RAFIK HARIRI UNIVERSITY

SMART BABY CRIB

Done by

AHMAD ACHAJI

AHMAD HAMMOUDEH

SLEIMAN SMAILI

ZIAD ZOK

Submitted to

DR. Ahmad Koubeissi

This Senior Project is submitted in Partial Fulfillment of the Requirements of the BE Degree of Mechatronics Engineering Major of the College of Engineering at Rafik Hariri University

MECHREF-LEBANON

April 2025

Ahmad Achaji (20210313)

Ahmad Hammoudeh (20210482)

Sleiman Smaili (20210629)

Ziad Zok (20210281)

ACKNOWLEDGMENT

This project is the culmination of countless hours of collaboration, dedication, and a shared commitment to innovation in the development of our Smart Baby Crib. It stands as a reflection of the collective efforts of a passionate and determined team striving to bring technological advancements into the world of infant care.

First and foremost, we would like to express our profound gratitude to Dr. Ahmad Koubeissi, whose unwavering support, insightful feedback, and continuous encouragement were vital throughout every stage of this project. His guidance not only propelled our progress but also helped us maintain focus and purpose from inception to final execution.

We would also like to extend our sincere appreciation to the faculty of the Mechatronics Engineering Department at Rafik Hariri University, whose foundational teaching and support laid the groundwork for our academic and professional development. Their role in shaping our engineering mindset cannot be overstated.

To our friends, who offered both moral support and practical advice during the challenging phases, thank you for your uplifting words and clever solutions that helped us navigate tough situations with confidence.

Finally, and most importantly, we owe our deepest thanks to our families. Their boundless patience, love, and sacrifices, whether emotional, physical, or financial, were the pillars that carried us through this journey. Without their enduring belief in our potential, this achievement would not have been possible.

This Smart Baby Crib project is as much theirs as it is ours.

ABSTRACT

This report presents the comprehensive design and development of a Smart Baby Crib, an innovative system aimed at enhancing infant care by integrating health monitoring, environmental control, and automated soothing mechanisms. The project scope involves the creation of a multifunctional crib that leverages IoT connectivity, real-time sensing, and embedded systems to ensure the baby’s comfort and safety while offering caregivers convenient remote access and control.

The Smart Baby Crib is equipped with vital health monitoring features, including temperature, heart rate, and oxygen level sensors, all wirelessly transmitted via a wearable bracelet. The system includes both manual and automatic operation modes, with automated rocking, music playback, and lighting control triggered by real-time sound detection and sensor feedback. Additionally, a heating and cooling subsystem is implemented using Peltier cells to regulate mattress temperature, ensuring a consistent sleep environment.

This report explores the mechanical, electrical, and embedded software design aspects of the crib, emphasizing safety, affordability, and user-friendliness. The project also addresses challenges related to data privacy, long-term operation, and system reliability, delivering a practical solution for modern-day parenting.

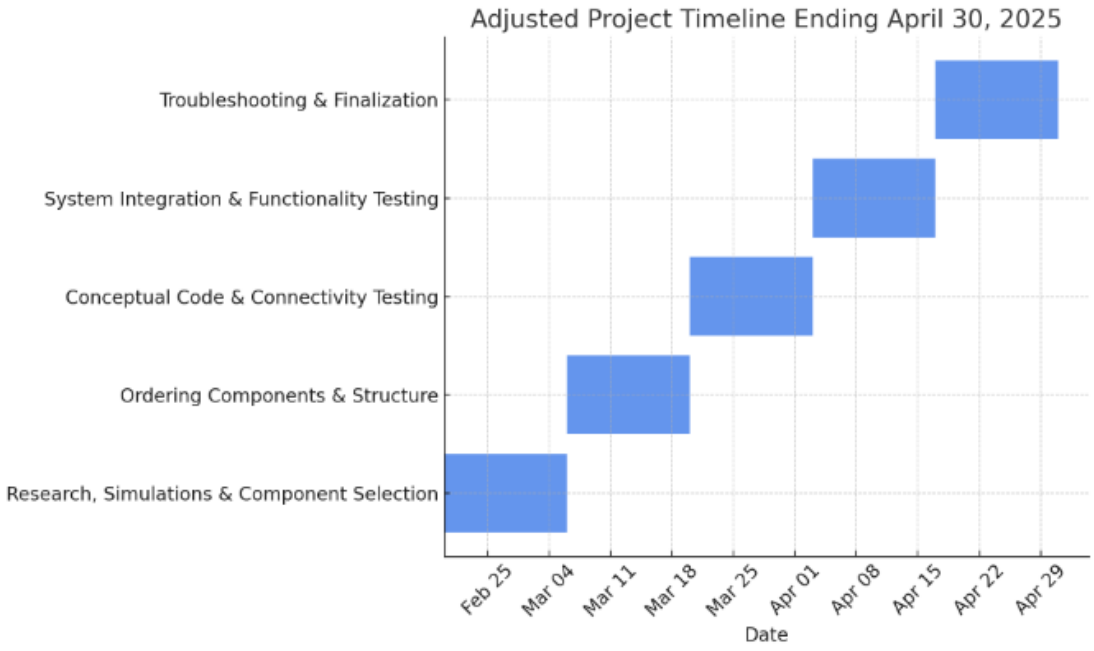
TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATION

