should allow extensions such as improved filtering, additional sensors, data logging, or advanced detection algorithms without major architectural changes.

2.3 Data Flow

The system follows a multi-stage data pipeline that connects the embedded acquisition hardware to the user interfaces. The complete end-to-end data flow is as follows:

1. **Sensor acquisition on Arduino Uno:** The Arduino continuously samples the connected sensors:
    - EKG signal (real input if authenticated, simulated waveform otherwise)
    - GSR sensor value (smoothed using a moving average)
    - Sound level (computed as signal energy over a short time window)

    In parallel, the Arduino monitors the RFID reader for user authentication and the joystick for stress simulation triggers.
2. **Serial transmission to laptop:** After each sampling cycle, the Arduino sends a formatted Serial message containing all relevant values:

- EKG value
- GSR average value
- Sound level
- Authentication state
- Stress state

3. **Parsing and processing in Python:** A Python application running on the laptop reads the Serial stream, parses each incoming line, and converts it into a structured data object. During the first seconds of execution, the software performs a calibration stage to compute baseline values for EKG, GSR, and sound.

4. **Stress / deception probability estimation:** After calibration is completed, the Python application compares incoming sensor values against baseline thresholds and computes a real-time deception probability score based on the frequency of threshold exceedance.

5. **Real-time dashboard streaming:** The processed data is transmitted to the web dashboard using Socket.IO. The dashboard receives updates through the `sensor_data` event and refreshes the charts and status indicators.

6. **Bluetooth score forwarding via ESP32:** In parallel, the Python application periodically sends the deception probability score to an ESP32 module using a second Serial connection. The ESP32 forwards the received score to a mobile phone using BLE notifications.

3 Hardware Design

3.1 Components Used

The hardware implementation is based on a modular design, where each component provides a specific sensing or communication function. The main components used in this project are listed below:

- **Arduino Uno:** Main microcontroller responsible for sensor acquisition, signal preprocessing, and user authentication logic.
- **ESP32 module:** Secondary microcontroller used as a Bluetooth Low Energy (BLE) communication bridge between the laptop and a mobile phone.

- **MFRC522 RFID Reader:** Used to authenticate the user by scanning RFID cards and comparing the UID to an authorized identifier.
- **EKG Sensor Module:** Provides an analog Electrocardiogram signal that is sampled by the Arduino in live mode.
- **GSR Sensor (Galvanic Skin Response):** Measures skin conductance variations, which may change with stress levels. The output is read as an analog value.
- **Microphone / Sound Sensor Module:** Measures sound intensity. The Arduino estimates the sound level by analyzing short-term signal variation.
- **Joystick Module:** Used as an additional user input device and as a manual trigger for stress simulation during testing.
- **Laptop / PC:** Runs the Python server, performs calibration and scoring, and hosts the real-time dashboard.
- **Mobile Phone:** Receives deception probability updates from the ESP32 via BLE notifications.

3.2 Pin Mapping

The Arduino Uno is connected to multiple sensors and modules using both digital and analog pins. Table 1 summarizes the main pin assignments used in this project.

Table 1: Arduino Uno pin mapping

Component	Signal / Function	Arduino Pin
MFRC522 RFID Reader	SS (SDA)	D10
MFRC522 RFID Reader	RST	D9
Status LED (Authorized)	Digital output	D6
Status LED (Denied)	Digital output	D5
GSR Sensor	Analog input	A0
Microphone / Sound Sensor	Analog input	A1
EKG Sensor	Analog input	A2
Joystick	Analog input	A3
I2C (Slave mode)	SDA / SCL	A4 / A5

In addition, the RFID module communicates with the Arduino through the SPI interface, which uses the standard Arduino Uno SPI pins (D11 for MOSI, D12 for MISO, and D13 for SCK).

4 Software Structure

The software implementation combines embedded firmware development with a Python-based backend and a web-based visualization interface. The following libraries and tools were used to implement the complete system:

4.0.1 Arduino Uno Firmware

The Arduino code was developed using the Arduino IDE and relies on the following libraries:

- `SPI.h`: provides SPI communication support required by the MFRC522 RFID module.
- `MFRC522.h`: handles RFID card detection, UID reading, and authentication logic.
- `Wire.h`: provides I2C communication support (used for optional binary data transmission as an I2C slave).

