UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Master's Degree in Artificial Intelligence and Data Engineering

LEATHERSENSE

Leather Environment Analysis and Temperature Handling Engine for Regulation and Sensor Enhancement

Group:

Emanuele Respino

Tommaso Falaschi

ACADEMIC YEAR 2023/2024

Contents

Chapter 1

Introduction

In the leather tanning industry, maintaining optimal environmental conditions and safety standards is paramount to ensuring product quality and worker safety. The process of leather tanning involves various chemical treatments and physical processes, which require precise control over several parameters. **LeatherSense** focuses on developing an IoT solution designed to monitor and regulate critical environmental factors during a specific tanning process, the **pickling** phase.

1.1   Pickling process

The pickling phase is a crucial step in the processing of vegetable-tanned leather, occurring within the sequence of *wet operations*. During this phase, the leather is prepared for the actual tanning process through a chemical treatment that temporarily stabilizes its structure.

Leather undergoes pickling after the steps of soaking, depilation, liming, deliming, and fleshing: it is immersed in an acidic solution, often containing vegetable tannins, to lower the pH and facilitate the penetration of the subsequent tanning agents. This solution prepares the leather for the tanning process by creating an ideal chemical environment for the reaction with the tannins and increasing leather preserving without deteriorating, stabilizing its structure and making the production process more flexible[1].

To ensure optimal results and high leather quality results, during pickling process is essential to continuously monitor and regulate water characteristics[2], especially:

- **pH Levels**: The pH level of the water is a critical factor in the pickling process. Maintaining the correct pH ensures that the leather absorbs the chemicals evenly, leading to a consistent and high-quality product. Deviations from the optimal pH range can result in defective leather and increased waste.

- **Salinity**: The salinity, measured in degrees Baumé, indicates the concentration of salts in the water. Accurate salinity levels are essential for the chemical

reactions required during tanning. Incorrect salinity can affect the leather's texture and durability, compromising the final product's quality.

- **Temperature**: Temperature control is vital for the stability of chemical processes involved in tanning. Excessive heat can accelerate reactions, potentially damaging the leather.

In addition to this, hazardous gases can be produced as results of chemical reactions during the tanning process such as **Hydrogen Sulfide** (H_2S). Monitoring H_2S levels is crucial for ensuring a safe working environment. Exposure to H_2S can pose serious health risks to workers, making it imperative to detect and mitigate its presence promptly.

Figure 1.1: Tanning drums used during pickling phase.

1.2 LeatherSense System

In this project, the developed system utilizes sensors to continuously measure the key **water characteristics** previously discussed, as well as the detection **hydrogen sulfide**. The data collected by the sensors are transmitted via an MQTT broker to a central database, where a Remote Control Application can analyzed and use them to trigger actuators which adjusts environmental conditions accordingly to user settings.

In this way, by implementing an IoT-based monitoring system, this project aims to **automatize** and **optimize** production processes, improving **worker safety**, ensuring regulatory compliance and maintaining high standards of product quality.
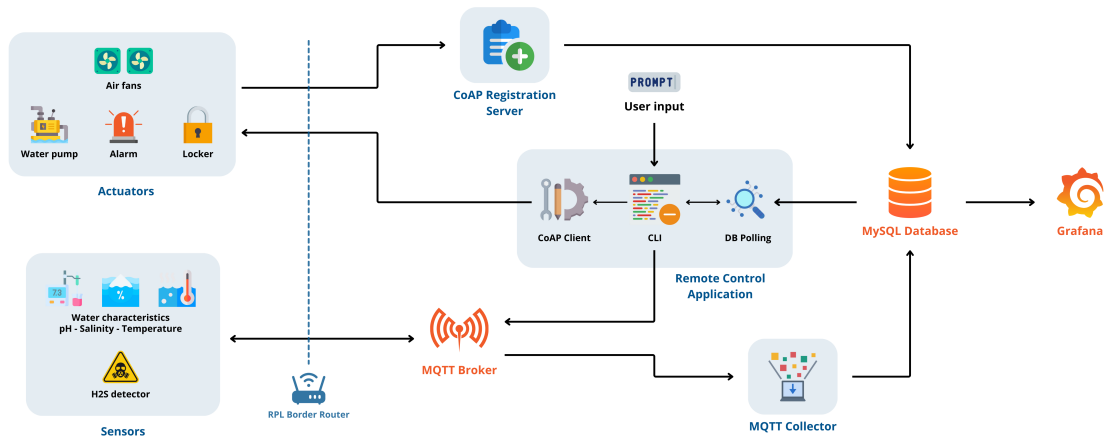
Chapter 2

Architecture

Figure 2.1: System architecture schema.

As we can see we have two type of networks, the **MQTT Network** and the **CoAP Network**, which interact with a **Back-end System**.

2.1 MQTT Network

All sensors are deployed in the MQTT Network. In particular we have two type of sensors:

- **Temperature**, **pH** and **Salinity** sensor, used to monitor water characteristics.

- **Hydrogen Sulfide** sensor, used to detect toxic gases which need to be aspirated.

These sensors measure values in real-time during entire production process. A key part of MQTT Network is the MQTT Broker, which allows the management of communication between sensors and the MQTT Client or the Remote Control Application thanks to the publish/subscribe mechanism.
In this project we used Mosquitto, an open source MQTT broker.

2.2 CoAP Network

All the actuators are deployed in a CoAP Network. There are four actuator types:

- **Air fans** allow to cool the water if its temperature is above a certain level. They are also used to drain out the H_2S which can be produced during the process.

- **Water pump** stabilizes the pH adding an acid or base solution accordingly to the ranges in which the parameter must belong to. It's also used to decrease the salinity level adding pure water.

- **Alarm** is activated when the salinity level is under a certain threshold: in this way the operator can intervene adding salts to the solution to restore a correct salinity level.

- **Locker** is activated when H_2S is detected to lock the tanning drum gate, avoiding possibly fatal inhalation of the gas by the operators.

The CoAP Network allows actuators to expose a resource retrievable by other network component once the registration as occurred.

The registration phase is implemented through a discovery procedure: the actuator sends a message to a CoAP Registration Server, which saves its IPv6 address into a database. In this way the Remote Control Application can read the database and send commands through the CoAP Client to the actuators.

2.3 Back-end System

The backend system include the following entities:

- **MQTT Client** handles the data collected by sensors. To do this, it initially subscribes to the topics in which the sensors will later publish the data and, once these have been published, inserts them into the database.

- **CoAP Registration Server** registers nodes which belong to the CoAP Network in the database: after an actuator has joined the *RPL DODAG* created by the border router, it will send a message to the CoAP Registration Server before exposing its resource.