TIME PLAN



Phase	Start Date	End Date
Research, Simulations & Component Selection	Feb 20, 2025	Mar 5, 2025
Ordering Components & Structure	Mar 6, 2025	Mar 19, 2025
Conceptual Code & Connectivity Testing	Mar 20, 2025	Apr 2, 2025
System Integration & Functionality Testing	Apr 3, 2025	Apr 16, 2025
Troubleshooting & Finalization	Apr 17, 2025	Apr 30, 2025

INTRODUCTION

1.1 Background Information

The evolution of baby monitoring devices and smart cribs reflects significant technological advancements aimed at enhancing infant safety and parental convenience. This section delves into the historical context, key innovations, and market trends that have shaped these technologies.

Historical Evolution

The first baby monitor, the Zenith Radio Nurse, was introduced in 1937. Developed by Eugene F. McDonald and designed by Isamu Noguchi, this device allowed parents to listen to their infants remotely, marking a pivotal moment in childcare technology (HelloBaby, n.d.). In 1997, Cynthia Altenhofen invented the first baby monitor with playback and two-way communication, enabling parents to record and play soothing messages to their babies (Back Then History, n.d.).

Key Innovations

The 21st century has witnessed the emergence of smart cribs, integrating Internet of Things (IoT) technology to offer features such as real-time monitoring, automated soothing mechanisms, and environmental controls. For instance, the SNOO Smart Sleeper, developed by Dr. Harvey Karp, utilizes responsive technology to mimic womb-like sensations, promoting better sleep for infants (National Center for Biotechnology Information, 2023). Additional advancements include cry detection systems and mobile applications for real-time updates and control.

Market Trends

The smart crib market has experienced substantial growth, with projections indicating a rise from USD 316.6 million in 2023 to approximately USD 641.7 million by 2034, driven by parental demand for enhanced safety and monitoring (Transparency Market Research, n.d.). Despite these advancements, challenges persist in affordability, data privacy, and the need for comprehensive, user-friendly designs that cater to a broader demographic. aimed at enhancing infant safety and parental convenience.

The first baby monitor, the Zenith Radio Nurse, was introduced in 1937. Developed by Eugene F. McDonald and designed by Isamu Noguchi, this device allowed parents to listen to their

infants remotely, marking a pivotal moment in childcare technology (HelloBaby, n.d.). In 1997, Cynthia Altenhofen invented the first baby monitor with playback and two-way communication, enabling parents to record and play soothing messages to their babies (Back Then History, n.d.).

The 21st century has witnessed the emergence of smart cribs, integrating Internet of Things (IoT) technology to offer features such as real-time monitoring, automated soothing mechanisms, and environmental controls. For instance, the SNOO Smart Sleeper, developed by Dr. Harvey Karp, utilizes responsive technology to mimic womb-like sensations, promoting better sleep for infants (National Center for Biotechnology Information, 2023). The smart crib market has experienced substantial growth, with projections indicating a rise from USD 316.6 million in 2023 to approximately USD 641.7 million by 2034, driven by parental demand for enhanced safety and monitoring (Transparency Market Research, n.d.).

Despite these advancements, challenges persist in affordability, data privacy, and the need for comprehensive, user-friendly designs that cater to a broader demographic.

1.2 Literature Review

Introduction

Automated systems are a prominent focus of modern engineering, aiming to minimize human interference in various domains. Within this context, Simultaneous Localization and Mapping (SLAM) systems have been extensively researched, laying the groundwork for numerous advancements. Similarly, the development of smart cribs has gained traction, combining mechatronic systems, baby monitoring technologies, and IoT applications to improve childcare.

Historical Development

A comprehensive exploration of relevant aspects reveals numerous dimensions influencing smart crib design. Historical developments trace back to early innovations in baby monitoring during the 20th century, when devices transitioned from simple analog systems to modern IoT-based designs. For instance, the introduction of the Zenith Radio Nurse in 1937 marked the beginning of remote auditory monitoring, while the 1997 invention of the two-way playback monitor by Cynthia Altenhofen enhanced interaction between parents and infants. In the 2000s, the rise of IoT allowed devices to transmit data in real-time, exemplified by products like the SNOO Smart

Sleeper, which integrates automated soothing features and environmental control mechanisms. This evolution demonstrates a continuous enhancement in functionality, connectivity, and user-centric design. The first baby monitor, the Zenith Radio Nurse, was introduced in 1937, developed by Eugene F. McDonald and designed by Isamu Noguchi (HelloBaby, n.d.). In the 2000s, the rise of IoT, popularized by figures like Kevin Ashton, enabled real-time data handling, culminating in sophisticated smart crib solutions like the Snoo Smart Sleeper in the 2010s. Ashton (2009) described IoT as a system where the Internet connects to the physical world via ubiquitous sensors.

