Internet of Things, a.a. 2023-2024
Projects' proposals

Prof. Luciano Bononi, Prof. Marco Di Felice, Dr. Ivan Dimitry Zyrianoff
{luciano.bononi, marco.difelice3, ivandimitry.ribeiro}@unibo.it

May 14, 2024

General rules

- **HOW** The project can be developed **individually or in groups of maximum 2 units**.
- **HOW** The project can implement one of the two provided drafts, or a new draft proposed by the student/group. In the first case (draft below), the system specifications can be customized/extended, but the overall complexity should not be significantly reduced. In the second case (draft from students), the proposal must be approved by either Prof. Di Felice or Prof. Bononi. The originality of the topic will not be evaluated. However, every project must implement the IoT data pipeline presented during the IoT course, including data generation and processing on the edge device (e.g., ESP32), data acquisition (e.g., via HTTP/MQTT/COAP protocols), data management (e.g., using a time-series database), data visualization (e.g., via a `GRAFANA` dashboard), and data processing and analytics.
- **HOW** The projects must be submitted exclusively via the Virtuale platform (Section: Project Submission). Submissions sent via emails or Teams will not be considered.
- **WHEN** Submit the project when ready. However, we will review the submissions on Virtuale at (the midnight of) these dates: 15 June, 15 July, 1 Sept, 15 Sept, 15 Oct, 15 Nov, 15 Dec, 15 Jan 2024, 15 Feb 2024. All submissions found on Virtuale will be considered for project discussion, which will be scheduled within the next two weeks. Instructions regarding the date and location of the discussion will be sent via email.
- **WHAT** Submit a zip file containing all the source code of the project and a technical report describing the project design and implementation. Sections of the report: Introduction, Project's Architecture, Project's Implementation, Results. The report must be written in English and use the IEEE conference template, which can be found at: `https://www.ieee.org/conferences/publishing/templates.html`

- **WHAT** In addition to the technical report, the student must prepare a presentation (with slides) of the project for the oral discussion. The slides must be submitted on Virtuale (Section: Project Presentation), at least one hour before the discussion.
- **WHAT** The oral discussion will involve presenting the project and demonstrating it using the students' devices. Questions related to the theoretical part of the course may be asked, either on topics related or unrelated to the project.
- **NOTE** All members of the group must be present during the discussion and should have a complete understanding of 100% of the project.

1 Proposal (IoT Alarm System with Bed Presence Detection)

Objective: Design an IoT-based alarm system that incorporates bed presence detection to ensure the alarm is only triggered when the user is in bed and automatically set off when the user gets up.

Motivation: Traditional alarm clocks can be disruptive. This IoT alarm system can provide a more ubiquitous and non-intrusive waking experience by incorporating bed presence detection.

Specifications: The IoT system must be designed as follows:

- *Hardware*: ESP32 + pressure sensor (pressure mat sensor) + speaker
- *Data acquisition*: Use a microcontroller (such as Arduino or ESP32) to acquire and transmit the sensor data to a data proxy via HTTP or CoAP protocols. The sketch must support the MQTT protocol to receive configuration updates from the proxy. The following commands must be supported via the MQTT protocol:
  1. `sampling_rate`: interval between consecutive sensor readings.
  2. `trigger_alarm`: triggers a pre-loaded sound effect/music to be played (in-loop) through the connected speaker until the user gets up or the system receives a `stop_alarm` command.
  3. `stop_alarm`: If an alarm is on, turn it off.
- *Data Proxy*: a Python application executed outside the micro-controller (e.g., in your laptop). The script receives the sensor data and sends them to an Influxdb instance. It also integrates with a simple front-end application (or terminal) that allows the user to set the time of the next alarm. The data proxy should communicate with the microcontroller to trigger the alarm.

- *Influxdb*: time-series database collecting sensor values.
- *Data analysis module*: a Python application that, based on the sensor data collected, can determine the number of hours the user slept each day.
- *Grafana*: develop a dashboard to display the actual values available in the database and the average sleeping time.
- *Evaluation*: investigate the performance of the IoT system in terms of:
  1. Mean Latency of the data acquisition process (network latency to send data to the proxy).
  2. Accuracy to detect if a person is in bed or not.

  Bonus: Optional features that will be considered for the evaluation:
  - Front-end (+1pt). Developed a front-end application and enhanced the data proxy, allowing the users to manage the alarms better (e.g., create, update and delete operations) and allowing them to configure several alarms and their periodicity (each day, each Monday, each weekday, etc.).
  - Telegram bot (+1pt). The user can create, update and remove alarms through a Telegram bot. Also, the user can stop the alarm using the bot.
  - Weather-based alarm (+1pt). The system integrates with a weather API and, based on the current weather conditions, chooses sound effect/music to play as the alarm that suits the mood.

2 Proposal 2 (Daily light tracking of plants)

Objective: Develop an IoT system for tracking daily light exposure of plants using luminosity sensors. One sensor should be placed into the plant pot, and others (at least 1 more) should be placed in the other possible plant positions. The system will analyze the light data of each sensor to determine the optimal placement of plant pots based on the specific light requirements of each plant.

Motivation: Proper light exposure is crucial for the health of plants. However, it is difficult to define the best position of a given plant, resulting in a try-and-error scenario (and many dead plants in the process). This project aims to improve plant growth and vitality by monitoring light conditions.

