IoT & 3D Intelligent Systems project
University of Modena and Reggio Emilia

UNIMORE

SMASHBOX

Simone Bugo
Francesco Marzo

SMASHBOX

Who we are

We are two students of Artificial Intelligence Engineering, passionate about technological innovation, security systems, and the future of smart infrastructures.

Francesco Marzo

Simone Bugo

The Birth of SMASHBOX

Driven by a shared vision to revolutionize traditional banking security, we created SMASHBOX: an intelligent, interconnected, and biometrically secured system of safe deposit boxes

SMASHBOX

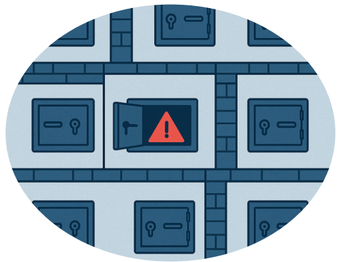
State of the art

Safety deposit box systems are obsolete

Once customers enter into a lease agreement for a safe deposit box, a physical key is assigned to them and this could lead to several problems

The key could be lost

The bank cannot retain a copy of the key due to privacy regulations, complicating the recovery process



No intercommunication between boxes

Every box is separated from the others, if one of them has been compromised the other are not informed

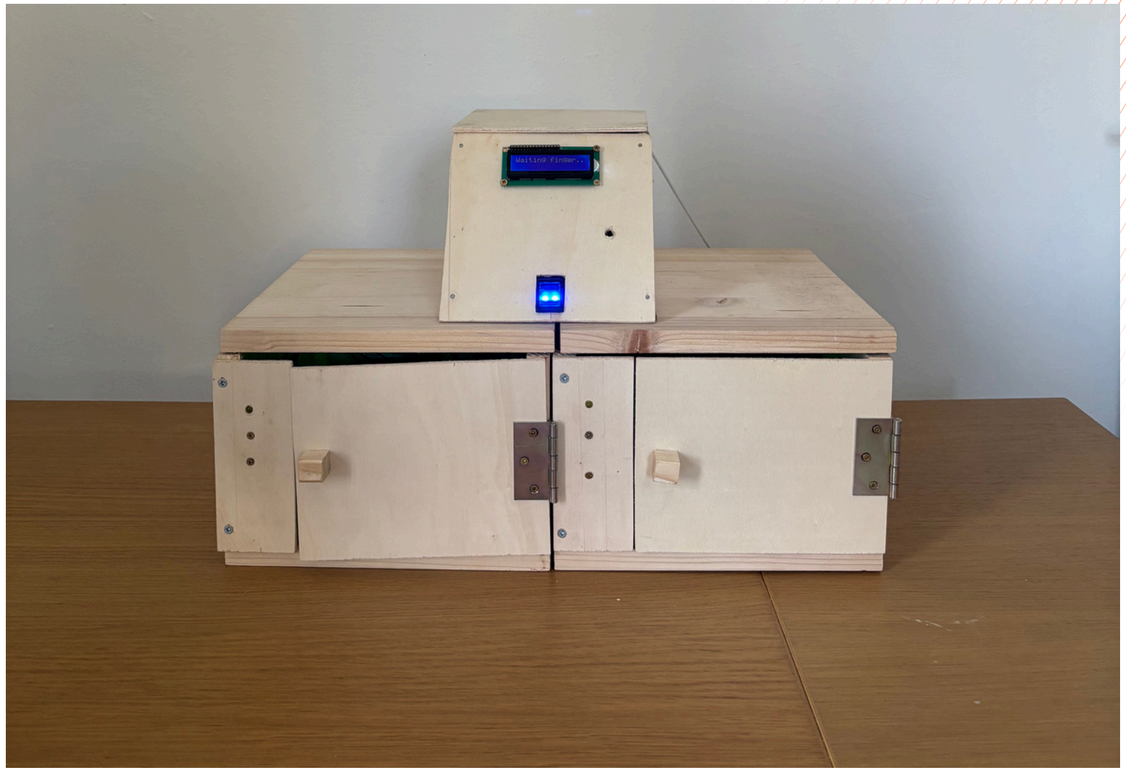
Lack of controll on single box

Boxes are located in vault rooms which are protected by alarms and security cameras but there isn’t a direct controll on the single box

So we thought about a possible solution...

SMASHBOX

Secure Monitoring And Smart Hub for Biometric Optical boXes



Vision

SMASHBOX

“SMASHBOX is redefining the traditional concept of bank safe deposit boxes, transforming them into intelligent, interconnected, and biometrically secured systems.
We are moving beyond static, conventional storage models to usher in a new era of dynamic, personalized, and digitally enhanced security.”

SMASHBOX

Key points

Innovators in Security Technology

We combine advanced sensor networks, biometric authentication, and smart connectivity to revolutionize traditional safe deposit boxes.

Bridging Physical and Digital Worlds

Our solutions connect physical security infrastructure with real-time digital monitoring, empowering customers with greater control and awareness.

Focused on Personalized Safety

SMASHBOX offers a user-centric experience: each safe deposit box is individually monitored, alert-enabled, and biometrically protected.

SMASHBOX

Our Product

SmashBox is composted by two main physical parts:

- One **central** part which is used by the bank operator
- Many **acquisition** boxes which are assigned to the clients

Central

Acquisitions

SMASHBOX

Our Product

Acquisition description

Components inside Acquisition

Arduino Uno

Temperature and Humidity sensor

Accellerometer

2x IR-SENSOR

Electromagnet

SMASHBOX

Our Product

Central description

Components inside Central

Arduino Uno

Fingerprint sensor

Display LCD

Security button

How does it work?

SMASHBOX

Registration process of new fingerprints

How does it work?