- **Remote Control Application** performs multiple crucial roles in the system:

  - *Command Line Interface* allows management and visualization of sensors and actuators status by the operator. The user has access to some commands which allow the modification of sensors parameters using MQTT, the starting and stopping of the pickling process and the visualization of the overall status of the system, thanks to a periodic access to the database to retrieve the latest data.

- – *Database Polling* is used to monitor and analyze the latest sensor values stored in the database by the MQTT Client, such that the actuators status can be updated, also accordingly to the ranges settled by the user.

- – *CoAP Client* allows to send command to the actuators when needed, exploiting the CoAP Network.

  To minimize the network traffic, the Remote Control Application avoids sending useless status updates to the actuators, since it knows their current status.

  In addition, it allows to periodically ping actuators to discover if something is not working anymore.

- **MySQL Database** to store the values measured by the sensors, the status of actuators and the current sensors parameters, providing an up-to-date repository for the Remote Control Application and Grafana.

- **Grafana** which is an open-source data monitoring platform used to visualize analytics about the data stored within the database.

2.4   Data Encoding

In resource-constrained scenarios as IoT systems, a crucial aspect is the choice of protocol to use. For data encoding, we opted to use **JSON**, which offers a simpler and more lightweight syntax compared to XML, since we have no constraints which force us to use it, as the integration in pre-existing systems that exploit XML instead of JSON. As results, JSON generally leads to smaller payloads, which allows less bandwidth usage and less resource utilization by devices.

Chapter 3

Deployment and Execution

3.1 Sensors

As already mentioned we have two type of sensors, and these publish the values detected in their respective topic *sensor/sn*, where *sn* is the relative sensor name. In particular we have:

- **sensor/temp_pH_sal** topic.

- **sensor/H$_2$S** topic.

We have different time of sensing and publication of values, based on the current state of the system:

- In **normal status** we have:

  1. *Sensing time*:
     - 10 seconds for temperature_pH_salinity sensor.
     - 8 seconds for H$_2$S sensor.

  2. *Publication time*:
     - 40 seconds for temperature_pH_salinity sensor.
     - 32 seconds for H$_2$S sensor.

- In **warning status** we have:

  1. *Sensing time*:
     - 5 seconds for temperature_pH_salinity sensor.
     - 4 seconds for H$_2$S sensor.

  2. *Publication time*:
     - 15 seconds for temperature_pH_salinity sensor.
     - 12 seconds for H$_2$S sensor.

- In **normal-warning** and **warning-normal** transitions values are immediately published.

The time values are very low because frequent monitoring is necessary throughout the production process, changes in the values could have serious consequences for both product and workers.

Energy can still be saved by publishing the values after a certain time if the system is always in the same state and by changing the frequency of the measurements in the normal-warning transition and vice versa.

To check the status of the system we rely on the data detected by the sensors, in particular for all values we have a maximum threshold, a minimum threshold and a delta, except for the temperature where we don't have the minimum threshold since it's not necessary, whereas for the H_2S sensor we don't have these values since the H_2S must only be detected or not without a value.

The verification of the system status for each sensor is done as follows:

- **H_2S** sensor:

  1. H_2S **not detected** $\rightarrow$ **normal** status.
  2. H_2S **detected** $\rightarrow$ **warning** status.

- **Temperature_pH_salinity** sensor:

  - If we are in normal status:
    1. All **values** are **within bounds** $\rightarrow$ **normal** status.
    2. At least one **value out of bounds** $\rightarrow$ **warning** status.
       In this case, it is also kept track if the values have exceeded the limits towards the minimum or towards the maximum.

  - If we are in warning status:
    1. If the value had exceeded the limit towards the minimum (maximum) and the sensed value is greater (lower) than minimun (maximum) threshold + (-) delta $\rightarrow$ **normal** status.
       This condition must be true for all type of values out of bounds.
    2. If the previous conditions are not true $\rightarrow$ **warning** status.

In the Figure 3.1 below we can see a diagram of the determination of the system status for the temperature_pH_salinity sensor.
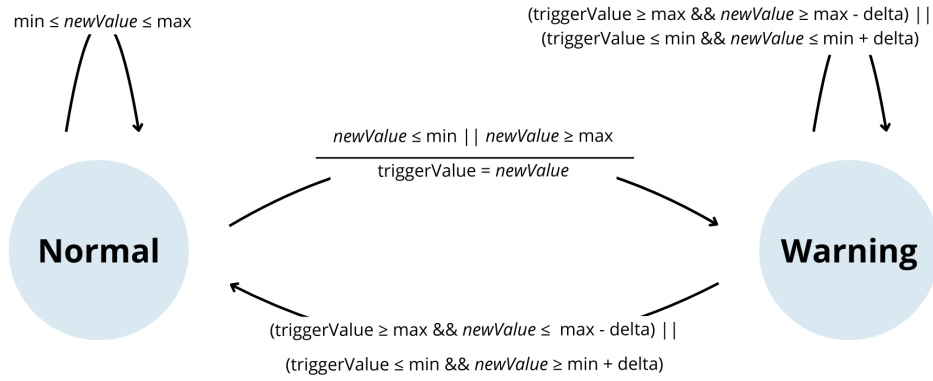
Figure 3.1: Sensor state machine.

To save more energy, the sensors can be started and stopped to do their data sensing task, via a command sent by the user from the Remote Control Application. In particular, the sensor will subscribe to the topic *pickling*, on which it can receive the following commands:

- **start** that will start data sensing.

- **stop** that will stop data sensing.

Initially, the sensor does not sense data until it receives the start command.
In addition to the start and stop commands, the temperature_pH_salinity sensor can receive, always from the user via the Remote Control Application, the values that will update the minimum/maximum thresholds and delta values. To do this we have three different topics, one for each type of data, which the sensor subscribes to:

- **params/temperature** topic, for maximum threshold and delta value for temperature.

- **params/ph** topic, for minimum/maximum threshold and delta value for pH.

- **params/salinity** topic, for minimum/maximum threshold and delta value for salinity.

The data sensed by the sensors are generated randomly, but it is possible to modify them using the button on the dongle. In particular we have:

- **H_2S** sensor: Initially, 0 is generated and the sensor continues to generate this value until the button is pressed which reverses the generated value. Thus, if 0 is generated, when the button is pressed the sensor will start generating 1 and vice versa.