Key Mechanisms and Technologies

Advancements in smart crib design emphasize key mechanisms such as:

- **Health Monitoring Sensors**: Integration of heartbeat, oxygen, and temperature sensors.
- **Cry Detection Systems**: Using microphones and machine learning to identify and respond to infant cries.
- **Automated Soothing**: Employing actuators for rocking motions.
- **IoT-Enabled Connectivity**: Real-time monitoring and remote control via mobile applications.
- **Comparative Analysis:**

  The expanded table compares more products available in the market, showcasing the strengths and limitations of each alongside the proposed smart crib. This approach highlights how the proposed crib fills gaps in affordability, features, and usability.

  Table 1 below demonstrates how the proposed crib integrates advanced features at an accessible price point, addressing gaps in affordability and functionality.

FEATURES	BASIC CRIBS	SNOO SMART SLEEPER	CRADLE WISE SMART CRIB	GRACO SENSE 2 SNOOZE	4 MOMS MAMAROO SLEEP BASSINET	PROPOSED CRIB
Cry Detection	No	Yes	Yes	Yes	Yes	Yes
Health Monitoring	No	No	Yes	No	No	Yes
Motion Actuators	Manual	Auto	Auto	Auto	Customizable	Auto
Mobile App Connectivity	No	Yes	Yes	No	Yes	Yes
Temperature control	No	No	No	No	No	Yes
Camera	No	Yes	Yes	No	No	Yes
Cost ($)	<200$	~1700$	~2000$	~300$	~350$	~400$

Table 1: Comparison between different Smart Crib products in the market

The figures below show the products used for the comparison

Figure 1: Cradle-Wise Smart Bassinet

Figure 2: SNOO Smart Sleeper Bassinet

Figure 3:Graco Sense2Snooze Bassinet

Figure 4: 4moms MamaRoo Sleep Bassinet

Current Challenges

Despite significant progress, challenges remain in areas such as:

- **Real-World Usability**: Limited focus on diverse household scenarios.
- **Affordability**: High costs limit accessibility to affluent families.
- **Data Privacy**: Concerns over IoT-connected devices storing sensitive information.

By addressing these gaps, the project seeks to combine functionality and accessibility in a comprehensive smart crib design., aiming to minimize human interference in various domains. Within this context, Simultaneous Localization and Mapping (SLAM) systems have been extensively researched, laying the groundwork for numerous advancements. Similarly, the development of smart cribs has gained traction, combining mechatronic systems, baby monitoring technologies, and IoT applications to improve childcare.

A comprehensive exploration of relevant aspects and subtopics reveals numerous dimensions influencing smart crib design. Key considerations include mechanical and electrical design, actuation systems with Arduino-based control algorithms, health monitoring sensors, and IoT-enabled real-time data management using platforms like Google Firebase. Further advancements incorporate cry detection for initiating soothing actions, temperature regulation mechanisms, and mobile app development for caregiver interaction.

Historical developments trace back to early innovations in baby monitoring during the 20th century. The first baby monitor, the Zenith Radio Nurse, was introduced in 1937, developed by Eugene F. McDonald and designed by Isamu Noguchi (HelloBaby, n.d.). The rise of IoT in the 2000s, popularized by figures like Kevin Ashton, enabled real-time data handling, culminating in sophisticated smart crib solutions like the Snoo Smart Sleeper in the 2010s. Ashton (2009) described the Internet of Things as a system where the Internet is connected to the physical world via ubiquitous sensors. Current trends explore AI integration, aiming for affordable, personalized childcare solutions. Landmark studies by pioneers like Dr. Harvey Karp emphasize soothing techniques and safe sleep environments. Karp's "5 S's" method—swaddling, side/stomach position, shushing, swinging, and sucking—has been widely adopted for calming infants (Healthline, 2019).

Despite significant progress, challenges remain in areas such as real-world usability, affordability, and privacy. The table below outlines the comparison of current systems and highlights how our project addresses these challenges:

CHALLENGES	EXISTING SOLUTIONS	PROPOSED SMART CRIB
Real-World Usability	Limited adaptability to diverse environments	Versatile design suitable for various households
Affordability	High-end systems priced over $1,500	Estimated cost below $400
Data Privacy	Insufficient encryption mechanisms	Implementation of robust data encryption

Table 2: Challenges and solutions of smart cribs

The proposed solutions aim to make the smart crib more accessible and secure, enhancing its appeal and functionality for a broader audience. While existing studies emphasize safety and automation, debates persist regarding wearable versus non-wearable sensors and balancing cost-effectiveness with advanced features. By addressing these gaps, this project seeks to combine functionality and accessibility in a single, comprehensive smart crib design.

1.3 Project Motivation

Although smart cribs have evolved substantially, gaps persist in delivering a fully integrated, user-centric solution. Current designs often focus on isolated features such as soothing mechanisms or health monitoring, failing to provide holistic systems. Additionally, the high costs associated with these advanced cribs limit accessibility for middle- to low-income families. Privacy concerns surrounding IoT-enabled baby monitoring also remain inadequately addressed.

The motivation behind this project lies in bridging these gaps by developing a versatile and affordable smart crib solution. By integrating monitoring, soothing, and environmental control functionalities, this project addresses pressing challenges in real-world usability, cost-

effectiveness, and user data privacy. This endeavor aims to enhance childcare accessibility while maintaining high standards of safety and comfort.