SMASHBOX

Unlock process of registered fingerprints

SMASHBOX

Notification System

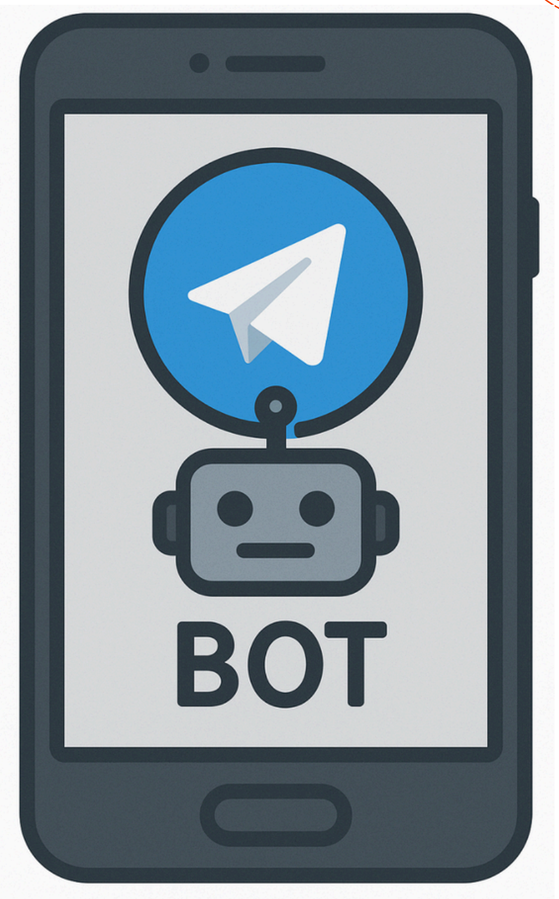
We have introduced an enhanced service that provides users with all the essential information they need, including:

Real-time notifications:

- Opening of the personal safe deposit box
- Alerts or anomaly reports

Activity history available:

- The user can query the bot to retrieve the log list related to their box (cmd: “/log”)



SMASHBOX

Price

Our goal is to deliver a product of the highest quality at an affordable price
The costs outlined in this slide represent the expenses associated with the prototype.

Central

- Components: 60€
- Casing: 10€

Each Acquisition

- Components: 80€
- Casing: 10€

SMASHBOX

Price

We aim to lower the prototype price with the actual production rollout of our product.
Our goal is to reach the price of:

- **60€ for the central**
- **70€ for each boxes**

Including hardware and casing

Estimating a purchase of around 100 units, we would be able to offer a competitive price of 7.000€

SMASHBOX

Price

To the 'physical' prices mentioned earlier, we need to add:

Maintenance costs:

- **150€/month → 1800€/year**

Software development:

- **working demo: 15.000€**
- **stable version: 40.000€**

Total estimate: 50.000€

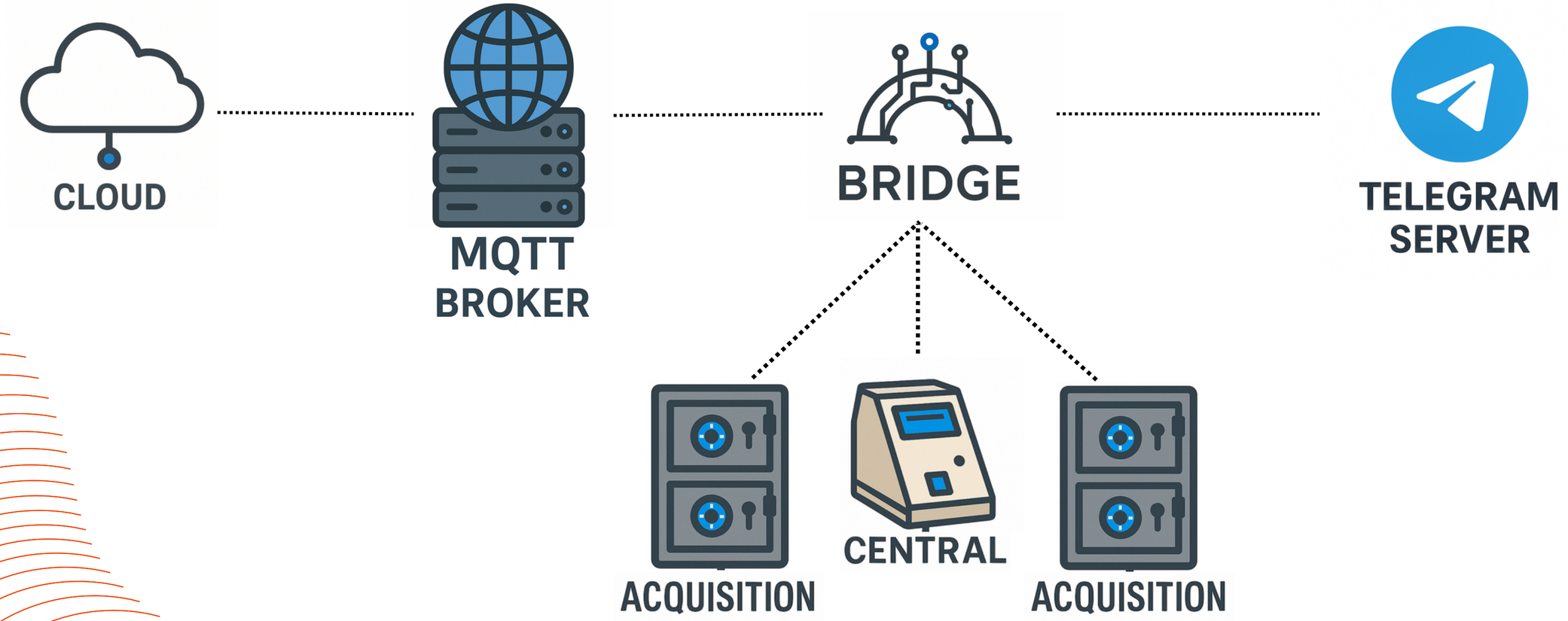
With the purchase of our system we offer a 1-year warranty, technical assistance and a training course for operators