4.0.2 Python Backend

The laptop-side application was implemented in Python and uses the following libraries:

- `pyserial`: enables Serial communication with the Arduino Uno and the ESP32 module.
- `Flask`: lightweight web framework used to serve the dashboard interface.
- `Flask-SocketIO`: enables real-time bidirectional communication between the Python server and the web dashboard.
- `threading`: used to run Serial reading in a background thread while keeping the web server responsive.

4.0.3 Web Dashboard

The dashboard is implemented as a single-page web interface and uses:

- **Socket.IO (JavaScript client)**: receives live sensor updates from the Python server.
- **Chart.js**: renders real-time line charts for EKG, GSR, and sound level visualization.
- **HTML/CSS**: provides the structure and styling of the interface.

4.0.4 ESP32 Firmware

The ESP32 firmware is responsible for BLE communication and uses the following libraries:

- `BLEDevice.h`, `BLEServer.h`, `BLEUtils.h`: core BLE functionality for creating a BLE server and advertising services.
- `BLE2902.h`: enables notification support for BLE characteristics.

5 Communication Protocols

The system uses multiple communication channels in order to connect the embedded hardware to the user interfaces. Each communication link has a dedicated protocol designed for simplicity and real-time performance.

5.1 Arduino Uno to Python (Serial)

The Arduino Uno transmits sensor readings and system state to the laptop using a USB Serial connection at 9600 baud. Data is sent as a human-readable text line, where each field is separated by the `|` character and each key-value pair uses the `:` delimiter.

An example Serial message is shown below:

```
EKG: 120 | GSR: 450 | SOUND: 33 | AUTH: 1 | STRESS: 0
```

The transmitted fields are:

- **EKG**: 8-bit value representing the EKG sample (0–255)
- **GSR**: averaged GSR sensor value (0–1023 range depending on ADC)
- **SOUND**: sound activity level computed from microphone input
- **AUTH**: authentication state (0 = denied, 1 = authorized)
- **STRESS**: stress state flag (0 = inactive, 1 = active)

5.2 Python to Web Dashboard (Socket.IO)

After parsing and processing the Serial data, the Python backend broadcasts updates to the dashboard using Socket.IO. The server emits a `sensor_data` event containing a JSON payload with the latest measurements and computed score.

The main fields included in the payload are:

- `ekg`, `gsr`, `sound`
- `auth`, `stress`
- `maybe_lying`: deception probability score (percentage)

This mechanism allows the dashboard to refresh plots and status indicators in real time without page reloads.

5.3 Python to ESP32 (Serial)

To support mobile monitoring, the Python backend periodically sends the computed deception probability score to the ESP32 module using a second Serial connection. The message format consists of a single integer value (0–100) followed by a newline character:

```
85
```

5.4 ESP32 to Mobile Phone (Bluetooth Low Energy)

The ESP32 operates as a BLE server and advertises a custom service. A BLE characteristic is configured with notification support, allowing the ESP32 to push updates to the phone whenever a new score is received from Python.

The BLE configuration uses the following identifiers:

- **Service UUID:** `12345678-1234-1234-1234-1234567890ab`
- **Characteristic UUID:** `abcdefab-1234-1234-1234-abcdefabcdef`

When a phone connects and subscribes to notifications, the ESP32 sends the received score as a string value through BLE notify events.

6 System Workflow

The system operates as a continuous real-time monitoring pipeline, starting from user authentication and calibration, and ending with live visualization and Bluetooth transmission. The complete workflow is described below.

6.1 Startup Phase

When the system is powered on, the Arduino Uno initializes all connected modules. At the same time, the Python backend is started on the laptop and waits for incoming Serial data from the Arduino.

6.2 Authentication and Operating Mode Selection

The Arduino continuously checks for RFID card presence. When a card is scanned, the UID is compared against a predefined authorized UID:

- If the UID is authorized, the system enables **live mode**, where real EKG sensor data is sampled.
- If the UID is not authorized, the system remains in **simulation mode**, where the EKG signal is generated internally.

6.3 Calibration Stage

After the Python backend begins receiving data, it enters a calibration stage with a fixed duration (15 seconds). During this period, baseline statistics are collected for each monitored signal:

- EKG
- GSR
- Sound level

At the end of the calibration interval, the baseline average and standard deviation values are computed and stored for later comparisons.

6.4 Live Monitoring and Scoring

Once calibration is completed, the system continuously processes incoming samples:

- Arduino sends new sensor values through Serial at a high update rate.
- Python parses each message and evaluates whether sensor values exceed baseline thresholds.
- A deception probability score is computed as a heuristic percentage based on the frequency of threshold exceedance.