1.4 Project Objective

The primary objective of this project is to design and develop an affordable smart crib that incorporates advanced monitoring, soothing, and environmental control systems. The proposed crib will integrate features such as heart rate and oxygen monitoring, cry detection, temperature regulation, and real-time IoT connectivity. Utilizing efficient power management and adaptive actuation mechanisms, the project strives to balance functionality and cost. Furthermore, incorporating secure data protocols aims to alleviate privacy concerns while ensuring reliability and caregiver trust.

This project's goal is to create a user-friendly and highly functional smart crib, setting a benchmark in cost-effective childcare technology.

1.5 Design Constraints

The design constraints of this smart crib project are categorized as technical and non-technical.

Technical Constraints:

- **Power Efficiency**: Actuators and systems must consume minimal power to ensure long battery life and lower operational costs.
- **Weight and Size**: The crib's weight should facilitate easy mobility while maintaining structural stability.
- **Sensor Reliability**: Vital signs sensors for heart rate, oxygen, and temperature must provide accurate, real-time data.
- **Communication Protocols**: Ensure seamless integration of Wi-Fi and Bluetooth for stable IoT connectivity.

- **Safety**: Design mechanisms to shield the baby from moving parts, electrical hazards, and overheating risks.

Non-Technical Constraints:

- **Cost-Effectiveness**: Affordable materials and manufacturing processes to reduce overall cost.
- **Privacy and Security**: Implementation of encryption protocols for safeguarding caregiver and infant data.
- **Environmental Impact**: Use of sustainable, eco-friendly materials for mattress covers and crib walls.

These constraints will guide the design process and ensure the final prototype's compliance with both technical excellence and social responsibility.

CHAPTER 2 DESIGN CONCEPTS

2.1 Design Iteration 1

First iteration for an up and down mechanism consists of a custom disk cam mechanism for the up and Down motion. But it has multiple problems, 1 risk of slipping, 2 low reliabilities, 3 can't lift high weights. The picture below shows the mechanism design.

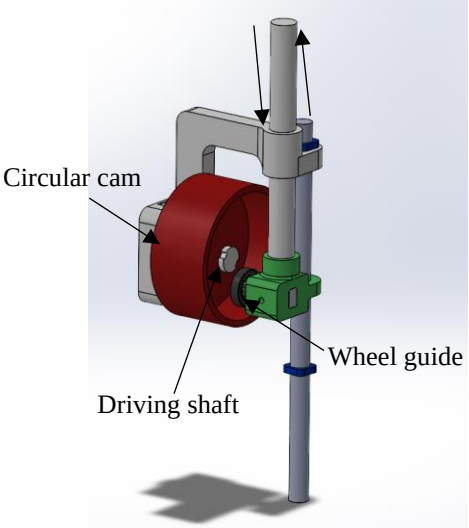


Figure 5: iteration 1 design

2.2 Design Iteration 2

In the second iteration we tried to create a 4-bar mechanism with guides to create a linear motion in the vertical orientation, the problem in this design is the multiple failure points/ weak points, and no control over the position of the platform due to gravity. The picture below shows the mechanism design.

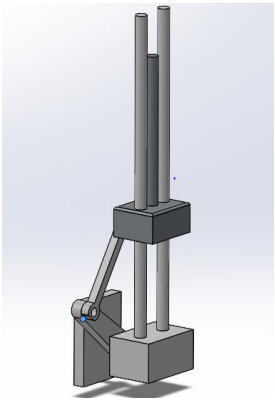


Figure 6: Iteration 2 design

2.3 Design Iteration 3

In the design we have 2 mechanisms, the first one is a standard screw-nut mechanism responsible for the up and down motion, and the other is a simple bracket and shaft for a motor to create a swinging / rocking motion. As shown in the figures below.