SMASHBOX

Architecture

SMASHBOX

Architectural schema



SMASHBOX

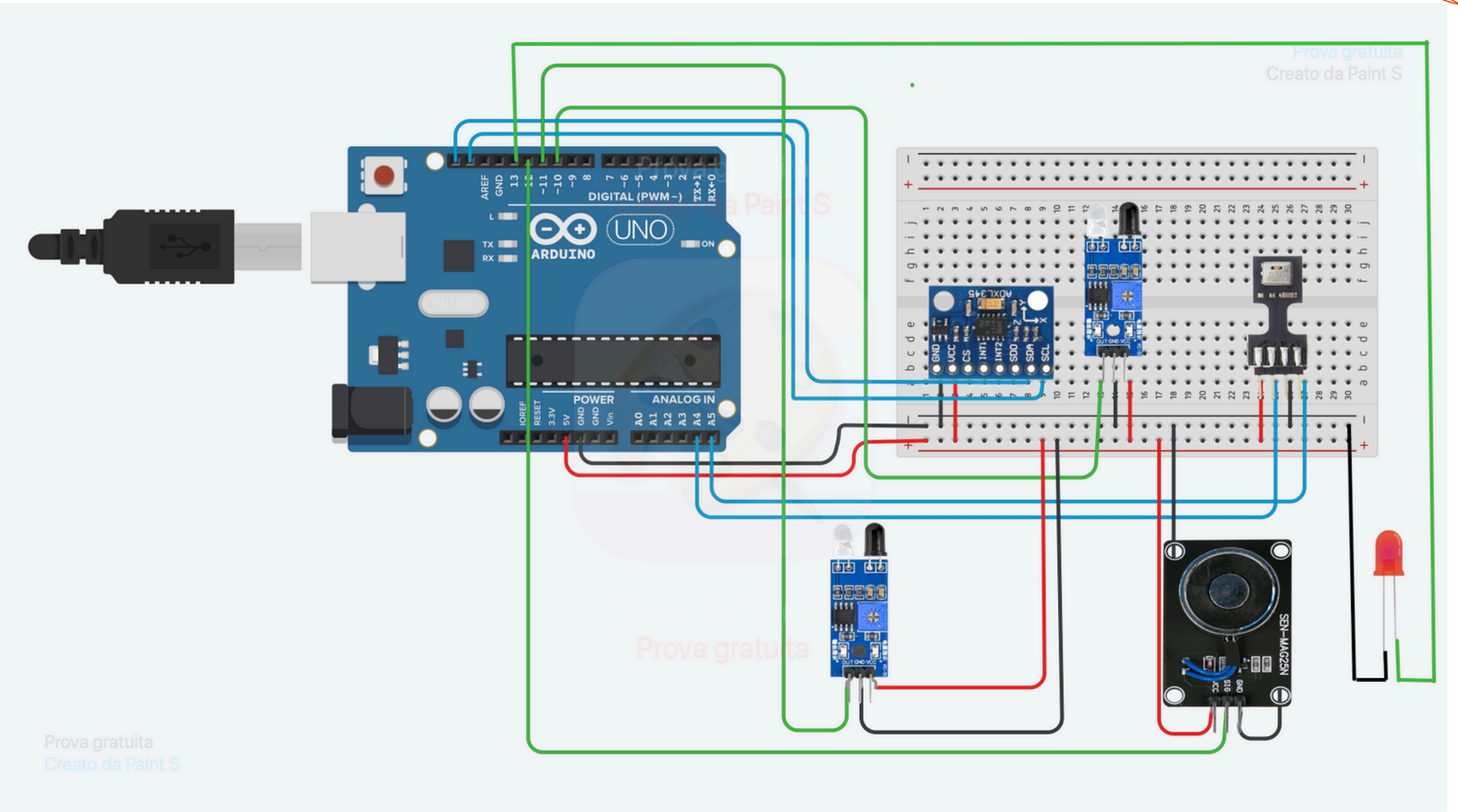
Sensors and Actuators

Acquisition

Description	Part Number	Protocol
Humidity and Temperature sensor	AHT25	I2C
IR-sensor	AZ-DELIVERY IR-ABSTAND SENSOR	Digital output
Accelerometer	ADXL345	I2C
Electromagnet	Joy-it SEN-MAG25N	Digital output
Led	Generic led	Digital output