- **Temperature_pH_salinity** sensor: The values are generated according to a Gaussian distribution. We therefore have an initial mean and an initial standard deviation for temperature, pH and salinity. Interaction with the button takes place as follows:

- Button pressed once in three seconds → temp_mean += temp_stddev

- Button pressed twice in three seconds → pH_mean += pH_stddev

- Button pressed three times in three seconds → sal_mean += sal_stddev

- Button pressed four times in three seconds → temp_mean -= temp_stddev

- Button pressed five times in three seconds → pH_mean -= pH_stddev

- Button pressed six times in three seconds → sal_mean -= sal_stddev

This allows you to increase or decrease the value of the generated data, so that you can simulate the sensing of values out of bounds or normal-warning status transition and vice versa.

In the Figure 3.2 we can see an example of what was said in the case of temperature.
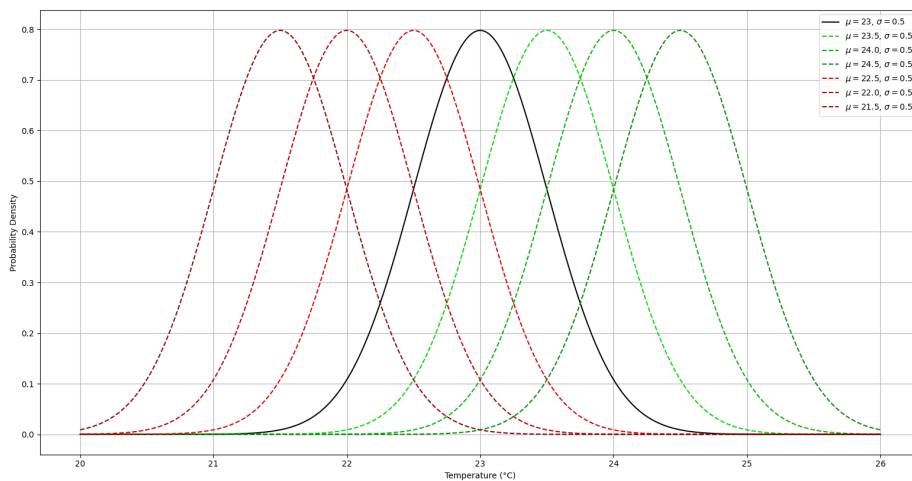

Figure 3.2: Gaussian distribution chart of temperature at mean change after button press

3.2   Actuators

When an actuator is powered on, firstly it must register itself to the CoAP Registration Server sending a JSON containing the resource name which the actuator is going to expose and its initial status, such that the registration server can then store them in the MySQL database together with the IPv6 address. In this way this can be retrieved and used to send commands to the actuator. When an actuator receive a command, it performs the relative action and then send a response to the Remote Control Application:

- *CHANGED 204* if the action has been performed correctly.

- *BAD OPTION 402* if an unknown action was specified.

- *BAD REQUEST 400* if no action was specified.

In Figure 3.13 a schema describes the general actuators behavior.
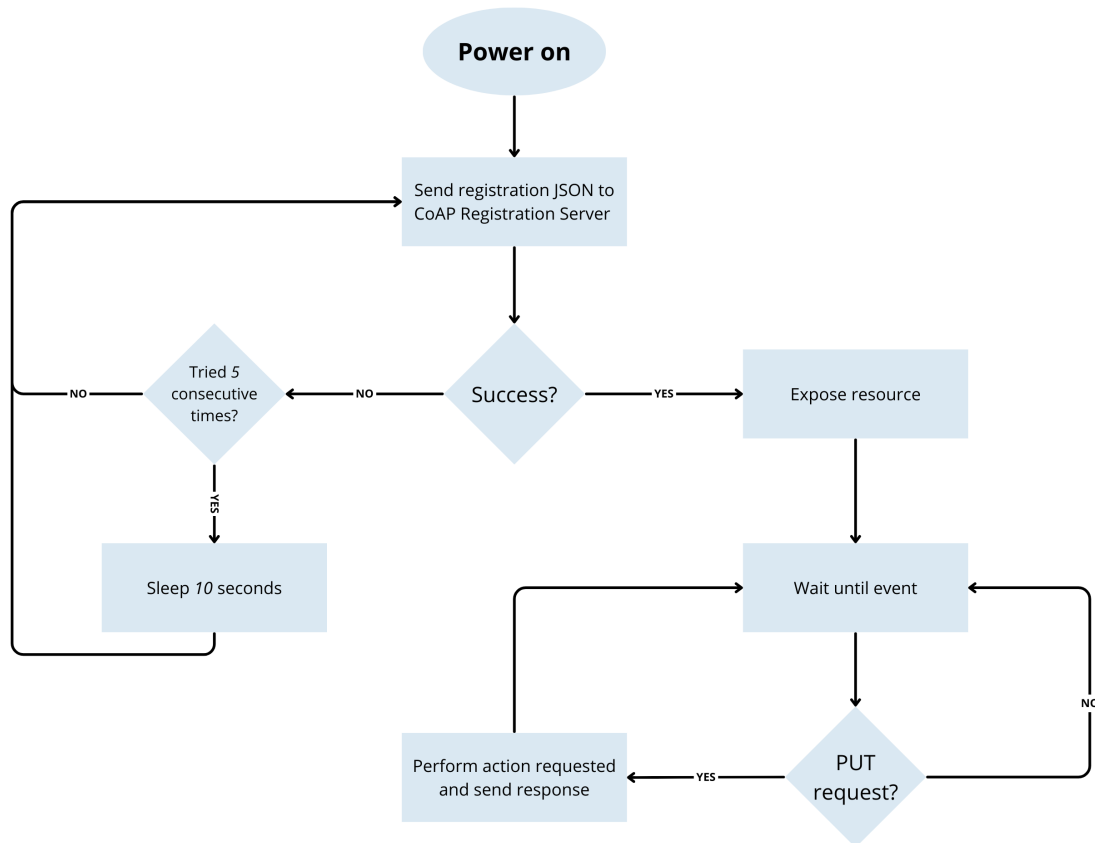

Figure 3.3: Schema for general actuator behavior.

The actions performed are actuator-specific, and they are various:

- **Air fans** can be turned on in *cooling* mode to decrease the solution temperature and/or in *exhaust* mode when H_2S is detected to drain it.

- **Water pump** can add an *acid* or *basic* solution to restore correct pH values or it can add *pure* water to decrease solution salinity.

- **Alarm** is turned *on* when the salinity level is under a certain threshold, while is switched *off* when it goes back above.

- **Locker** is turned *on* when H_2S is detected to lock the tanning drum gate and then switched *off* to unlock it when H_2S is no more detected.

All those action are simulated in the system using LEDs, as described in Figure 3.4.