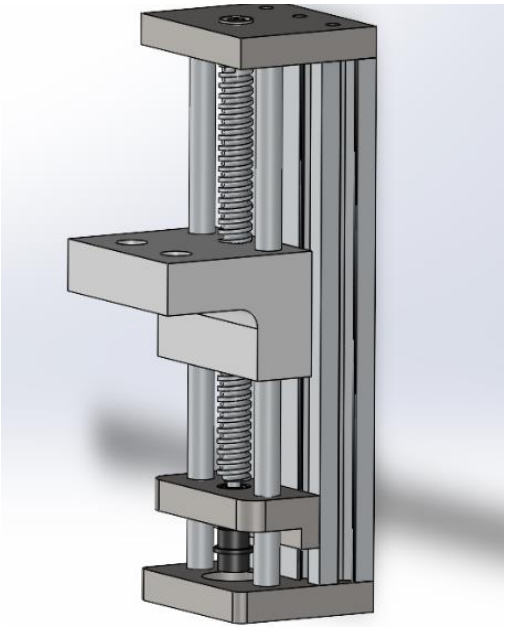


Figure 7: Screw-nut mechanism

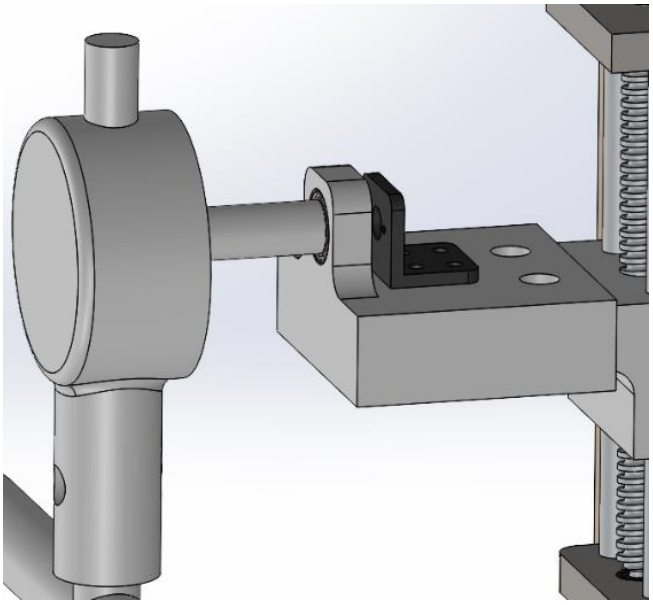


Figure 8: Motor bracket and shaft

2.4 Design 3 Analysis

Analysis:

These mechanisms provide the 2-motion type in the system, and they need to be reliable ensuring smooth motion and safety for the baby. Using SolidWorks analysis we checked the displacement, von misses' stresses and safety factor, on 3 different materials: 1-ABS, 2- 6061 Aluminum alloy, and 3- A36 Steel.

Motor Shaft

Abs is a widely used plastic used in injection molding and in 3d printing, although the injection molding ABS parts are usually stronger and have 25-30MPA yield strength, 3D printed ABS is weaker due to the imperfections in the printing process and layer adhesion, and due to the layers that are considered weak points and can be snapped if not oriented correctly, so we will assume that 3D Printed ABS has a max yield strength of 15MPA with good quality print and adequate layer position. The figures show analysis of the Motor shaft in ABS.

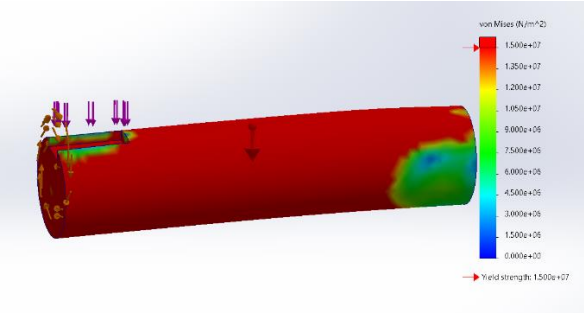


Figure 9: Von Misses analysis on ABS motor shaft

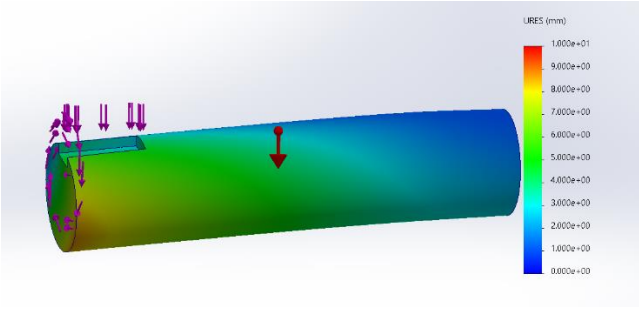


Figure 10: Displacement analysis on ABS motor shaft

According to von Misses' analysis the ABS shaft cannot Support the load and undergoes high displacement and eventually yielding.

Like ABS we tried to use 6061 Aluminum alloy and conducted the analysis, but it was also not strong enough and caused failure. The pictures below visualize the analysis of this part using SolidWorks.

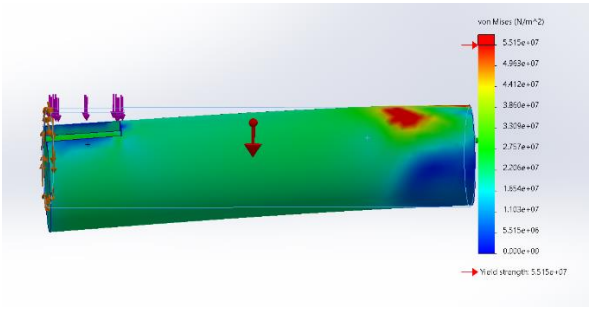


Figure 11: Von Misses analysis on "6061" motor shaft

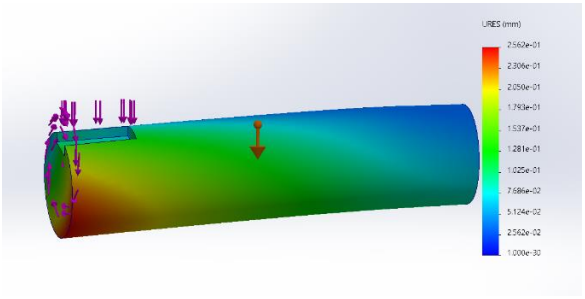


Figure 12: Displacement analysis on "6061" motor shaft

Although displacement is small there are areas of the design that exceeded the max yield strength and can cause failure anytime.