SMASHBOX

Sensors and Actuators



Digital Output

I2C

VCC

GND

SMASHBOX

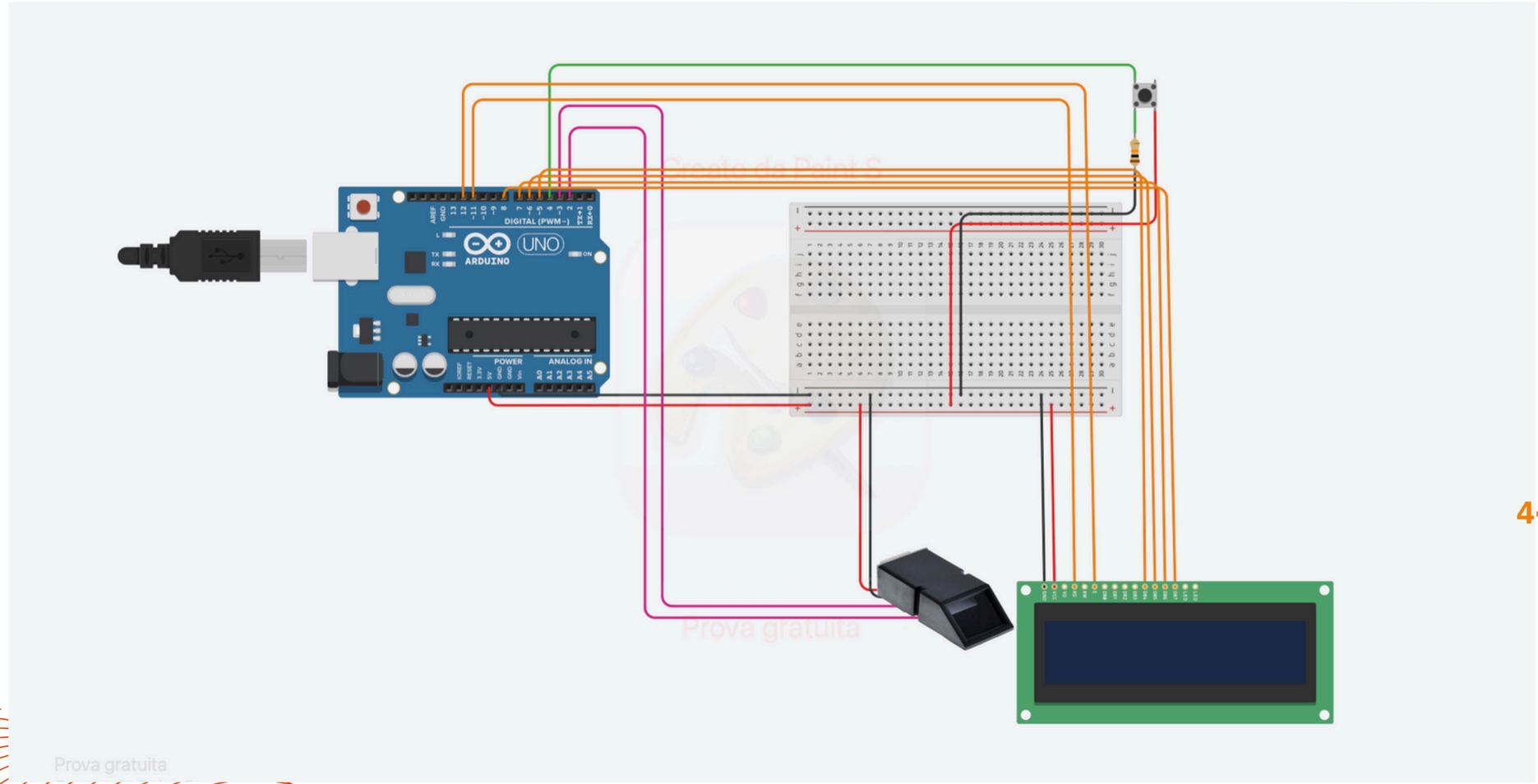
Sensors and Actuators

Central

Description	Part Number	Protocol
Fingerprint sensor	JM-101	UART
LCD display	ARCELI RP2040	4-bit parallel interface
Button	Generic button	Digital input

SMASHBOX

Sensors and Actuators



UART

Digital input

4-bit parallel interface

VCC

GND

SMASHBOX

Bridge

BRIDGE

The Bridge is a Python script that performs multiple functions:

1. It manages bidirectional communication between:

- **LOCAL (receives data from Arduino)**
- **CLOUD (sends data to the Cloud / receives commands)**

2. It uses a Parser to:

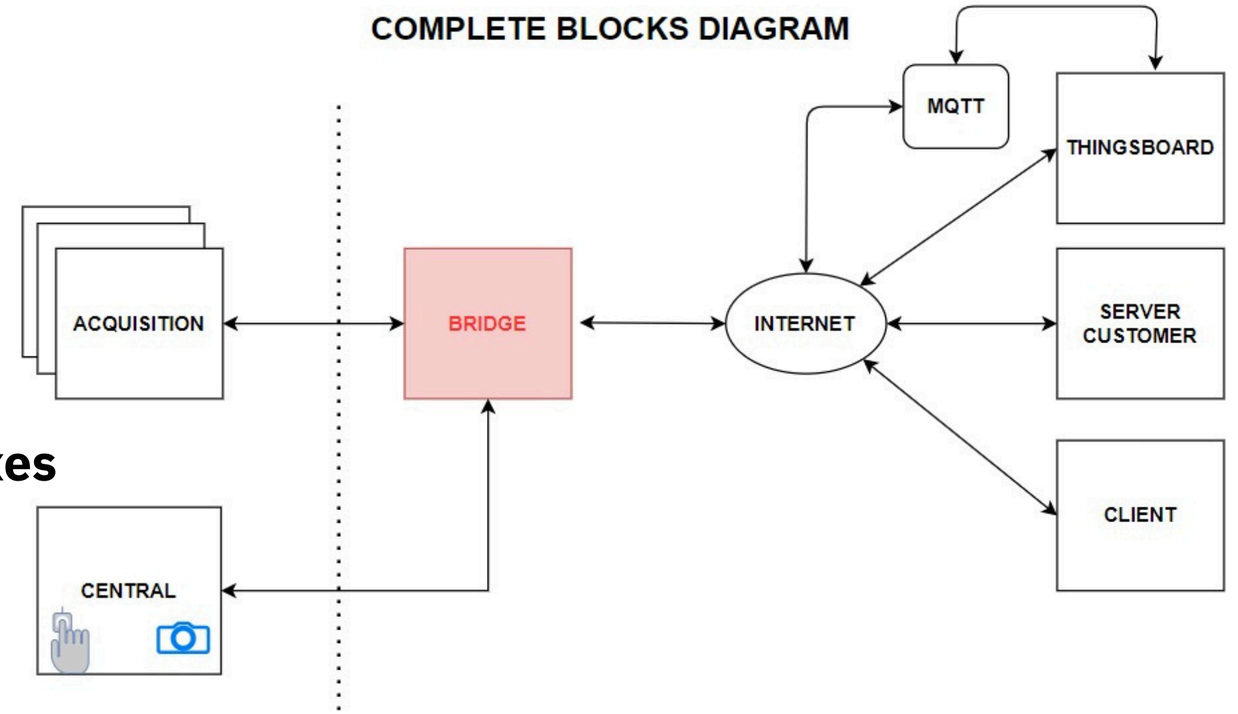
- **Unpack the data**
- **Perform MQTT updates (faster than HTTP)**

3. It functions as a central node between:

- **CENTRAL (control logic)**
- **Multiple ACQUISITIONS (sensor boxes)**

4. In case of a critical event (Safe Mode):

- **The Bridge propagates the command to all the boxes**



SMASHBOX

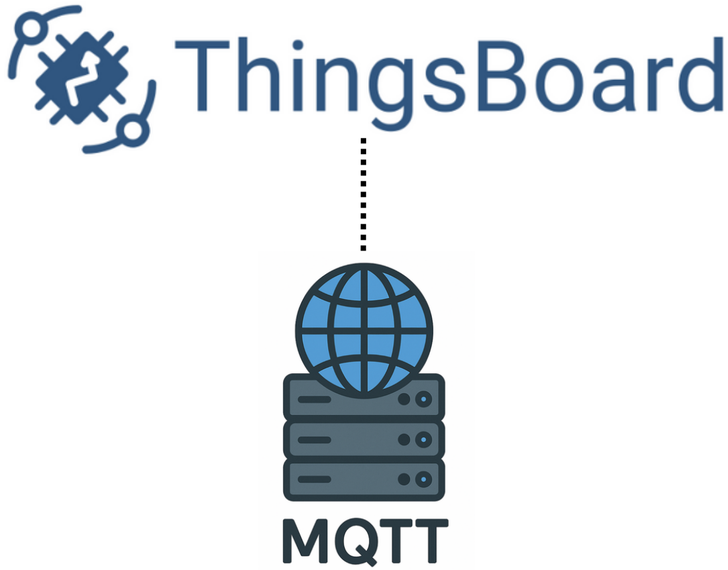
Cloud Platform

Thingsboard is the chosen Platform for Remote Monitoring

Main Features:

- **Representation of each box through a Digital Twin**
- **Real-time data visualization (charts, statuses)**
- **Automatic triggers in response to received events**
- **Manages the propagation of Safe Mode**

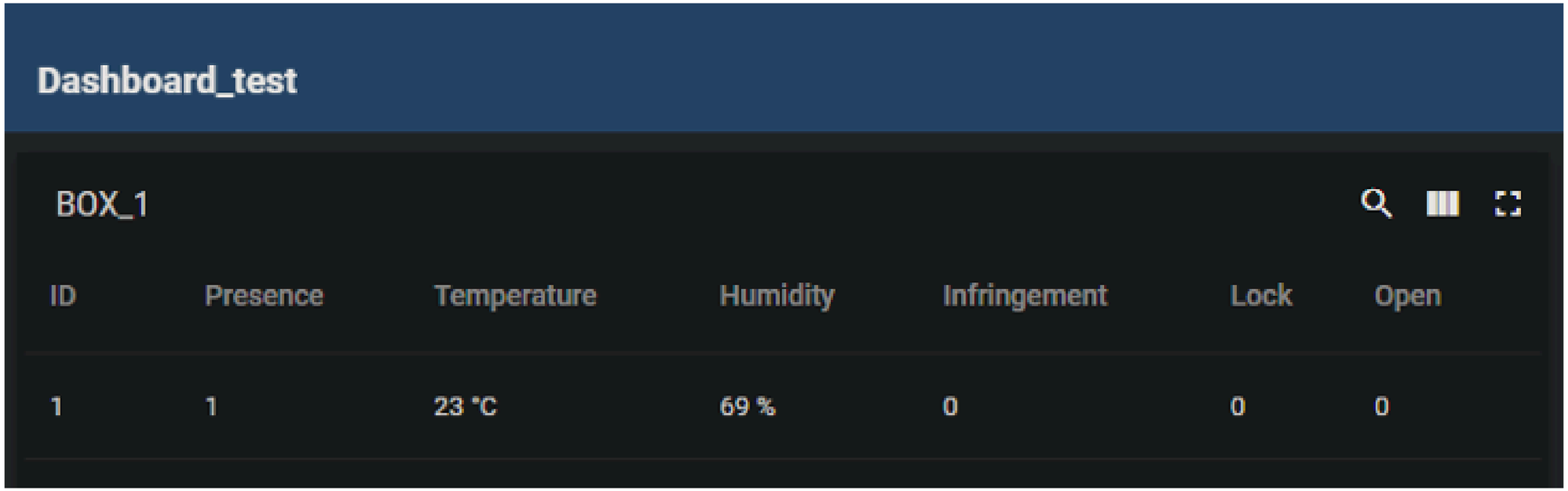
A server hosting an MQTT broker has been used to enable communication between BRIDGE and ThingsBoard.



Cloud Platform

Each device is configured with a unique access token for communication via the integrated MQTT broker
A custom dashboard is developed to:

- **Display data in real time**
- **Facilitate continuous monitoring of device status**



SMASHBOX

Cloud Platform

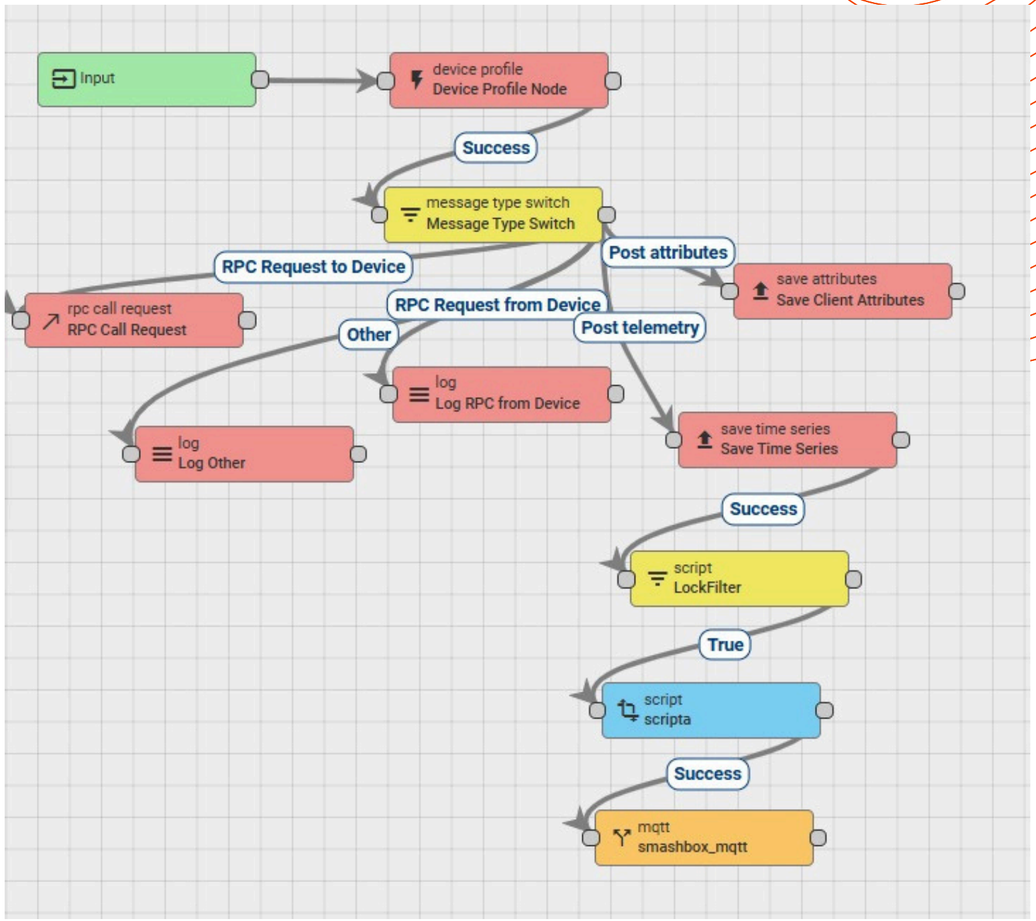
A Rule Chain was implemented to define trigger conditions based on predefined threshold violations

When an alarm condition is detected:

- **The Rule Chain generates an outgoing MQTT message**
- **The message is sent to MQTT broker**

The Bridge, subscribed to the broker:

- **Receives MQTT alarm messages**
- **Executes corresponding actions based on the notification**



SMASHBOX

System analysis (high level)

Startup Process:

- **Initialization: ACQs and CNTRL enter standby mode.**
- **Connection: Devices connect to the Bridge.**
- **Assignment: Bridge recognizes all the units.**
- **Enroll: Bridge enable central to start enroll fase.**
- **Identification: each boxes registered get an id.**

Normal Operation (IDLE Mode):

- **ACQs collect environmental & security data.**
- **Data is sent → Bridge → Server.**
- **Boxes can be unlocked via registered fingerprints.**

Alarm Mode (SAFE MODE):

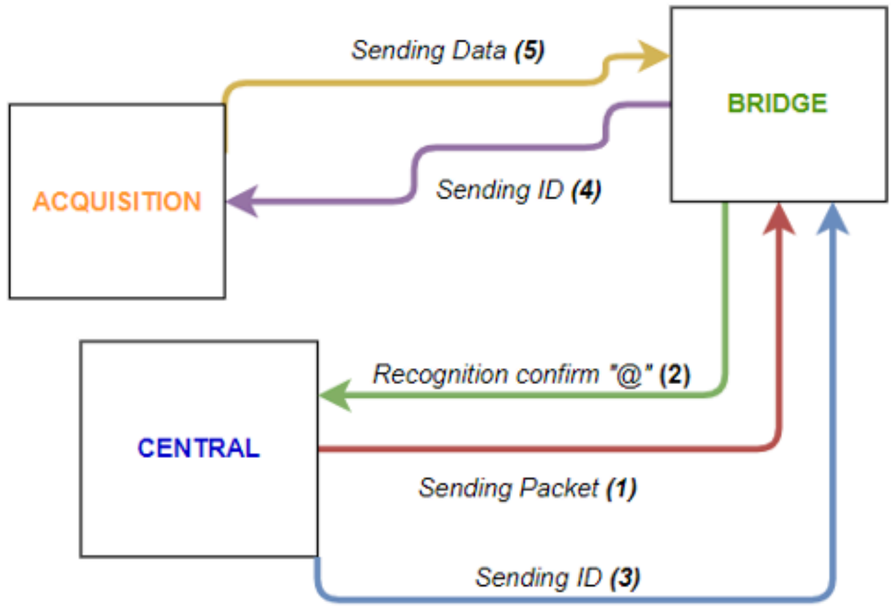
- **Triggered by:**
  - **Unauthorized opening**
  - **Physical shock**
  - **Abnormal temperature/humidity**
- **System response:**
  - **Local alerts (LEDs, locks)**
  - **Notification sent to Telegram Bot**

SMASHBOX

Technical analysis
First-Time Start Up Phase

Start Up phase is composted by 2 sub-phases:

- **Recognition →bridges identifies which serial port is connected to central and acquisitions**
- **Enroll → assignment of an id based on the registered fingerprint to a box**



1) CNTRL -> BRIDGE: 7-byte packet: 0x00
2): BRIDGE-> CNTRL : 1-char "@"
3) CNTRL -> BRIDGE: 2-byte pkt (header, data)
4) BRIDGE-> ACQ: 10-byte pkt (id)
5) ACQ-> BRIDGE 7-byte pkt (data acquisition)

SMASHBOX

Technical analysis

Acquisition protocol

After receiving its unique ID from the CNTRL, each ACQ continuously acquires data in real-time from sensors installed inside the boxes:

- **Temperature & Humidity**
- **Accelerometer**
- **Object presence**
- **Door open/close status**

At the end of each acquisition cycle, the ACQ sends a 7-byte data packet via serial to the BRIDGE. This loop runs periodically.

ID (1 Byte)	Presence (1 Byte)	Temperature (1 Byte)	Humidity (1 Byte)	Infraction (1 byte)	Lock (1 byte)	Open (1 byte)

SMASHBOX

Technical analysis
Acquisition protocol

Normal Mode:

- **The 7-byte packet contains the actual sensor data acquired during the cycle. The BRIDGE then forwards this data to the server.**

Safe Mode:

- **All 7 bytes are set to 0, except for the “Lock” field, which is set to 1.**
- **This packet is sent repeatedly until SAFE MODE is exited.**

	Id	Presence	Temperature	Humidity	Infraction	Lock	Open
normal mode	45	0	22	46	0	1	0
safe mode	0	0	0	0	0	1	0

SMASHBOX

Technical analysis
Central Protocol

CNTRL sends action requests to BRIDGE via a 2-byte ACTION_PACKET:

- **1 byte for action header (PACKET_IDLE, PACKET_CHECK, PACKET_ENROLL, PACKET_SAFE)**
- **1 byte for action data**

Three action modes:

1. **Enroll mode**
2. **Fingerprint Mode**
3. **Check Mode**

Enroll Mode	FIngerprint Mode	Check Mode
• Wait for double check fingerprint registration • Sends header PACKET_ENROLL + random ID • BRIDGE forwards a 10-byte packet to ACQ	• Listening state for new fingerprint input	• Triggered when a finger is placed • If matched, sends header PACKET_CHECK + recognized ID • Display shows possibly unlocked Box ID