Water pump

| Connecting | | Off | Acid | Pure | Base |

Not connected | Connected

Air fans

| Connecting | | Off | Exhaust | Cooling | Both |

Not connected | Connected

Alarm

| Connecting | | Off | On |

Not connected | Connected

Locker

| Connecting | | Off | On |

Not connected | Connected

Figure 3.4: LED colors meaning for each actuator type.
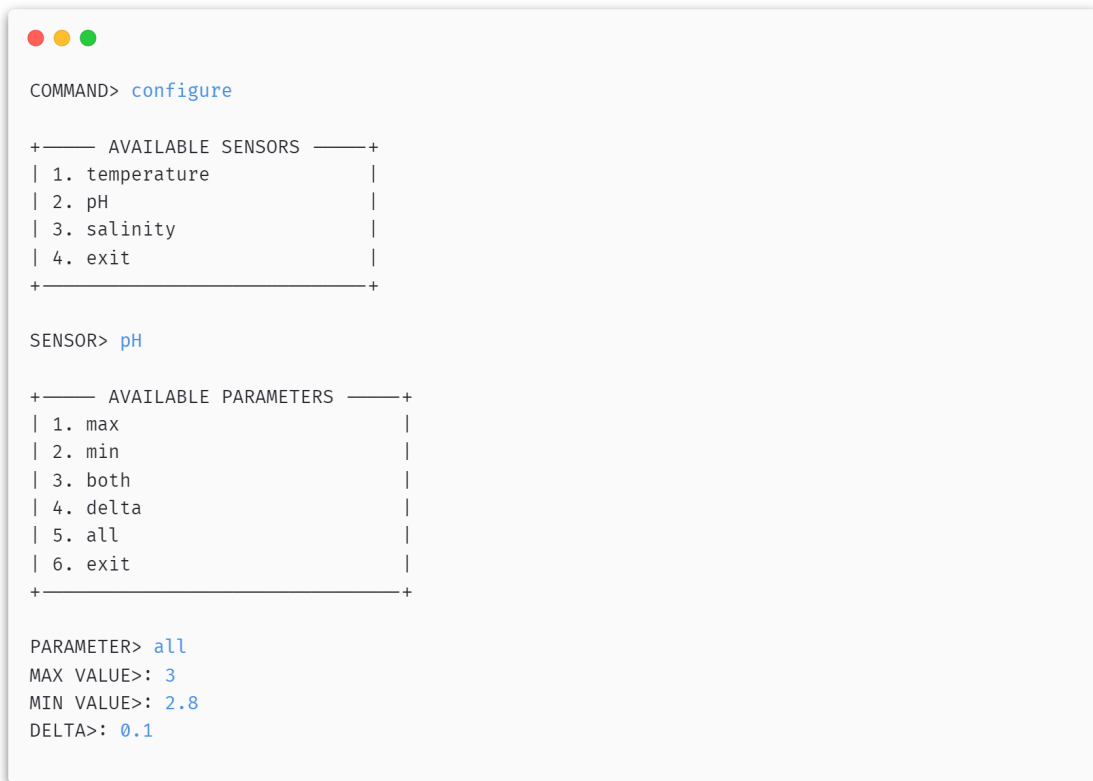
3.3   Remote Control Application

The Remote Control Application performs multiple crucial roles in the system, exposing a CLI which allows the user to interact and monitor the system through different commands.

Figure 3.5: CLI initial display.

3.3.1   Configure

As we can see in Figure 3.6, it allows to configure the minimum/maximum thresholds and the delta parameter for each type of data sensed: firstly the user select the sensor, then he specify the parameters that need to be changed. The application ensure the new parameters are entered in a consistent way.

```
COMMAND> configure

+———— AVAILABLE SENSORS ————+
| 1. temperature            |
| 2. pH                     |
| 3. salinity               |
| 4. exit                   |
+———————————————————————————+

SENSOR> pH

+———— AVAILABLE PARAMETERS ————+
| 1. max                       |
| 2. min                       |
| 3. both                      |
| 4. delta                     |
| 5. all                       |
| 6. exit                      |
+——————————————————————————————+

PARAMETER> all
MAX VALUE>: 3
MIN VALUE>: 2.8
DELTA>: 0.1
```
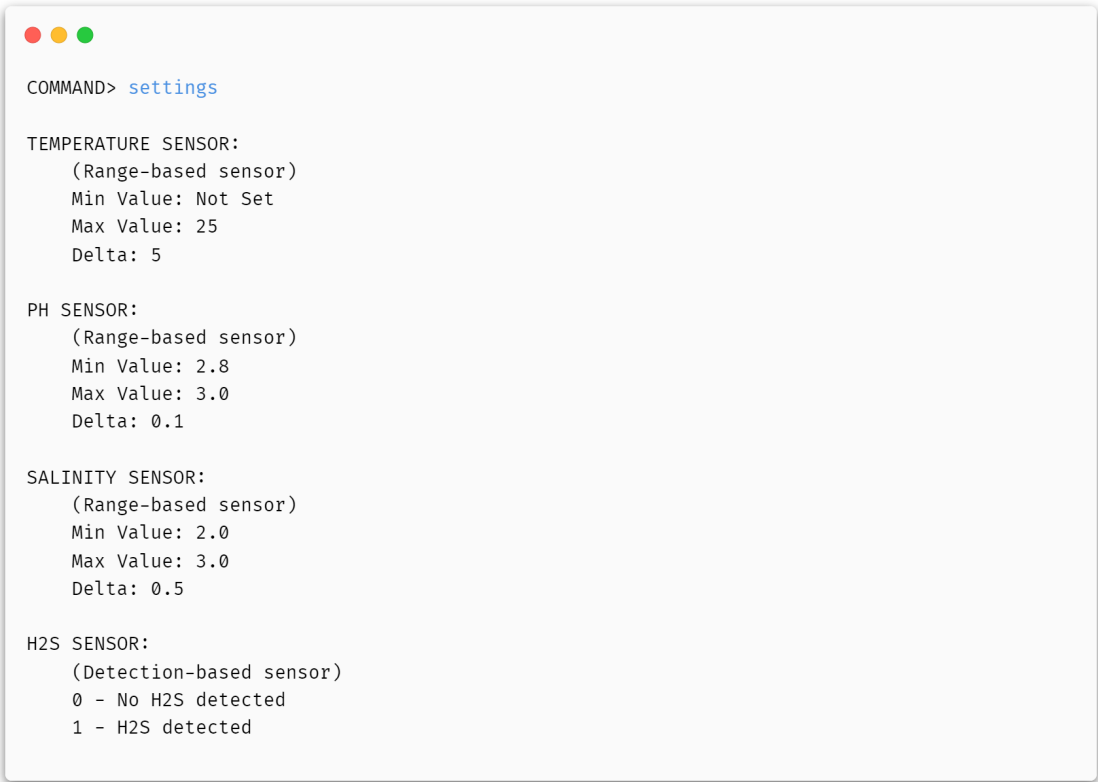
Figure 3.6: Output for *Configure* command.