Finally, we Tried A36 Carbon steel and according to the analysis it is suitable and safe to use as a motor shaft, but for more safety it is recommended to change the design of the mechanism to ensure a higher safety factor. The figures show the solid works analysis of the shaft, and Safety factor analysis around the whole design to check if high stresses areas are at risk of failure.

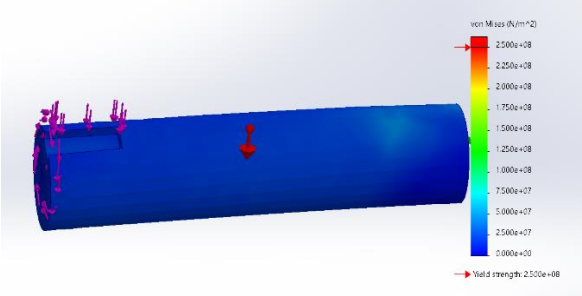


Figure 13: Von Misses analysis on "A36" motor shaft

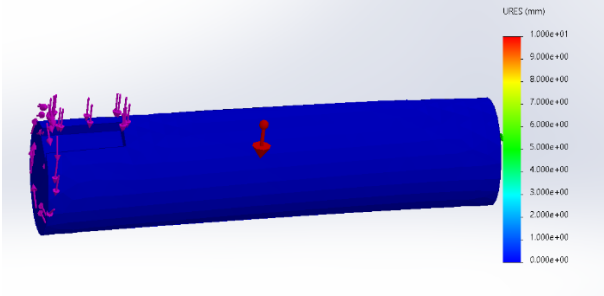


Figure 14: Von Misses analysis on "A36" motor shaft

Since the displacement was negligible of less than 0.5mm, and stress was under maximum yield strength by a lot we conducted a Factor of safety analysis to see how much safety factor we have using this A36 carbon steel material as a motor shaft. The figures below show the SF distribution and strain analysis across the shaft.

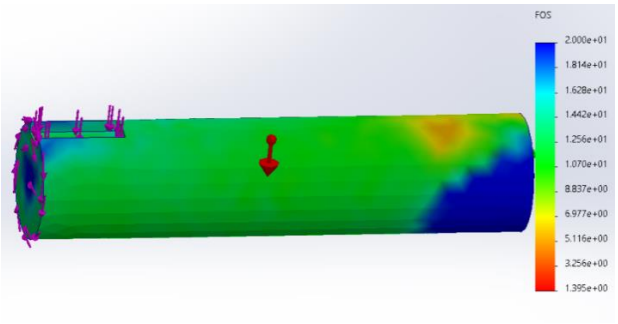


Figure 15:Safety factor analysis on "A36" motor shaft

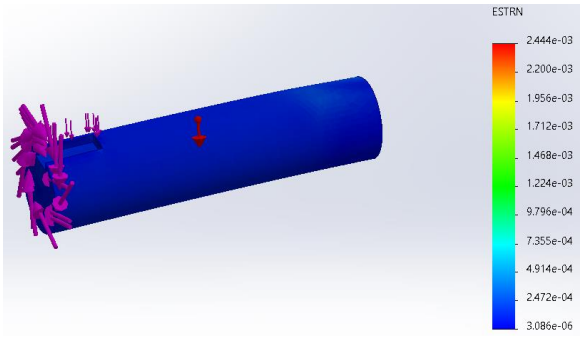


Figure 16: Strain analysis on "A36" motor shaft

We can see that in the high stresses area the factor of safety is still acceptable with the lowest value of 3 and higher safety factors across the shaft.

Screw-nut-platform

The screw nut platform is what connecting the whole bed frame into the up and down mechanism and the wooden base, so it must be strong enough with high factor of safety. The figures below show the displacement, factor of safety, Strain and von misses' analysis of the platform und the full 200n load of the crib.