SMASHBOX

Technical analysis

Safe Mode Protocol

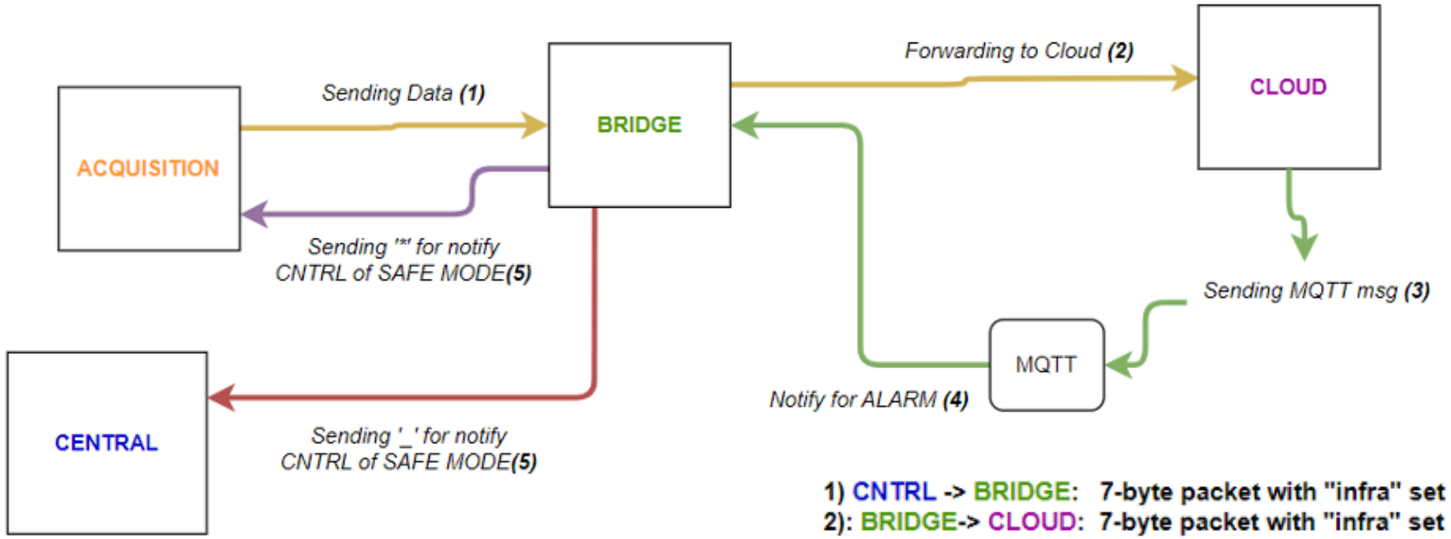
Mode triggered by the server.
When ACQ data reaches the server, it is processed in real time.

If any of the following conditions are detected:

- **Tampering (via accelerometer)**
- **Temperature or humidity exceed predefined thresholds**

Then:

- **A message is sent via MQTT broker to the BRIDGE**
- **The BRIDGE notifies the entire "local" subsystem: Both CNTRL and ACQ components**



1) CNTRL -> BRIDGE: 7-byte packet with "infra" set
2): BRIDGE-> CLOUD: 7-byte packet with "infra" set
3) CLOUD-> BRIDGE: MQTT msg
4) CLOUD-> BRIDGE: MQTT msg
5) BRIDGE-> CNTRL 1 char '_' to notify CNTRL
5) BRIDGE-> ACQ 1 char '*' to notify ACQ

Telegram notification

SMASHBOX

Future developments

SMASHBOX

Overview

Goal

Enhance hardware reliability, software flexibility, and overall system scalability for real-world product deployment

---

Areas of Focus:

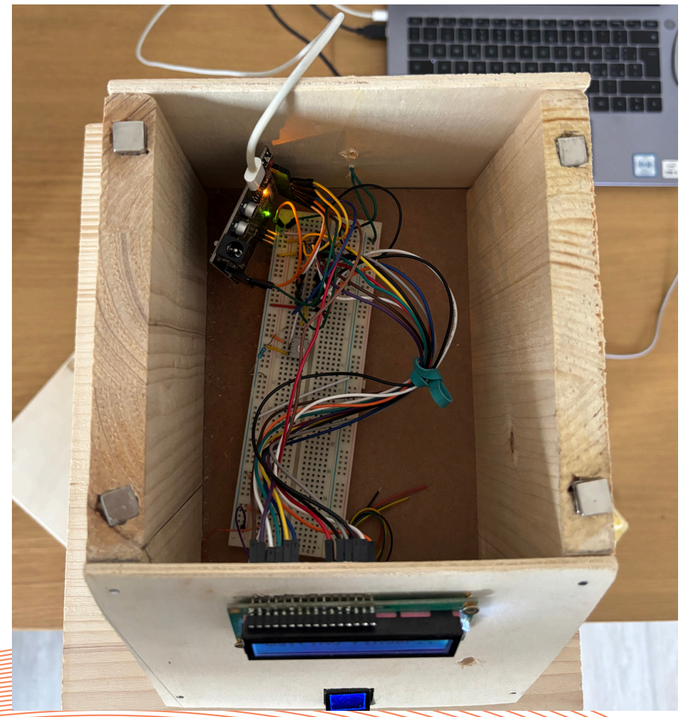
- Hardware upgrades
- Wireless communication
- Software refactoring and modularization

SMASHBOX

Hardware

From Prototype to Production

Current Situation:
Wired breadboard-based prototype using Arduino and bulky components.



Future Goals:

- **Transition from wired prototype to custom PCB to reduce size**
  - **Design via PCB CAD tools**
- **Replace Arduino with a dedicated microcontroller or ASIC**
  - **Compact size**
  - **Reduced cost and power consumption**
  - **Increased reliability for mass production**

Wireless Communication

Problem:

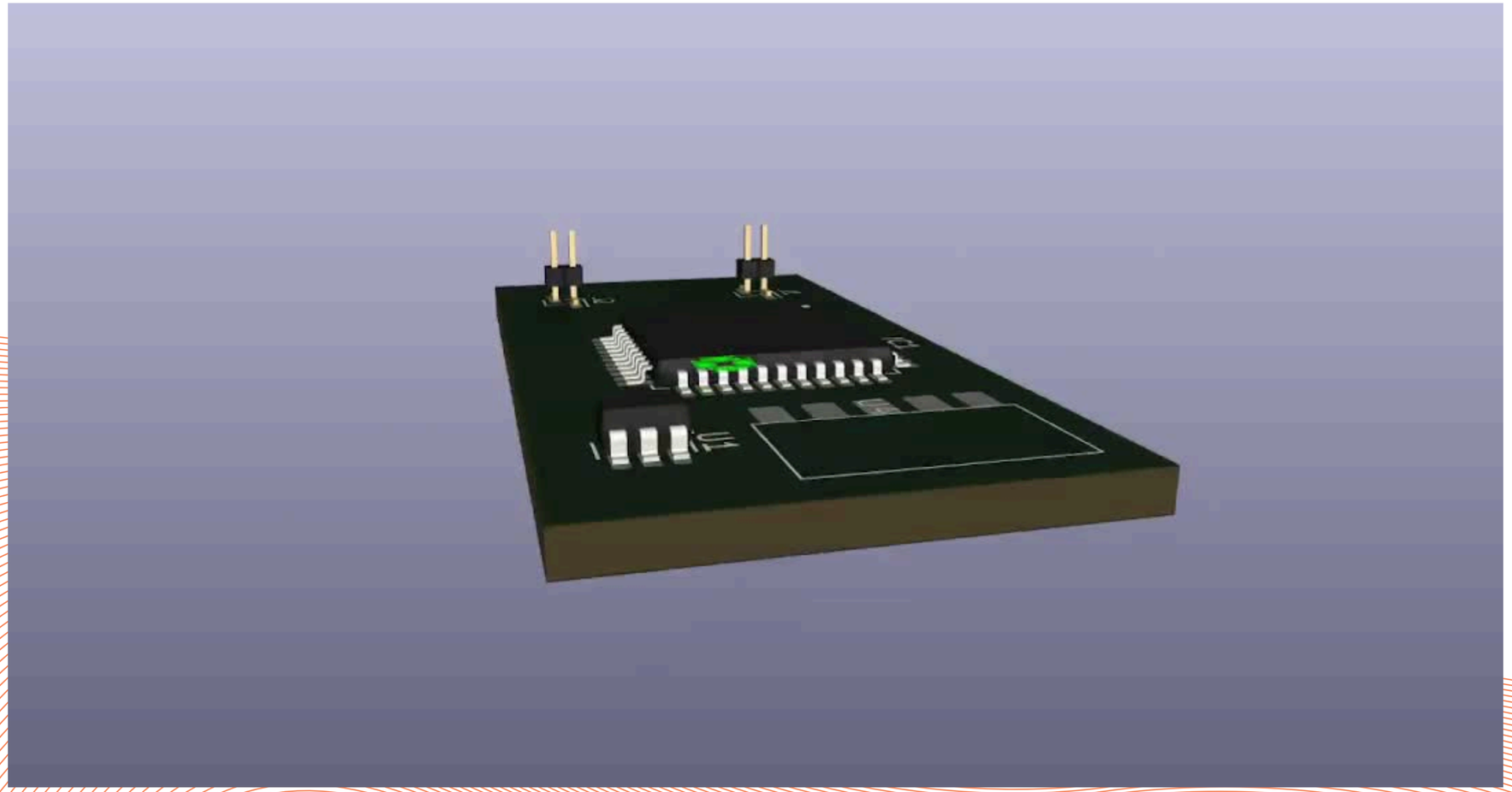
- **Current CNTRL-ACQ communication relies on physical wiring which are not viable in production scenarios.**

Future Implementation:

- **Integrate Wi-Fi transceivers for wireless data exchange between control and acquisition units**
- **Allows better software deployment and remote access**
- **Crucial for product usability in real-world environments**

SMASHBOX

CAD design



Software Architecture Refactoring (HAL-Based)

Objective:

- **Make the software platform-independent and more resilient to obsolescence.**

Key Actions:

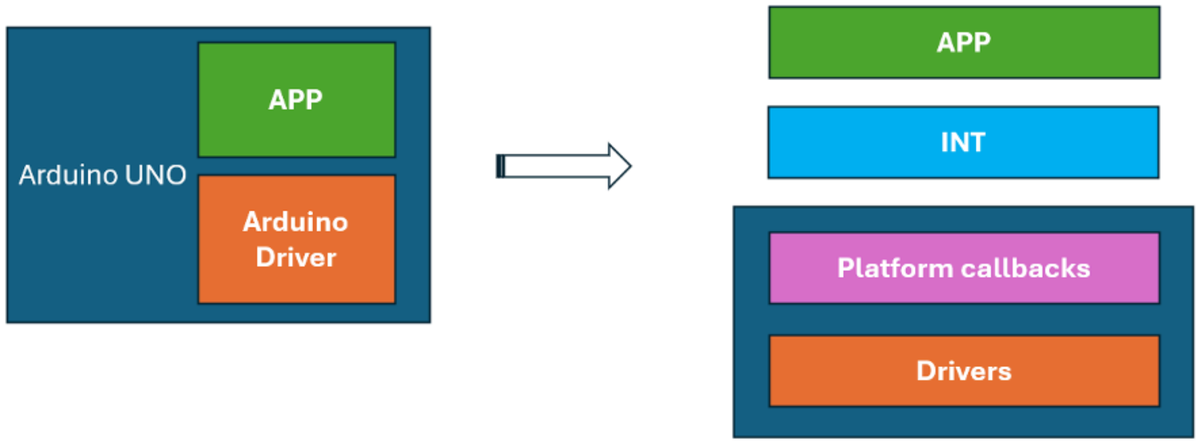
- **Refactoring code-base creating an Hardware Abstraction Layer (HAL)**
  - **Improve modularity and code reuse**
  - **Easier migration to new platforms**

- **Move from specific Arduino drivers to a multilayer architecture, composed by:**
  - **Dedicated layer for interfaces (INT)**
  - **Application layer which exploits interfaces layers (APP)**
  - **Low level drivers for specific platform**

Benefits of HAL Architecture

Why HAL?

- **Shields the application from hardware devices changes (transceivers, sensors)**
- **Minimizes code porting when migrating to new microcontrollers**
- **Supports long-term maintainability**
- **Enhances software portability and scalability**



In summary:
If we assume that Arduino (or any microcontroller unit) is discontinued, only the low-level driver layer would need to be completely rewritten. The application logic would remain unchanged.

Old vs New apporach

Old Approach

```
void loop ()
{
    /* acquire data via I2C */
     data = Arduino_I2C_read ();

    /* elaborate data */
     // some logic depending on APP

    /* sending via UART */
     Arduino_UART_send(&data);
}
```

```
int Arduino_I2C_read ()
{
        //…
}
```

```
int Arduino_UART_send()
{
        //…
}
```

Old vs New apporach

New Approach

APP

```
int main()
{
    /* acquire data via I2C */
     data = I2C_read ();

    /* elaborate data */
     // some logic depending on APP

    /* sending via UART */
    UART_send(&data);
}
```

INTERFACES

```
int I2C_read ();

int UART_send();
```

```
class Platform_I2C : I2C_Interface
{
        int I2C_read ()
        {
           // Platform Low Level Implementation
        }
}

class Platform_UART : UART_Interface
{
        int UART_send()
        {
         // Platform Low Level Implementation
        }
}
```

ONLY THIS CODE MUST BE WRITTEN WITH A NEW PLATFORM

SMASHBOX

Thanks for your attention

GitHub

Simone Bugo
Francesco Marzo