3.3.2   Settings

As we can see in Figure 3.7, it displays the value of current parameters for each type of data, giving some additional information on how each sensor works.

```
COMMAND> settings

TEMPERATURE SENSOR:
    (Range-based sensor)
    Min Value: Not Set
    Max Value: 25
    Delta: 5

PH SENSOR:
    (Range-based sensor)
    Min Value: 2.8
    Max Value: 3.0
    Delta: 0.1

SALINITY SENSOR:
    (Range-based sensor)
    Min Value: 2.0
    Max Value: 3.0
    Delta: 0.5

H2S SENSOR:
    (Detection-based sensor)
    0 - No H2S detected
    1 - H2S detected
```
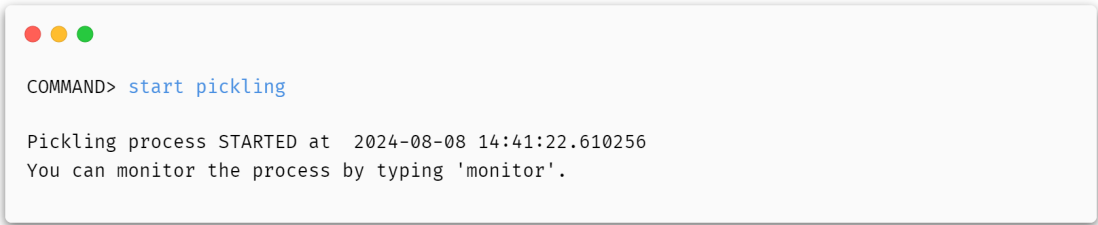
Figure 3.7: Output for *Settings* command.

3.3.3 Start/stop pickling

Those commands manage the pickling process:

- *Start Pickling* starts the data sensing by the sensors and the data polling from the database for their analysis and status updating of the actuators. The application memorize the current status of the overall system which can be displayed using the *Monitor* command.

```
COMMAND> start pickling

Pickling process STARTED at  2024-08-08 14:41:22.610256
You can monitor the process by typing 'monitor'.
```
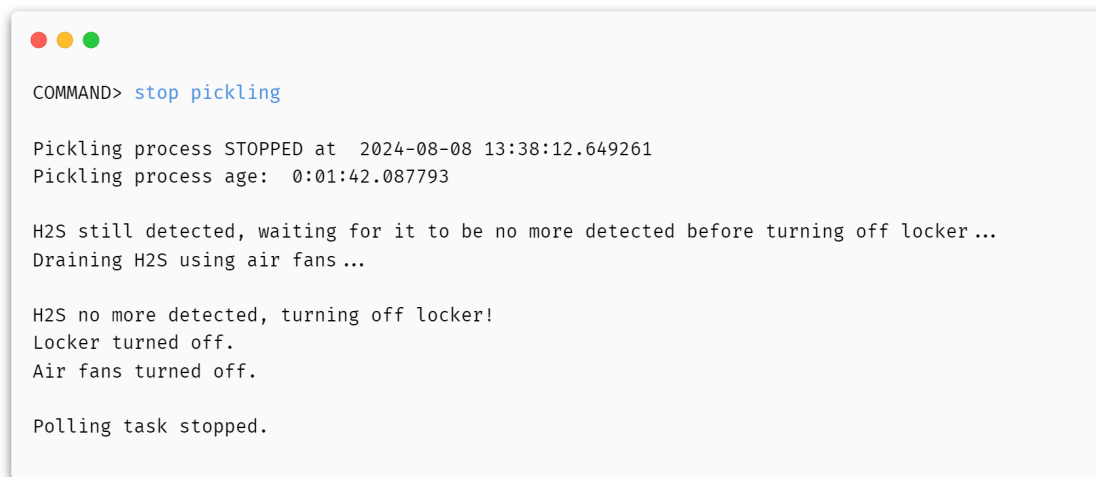
Figure 3.8: Output for *Start Pickling* command.

- *Stop Pickling* stops the process in a **safe** way. It firstly turns off the water pump and the alarm and then, if H$_2$S is still detected, it sets the air fans in exhaust mode and it maintains active the locker until the gas is drained out

```
COMMAND> stop pickling

Pickling process STOPPED at  2024-08-08 13:38:12.649261
Pickling process age:  0:01:42.087793

H2S still detected, waiting for it to be no more detected before turning off locker ...
Draining H2S using air fans ...

H2S no more detected, turning off locker!
Locker turned off.
Air fans turned off.

Polling task stopped.
```

Figure 3.9: Output for *Stop Pickling* command.
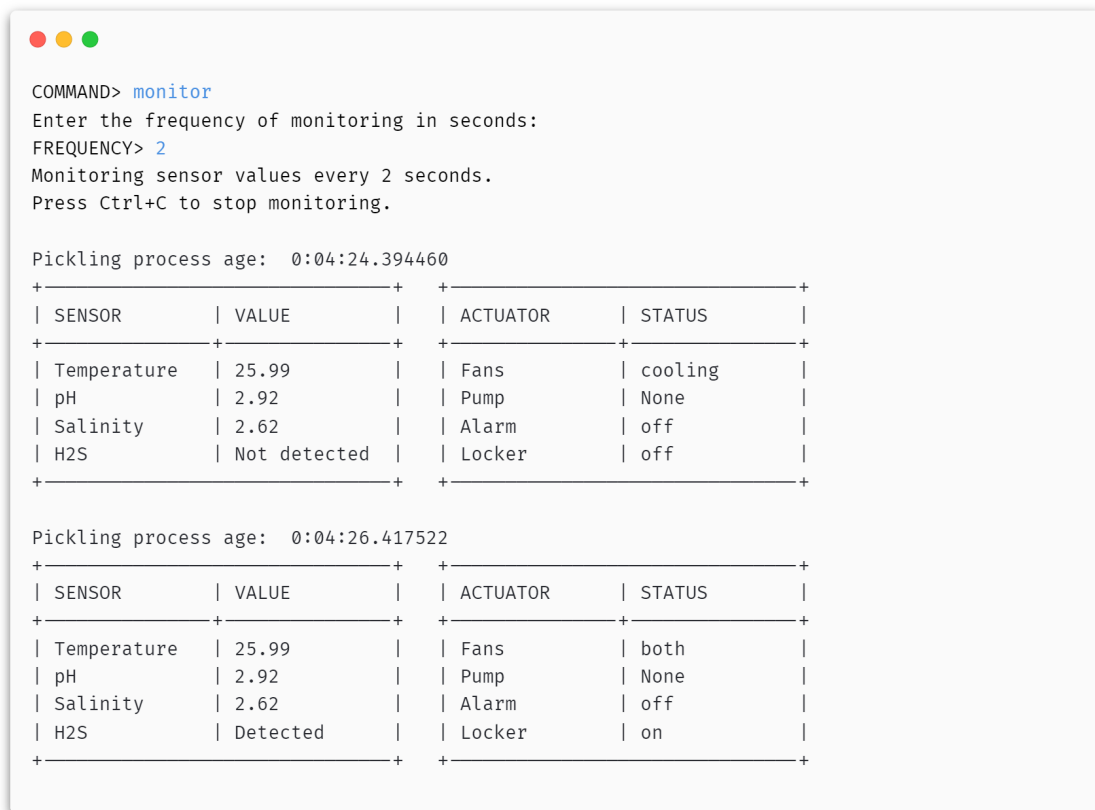
and it is no more detected. In addition, In the end, it stops the data sensing and the data polling. Commands are always sent only if the actuators are not already in the correct status.