ABS

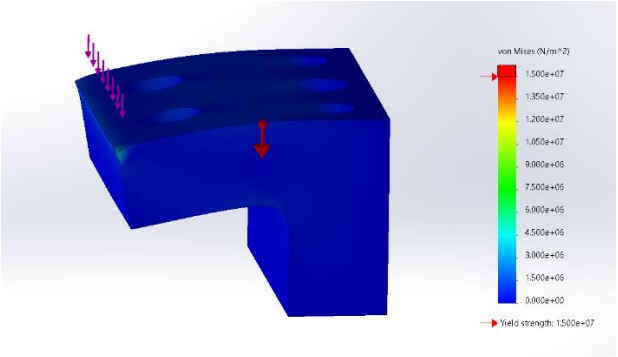


Figure 17: Von Misses analysis on "ABS" Platform

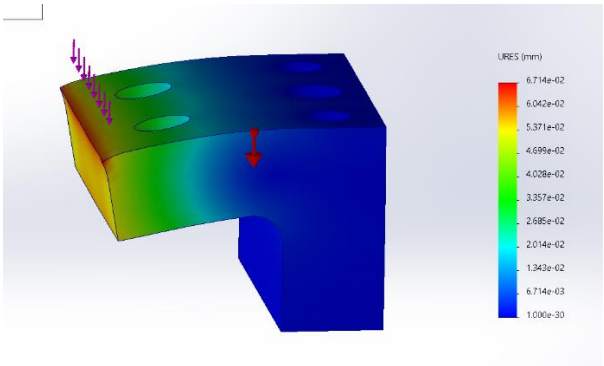


Figure 18: Von Misses analysis on "ABS" Platform

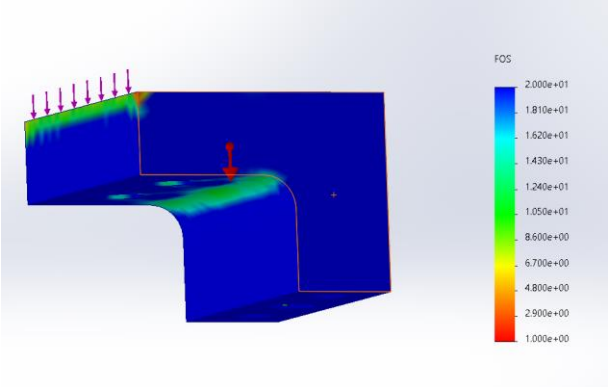


Figure 19: FOS analysis on "ABS" Platform

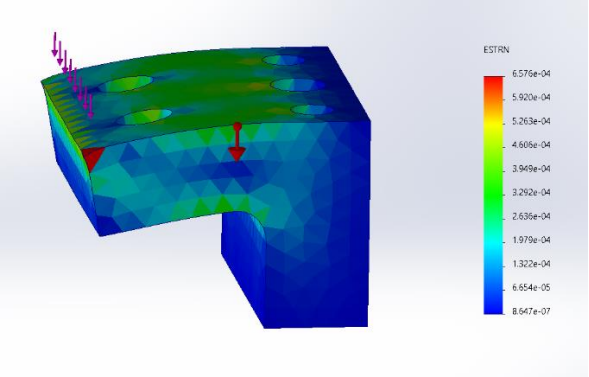


Figure 20: Strain analysis on "ABS" Platform

The analysis shows that abs are safe enough to use in our design because of the high factor of safety around 10-20 (avg) and very small displacement around 0.0671mm maximum.

6061 ALUMINIUM ALLOY

For increased safety and since this part is moving all the time and needs increased wear resistance we tried to use "6061 Aluminum Alloy" because its easy machine and cheap while maintaining good material properties. The figures below show the analysis in SW.

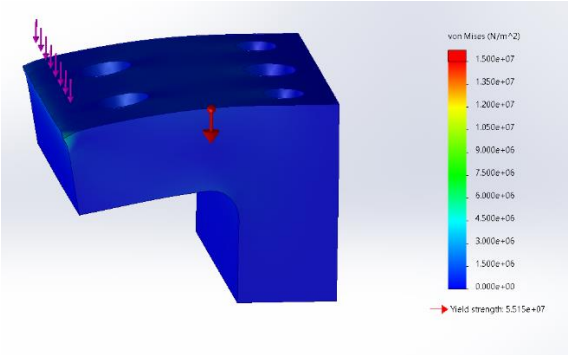


Figure 21: Von Misses analysis on "6061" Platform

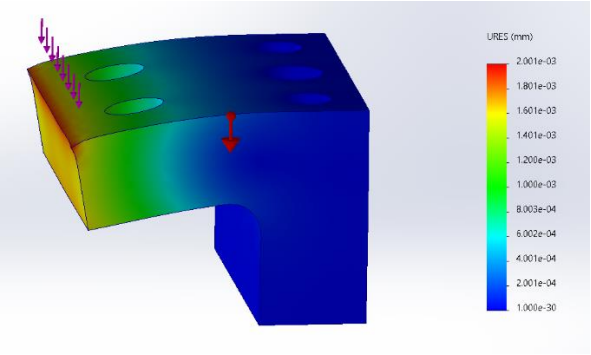


Figure 22: Dis analysis on "6061" Platform

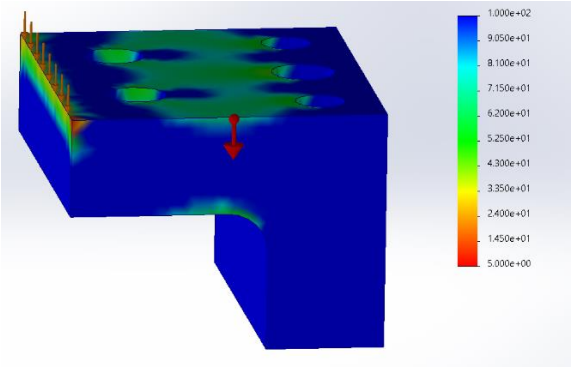


Figure 23: FOS analysis on "6061" Platform

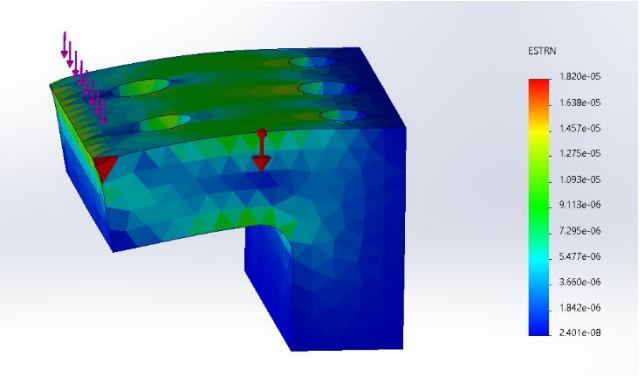


Figure 24: Strain analysis on "6061" Platform

Using Aluminum 6061 instead of ABS increased the minimum factor of safety from 3 to around 9 and decreased the displacement from 0.0671 to 0.00200. So, 6061 aluminum is the better choice for this part.