Specifications: The data pipeline must be deployed as follows:

- *Hardware*: ESP32 + X ($X \geq 2$) LDR sensors (luminosity sensors).

- *Data acquisition*: Use a microcontroller (such as Arduino or ESP32) to acquire and transmit the sensor data periodically through CoAP or HTTP. Each message should have a specific sensor ID and location (pre-set in a configuration phase through the Data Proxy). In addition, the sketch must be able to receive configuration updates from the proxy through the MQTT protocol. Those messages are related to changes in the placement of pots or the sensor configuration. Each sensor has its own MQTT topic. The commands that must be implemented are:

  1. `sampling_rate`: interval between consecutive sensor readings.
  2. `change_position`: alters the position from position A to position B (as defined in the message).

- *Data Proxy*: a Python application executed outside the micro-controller (e.g., in your laptop). The Python application receives the sensor data and sends it to an Influxdb instance. Also, it enables configuration changes via the receiving protocol. Further, the user should be able to set the plants it has (type, amount of solar light required per day, and sensor associated with it) and the position in which sensors are deployed. Each position has an ID, a name, a description and a sensor associated with it.

- *Influxdb*: time-series database collecting sensor values.

- *Data analytics module*: a Python script processing each position's solar light and predicting the sun amount for the next N hours. The prediction should also be sent to Influxdb. The Data Analysis module should also be able to determine the optimum position of a plant – e.g., move the `id:01:basil` to from `position:01:balcony` to `position:03:bedroom-windown` because it will have one h more of solar light daily.

- *Grafana*: develop a dashboard to display the collected time series. It should display a graph for each position and each plant. Each graph should contain two lines: predicted and real-data.

- *Evaluation*: investigate the performance of the IoT system in terms of:

  1. Mean Square Error of the predicted value.
  2. Mean Latency of the data acquisition process (network latency to send data to the proxy).

Bonus: Optional features that will be considered for the evaluation:

- Front-end (+1pt).Developed a front-end application that makes it possible to set and visualize the blueprint of a house and to set, delete and modify the position of the plants and sensors through this app.

- Telegram bot (+2pt). Develop a telegram bot that notifies a user to change the plant position. The user can also update the position of plants using the bot.

3 Proposal (HVAC waste detection based on rapid temperature changes)

Develop an IoT system for detecting HVAC waste based on rapid temperature changes near air exits (e.g., windows). The system will analyze rapid changes in the temperature data to identify instances where energy may be wasted due to unnecessary heating or cooling – i.e., heating is on and a window is opened.

Motivation: HVAC systems significantly contribute to household energy consumption. Detecting instances of HVAC waste can help users optimize energy usage, reduce utility bills, and contribute to environmental sustainability.

Specifications: The data pipeline must be deployed as follows:

- *Hardware*: ESP32 + 1 temperature sensor placed near air exits + 1 temperature sensor placed outside + 1 LED.
- *Data acquisition*: Use the microcontroller to acquire temperature data periodically. Rapid temperature changes indicative of airflow will trigger an event. The system should get the sensors' data and transmit it to the data proxy using CoAP or HTTP protocol. The board should also support the following commands via the MQTT protocol:
  1. `sampling_rate`: interval between consecutive sensor readings.
  2. `start/stop`: start/stop the sensor(s) reading.
- *Data Proxy*: a Python application executed outside the micro-controller (e.g., in your laptop or on a Raspberry Pi). The Python application receives the temperature data and sends it to an InfluxDB instance for storage. It also allows setting the sampling rate of each sensor.
- *InfluxDB*: time-series database collecting temperature data.
- *Data analytics module*: a Python script processing temperature data to detect rapid changes indicative of HVAC waste. The outside temperature sensor will help differentiate between internal and external temperature influences. **In addition, the system must be able to forecast the next indoor and outdoor temperature values**.
- *Alarm System*: An alarm system will be triggered when rapid temperature changes are detected, indicating potential HVAC waste. The alarm events must be stored in Influxdb and must be displayed by turning on the LED connected to the micro-controller.
- *Grafana*: the dashboard should have one graph for each sensor and one graph that is the accumulative count of alarm events. The dashboard must also show the forecast values for the indoor and outdoor temperatures.

- *Evaluation*: Investigate the performance of the IoT system in terms of:
  1. Accuracy of the temperature forecast system.
  2. Mean Latency of the data acquisition process (network latency to send data to the proxy).

Bonus:

- Integration with HVAC System (+2pt): Integrate the system with the HVAC system via API to detect if the HVAC system is actively heating or cooling before triggering an alarm for potential waste. **Alternatively, users can manually indicate the HVAC status through a Telegram bot or a mobile app**.
- The system integrates with a weather API (+1pt) to monitor the outside temperature (data source in addition to the sensor).

4 Special (Research-oriented) Projects

Details must be discussed with Prof. Marco Di Felice or Prof. Luciano Bononi. Each proposal is for a single, highly-motivated student/group. The topics refer to IoT research areas:

- Micro-controller code offloading/virtualization with WebAssembly.
- TinyML (ML techniques for micro-controllers): Automatic re-training (on cloud) and dynamic model download and loading on micro-controller.
- Pro-active edge caching via federated learning techniques.
- ChatGPT for IoT; code generation for the automatic deployment of IoT mashup applications.
- Semi-automatic generation of Digital Twins of IoT appliances.
- IoT global market based on distributed ledger (blockchain) and distributed Oracle technologies.