6.5 Real-Time Visualization

The Python backend broadcasts the latest sensor values and computed score to the web dashboard using Socket.IO. The dashboard updates:

- Real-time plots for EKG, GSR, and sound level

- Authentication status indicator
- Stress state indicator
- Deception probability percentage

6.6 Bluetooth Transmission to Mobile Phone

In parallel with the web visualization, the Python backend periodically sends the deception probability score to the ESP32 through Serial communication. When a phone is connected via BLE, the ESP32 forwards the received score using notification messages, enabling real-time monitoring on a mobile device.

7 Calibration and Detection Algorithm

7.1 Calibration Stage (Baseline Computation)

In order to detect significant variations in biometric signals, the system begins with a calibration stage. The goal of calibration is to record a baseline profile for the monitored user in a neutral state, before any stress or abnormal activity occurs.

After the Python backend starts receiving Serial data from the Arduino, it collects samples for a fixed duration of 15 seconds. During this time interval, the following signals are stored:

- **EKG** (heart electrical activity amplitude)
- **GSR** (skin conductance level)
- **Sound level** (microphone activity energy)

For each signal, the calibration phase computes two statistical parameters:

- The baseline mean value (μ)
- The baseline standard deviation (σ)

The mean is calculated as:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

and the standard deviation is calculated as:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

These baseline parameters are stored and used as reference values during the live monitoring stage. Once calibration is completed, the system transitions automatically into continuous detection and scoring.

8 Web Dashboard

The web dashboard represents the main user interface of the system and provides a real-time visualization of both raw sensor signals and processed status indicators. It is implemented as a single-page web application using HTML, CSS, and JavaScript, and it is served directly by the Python backend through Flask.

8.1 Real-Time Signal Visualization

The dashboard displays three live line charts, each corresponding to one of the monitored biometric signals:

- **Electrocardiogram (EKG):** shows the current heart activity waveform amplitude.
- **Galvanic Skin Response (GSR):** shows variations in skin conductance over time.
- **Sound Level:** shows short-term microphone activity, representing voice intensity and ambient sound changes.

All charts are implemented using the **Chart.js** library. To ensure smooth performance and prevent memory overflow, the dashboard keeps only a fixed number of recent points (100 samples) for each plot, creating a rolling window effect.

8.2 System Status Indicators

In addition to signal plots, the dashboard includes a dedicated status section that displays key system states:

- **Authentication Status:** shows whether the user is authorized (`YES`) or not (`NO`).

- **Stress Detected:** shows whether the stress flag is active.
- **Deception Probability:** shows the computed deception/stress probability score as a percentage.

These indicators allow the user to interpret the sensor signals more easily and quickly understand the current system state.

8.3 Live Communication Using Socket.IO

The dashboard receives live data through a Socket.IO connection to the Python backend. The server broadcasts updates using the `sensor_data` event, and each received message updates both the charts and the status indicators without reloading the page.

8.4 Demo Mode Fallback

For demonstration purposes, the dashboard includes a fallback mode. If the Socket.IO server is not available, the page automatically switches to a simulated data generator, producing artificial EKG, GSR, and sound signals. This feature ensures that the interface can still be presented and tested even when the backend is offline.

9 Live Demo

To illustrate the real-time operation of the polygraph monitoring system, a live demonstration video has been recorded and made available online. This video showcases the full workflow of the system, including RFID authentication, sensor acquisition, calibration, real-time dashboard updates, and scoring output.

You can view the live demo here:
`https://drive.google.com/file/d/1y6M6QZSXl0O5DD4yhZ9pXTN-Fntksz2a/view?usp=sharing`

10 Setup and Run Instructions

10.1 Hardware Setup

1. Connect sensors and RFID module to the Arduino Uno accordingly.
2. Connect the Arduino to the laptop via USB.
3. Connect the ESP32 to the laptop via USB.

10.2 Software Setup

1. Install Python dependencies:

```
pip install flask flask-socketio pyserial
```

2. Update the COM ports in `pythonWeb.py`:

```
ARDUINO_PORT = 'COM10'
ESP32_PORT   = 'COM16'
```

3. Run the Python server:

```
python pythonWeb.py
```

4. Open the dashboard in a browser:

```
http://localhost:5000
```