Their outputs are shown in Figure 3.8 and Figure 3.9.

3.3.4 Monitor

As shown in Figure 3.10, it allows the displaying of the overall system status during the pickling process. The user enters how often the displayed values should be updated, then the last data sensed and the current actuator status is shown each time, together with the pickling process age.

Monitor also manages values which are too old (*None*) and actuators that have never been connected yet (*None*) or which have been disconnected (*Error*).

```
COMMAND> monitor
Enter the frequency of monitoring in seconds:
FREQUENCY> 2
Monitoring sensor values every 2 seconds.
Press Ctrl+C to stop monitoring.

Pickling process age:  0:04:24.394460
+----------------------------+    +--------------------------------+
| SENSOR       | VALUE       |    | ACTUATOR     | STATUS          |
+----------------------------+    +--------------------------------+
| Temperature  | 25.99       |    | Fans         | cooling         |
| pH           | 2.92        |    | Pump         | None            |
| Salinity     | 2.62        |    | Alarm        | off             |
| H2S          | Not detected |   | Locker       | off             |
+----------------------------+    +--------------------------------+


Pickling process age:  0:04:26.417522
+----------------------------+    +--------------------------------+
| SENSOR       | VALUE       |    | ACTUATOR     | STATUS          |
+----------------------------+    +--------------------------------+
| Temperature  | 25.99       |    | Fans         | both            |
| pH           | 2.92        |    | Pump         | None            |
| Salinity     | 2.62        |    | Alarm        | off             |
| H2S          | Detected    |    | Locker       | on              |
+----------------------------+    +--------------------------------+
```
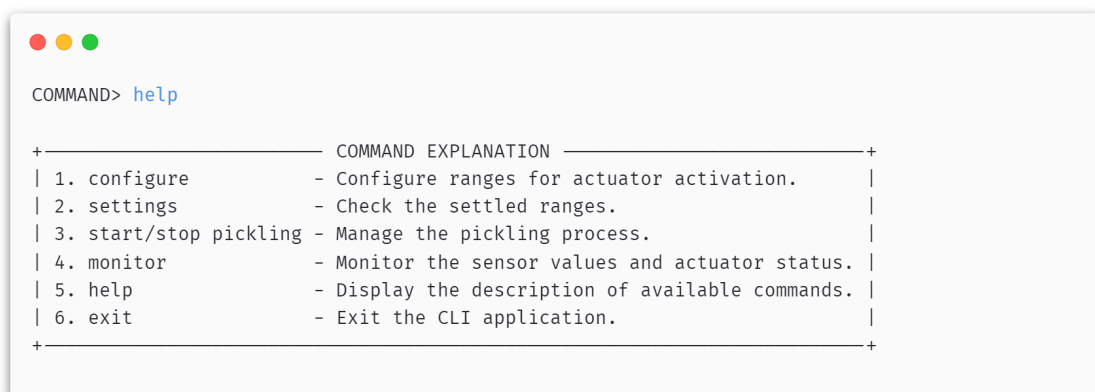
Figure 3.10: Output for *Monitor* command.

3.3.5   Help

It shows the list of available commands with a short description, as it can be seen in Figure 3.11.
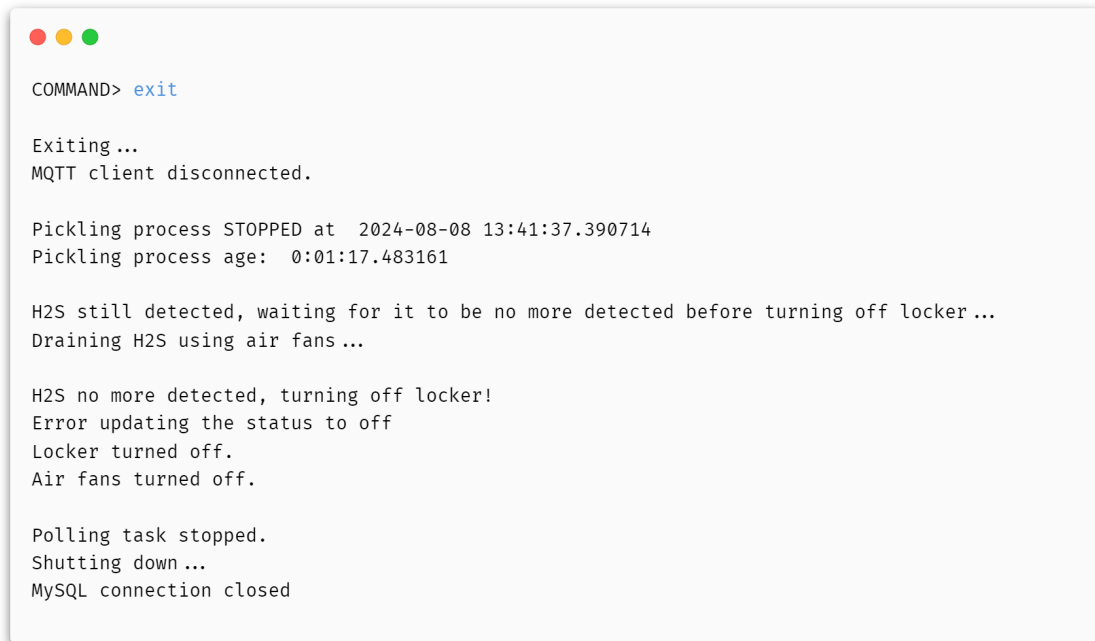
```
COMMAND> help

+------------------------ COMMAND EXPLANATION ------------------------+
| 1. configure        - Configure ranges for actuator activation.    |
| 2. settings         - Check the settled ranges.                    |
| 3. start/stop pickling - Manage the pickling process.              |
| 4. monitor          - Monitor the sensor values and actuator status. |
| 5. help             - Display the description of available commands. |
| 6. exit             - Exit the CLI application.                    |
+--------------------------------------------------------------------+
```

Figure 3.11: Output for *Help* command.

3.3.6   Exit

It terminates the application by ensuring that the associated processes are correctly terminated: if the pickling process is active, it takes care to stop it firstly, possibly waiting if H_2S is still detected. Its output is shown in Figure 3.12.

```
COMMAND> exit

Exiting ...
MQTT client disconnected.

Pickling process STOPPED at  2024-08-08 13:41:37.390714
Pickling process age:  0:01:17.483161

H2S still detected, waiting for it to be no more detected before turning off locker ...
Draining H2S using air fans ...

H2S no more detected, turning off locker!
Error updating the status to off
Locker turned off.
Air fans turned off.

Polling task stopped.
Shutting down ...
MySQL connection closed
```
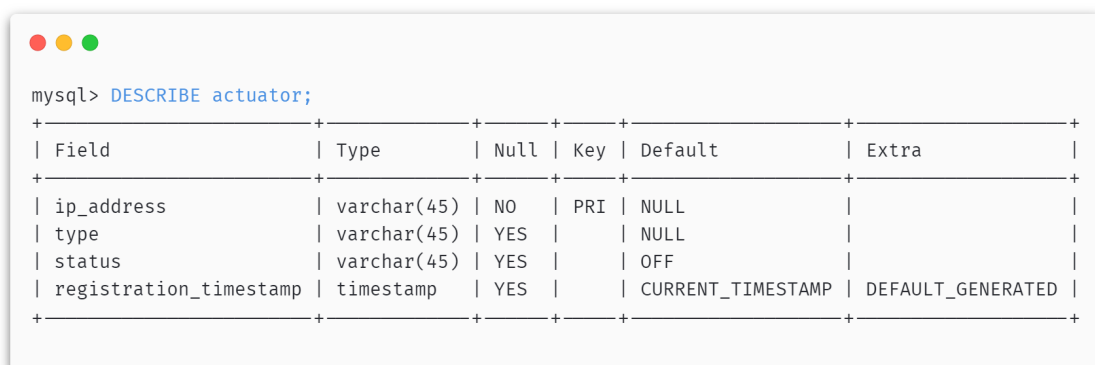
Figure 3.12: Output for *Exit* command.

3.4   Database

In the *leathersense* database we have the following tables:

- **actuator**: Table used for monitoring actuators and their status. If the status of an actuator changes, a new record is entered into the table. This is done so that the user can always obtain the updated status of the actuator.
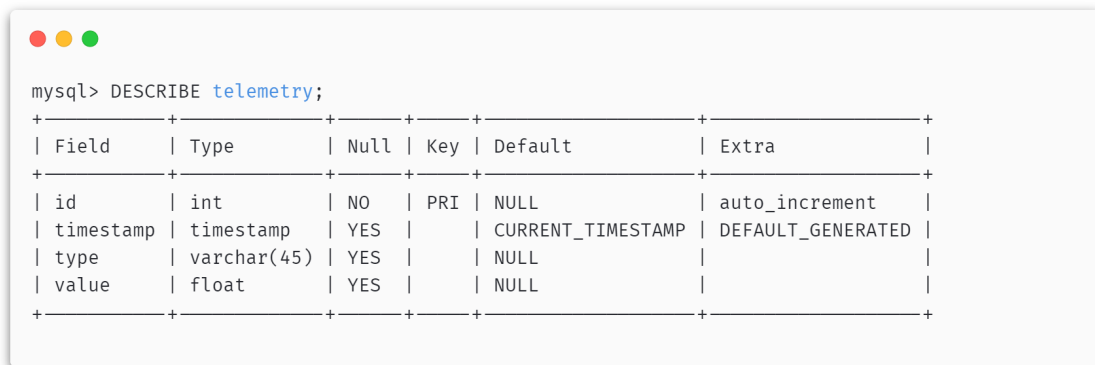
```
mysql> DESCRIBE actuator;
+----------------------+-------------+------+-----+-------------------+-------------------+
| Field                | Type        | Null | Key | Default           | Extra             |
+----------------------+-------------+------+-----+-------------------+-------------------+
| ip_address           | varchar(45) | NO   | PRI | NULL              |                   |
| type                 | varchar(45) | YES  |     | NULL              |                   |
| status               | varchar(45) | YES  |     | OFF               |                   |
| registration_timestamp | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+----------------------+-------------+------+-----+-------------------+-------------------+
```

Figure 3.13: Description of the actuator table.

- **telemetry**: The telemetry table contains all data published by the sensors in their respective topics. It is used by the user to monitor the sensed data and by the actuators to update their status.
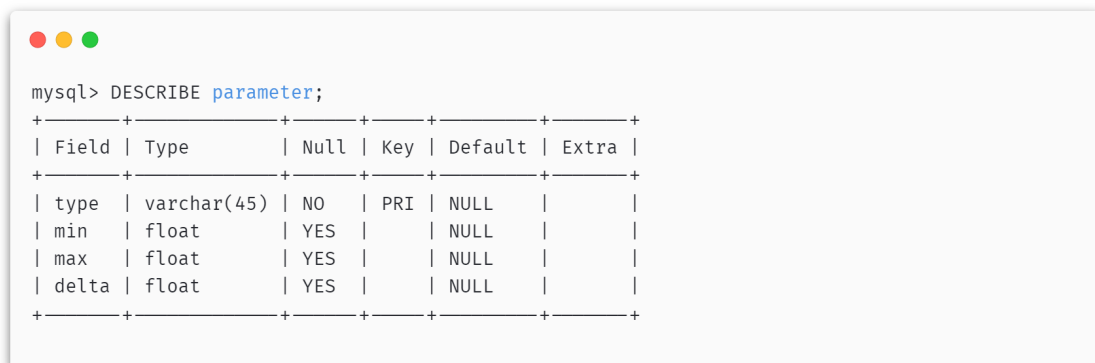
```
mysql> DESCRIBE telemetry;
+-----------+-------------+------+-----+-------------------+-------------------+
| Field     | Type        | Null | Key | Default           | Extra             |
+-----------+-------------+------+-----+-------------------+-------------------+
| id        | int         | NO   | PRI | NULL              | auto_increment    |
| timestamp | timestamp   | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
| type      | varchar(45) | YES  |     | NULL              |                   |
| value     | float       | YES  |     | NULL              |                   |
+-----------+-------------+------+-----+-------------------+-------------------+
```

Figure 3.14: Description of the telemetry table.

- **parameter**: Table allowing the storage of parameters for each type of data within the database. This makes it possible to retrieve this data to give the user a display that is consistent with the current state of the system.

```
mysql> DESCRIBE parameter;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| type  | varchar(45) | NO   | PRI | NULL    |       |
| min   | float       | YES  |     | NULL    |       |
| max   | float       | YES  |     | NULL    |       |
| delta | float       | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

Figure 3.15: Description of the param table.

Chapter 4

Grafana

Grafana provides an interface through which the user can graphically monitor the overall process status.

The graphical interface has two sections:

- **Status**, Fig. 4.1: it shows the current status of each actuator and the current value of each parameter monitored, providing also a feedback on the ranges in which each one must stay.
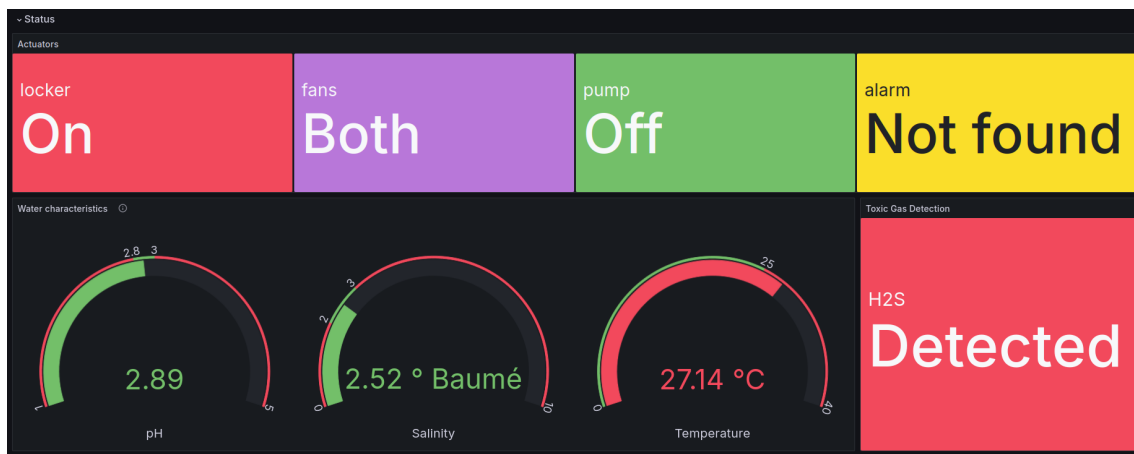

Figure 4.1: Current status interface.

- **Trends**, Fig. 4.2: it shows the line plot of each water parameter highlighting the minimum, maximum and average values. Furthermore, it displays the historical values of H_2S detection results.
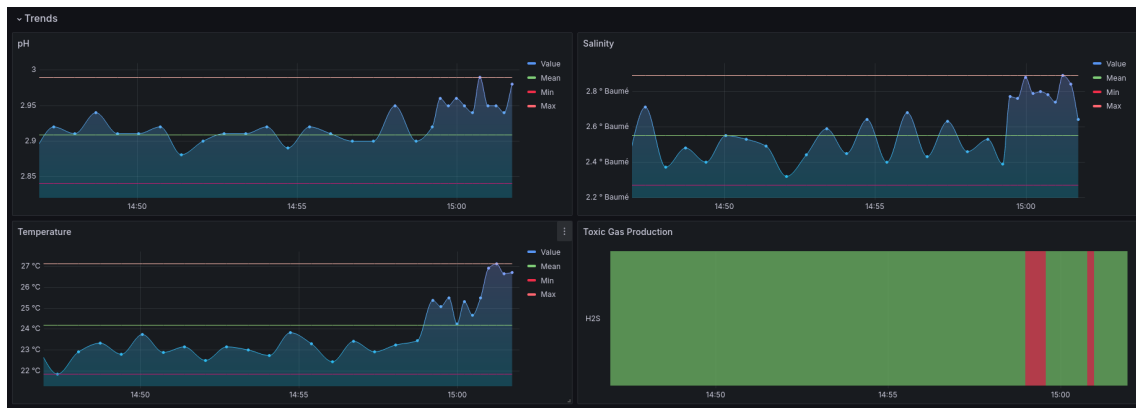
Figure 4.2: Value trends interface.

Chapter 5

Use Case

In the leather tanning industry, maintaining optimal environmental conditions and safety standards is paramount to ensuring product quality and worker safety. In particular, during the pickling phase, the company faces challenges in maintaining consistent product quality due to fluctuations in environmental conditions, such as pH, salinity, and temperature, and is concerned about worker safety due to the potential release of hydrogen sulfide (H_2S) gas during processing.

The implementation of an IoT solution as the LeatherSense system enables the automatic and continuous monitoring and promptly regulation of those critical parameters during the pickling phase, through different sensors and actuators devices which are installed in the tanning drums and settled using the Remote Control Application.

5.1 Benefits

This solution leads to several benefits:

- **Improved Product Quality**: By continuously monitoring pH, salinity, and temperature, the system ensures that leather is processed under optimal conditions, reducing defects and improving overall product quality.

- **Enhanced Worker Safety**: The system detects H_2S levels in real-time, automatically activating air fans to ventilate the area and locking the tanning drum gate to prevent accidental exposure to toxic gases.

- **Operational Efficiency**: Automated control of environmental factors, such as the activation of water pumps and air fans, reduces the need for manual intervention, leading to a more efficient and consistent production process.

5.2 Functions implemented

LeatherSense implements some crucial function:

- **Real-time Monitoring**: Sensors continuously track water characteristics (pH, salinity, temperature) and H2S levels, publishing data to the MQTT network.

- **Automatic Regulation**: Actuators automatically adjust environmental conditions based on sensor data. For example, water pumps regulate pH and salinity, while air fans manage temperature and toxic gas levels.

- **Data Logging and Visualization**: All data is stored in a MySQL database and visualized through Grafana, enabling operators to monitor trends and analyze process performance.

5.3 Conclusion

This IoT system can provide a comprehensive solution for the leather tanning industry, specifically designed to optimize the pickling phase. By automating the monitoring and regulation of key environmental factors, the system improves product quality and operational efficiency also enhancing worker safety. The system's ability to adapt to real-time conditions and provide actionable insights makes it a valuable tool for modernizing the leather tanning process.

Bibliography

[1] BuyLeatherOnline, "Pelle conciata al vegetale: Storia e caratteristiche," Sep. 2022. [Online]. Available: https://buyleatheronline.com/it/blog/cosa-si-intende-per-pelle-conciata-al-vegetale-n27

[2] D. Pirolo, "Piclaggio: Preconcia delle pelli," Apr. 2015. [Online]. Available: https://www.italiapelle.com/piclaggio-preconcia-delle-pelli/