Design Weaknesses:

In conclusion, although this design is safe to use in the simulated environment, the design practically has 1 critical weak point where all the stress and load are concentrated, which is the shaft part. A suitable solution for this problem is to create a design where the motor shaft is only responsible for rotation and is not under load from the crib's weight.

2.5 Final Design Iteration

Analysis:

Meshing Gears

After running into multiple problems regarding the 2 motion mechanisms we have decided to implement a single motion mechanism. To cut costs and increase durability of the system.

The new mechanism consists of 2 meshing gears driven by a high torque dc motor. The motor shaft is responsible for driving the upper gear. And the lower gear is connected to the crib handles. This gears mesh ensures that the shaft of the motor is not subjected to the weight of the crib therefore less torque required for the crib to start swinging. The figures below show the gear configuration along with some analysis using SolidWorks.

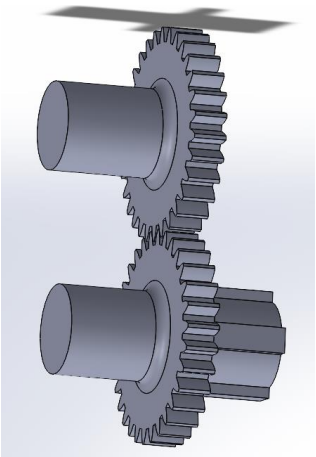


Figure 25: Meshing gears

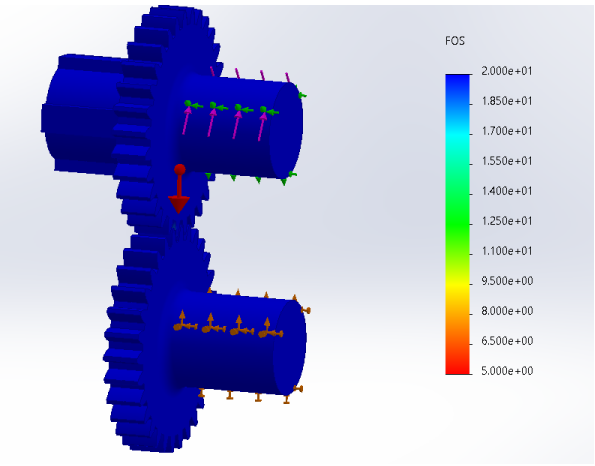


Figure 26: Factor of safety analysis

The Lower gear is connected from both sides, the first side is connected to the motor shaft, while the other side is connected to the crib handles.

The lower gear is also responsible for holding the weight of the crib from one of the 2 sides

This gear shaft must be strong enough to ensure stable rotation and avoid failures due to the weight of the baby or if one of the parents accidentally leaned on the crib. To make sure no such failures can happen we simulated a 40kg load on the far end of the gear as an extreme scenario. The pictures below show the simulation of the gear using SolidWorks.

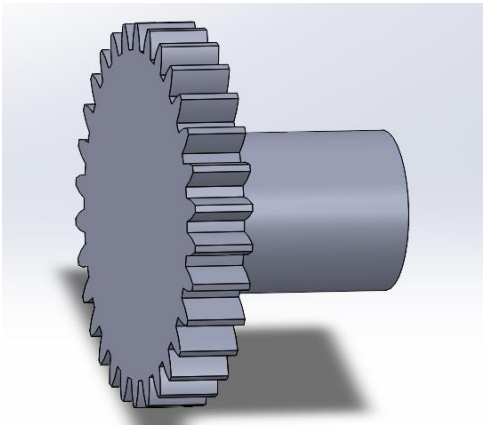


Figure 27: Lower Gear

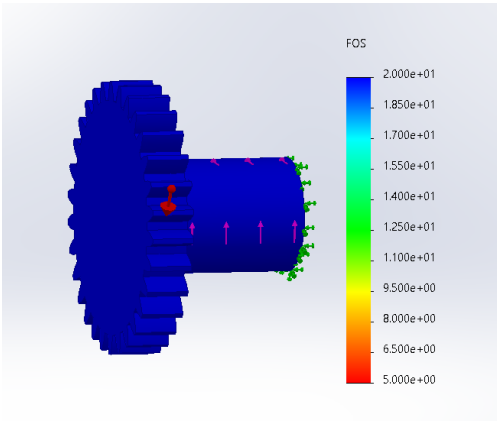


Figure 28: Lower Gear factor of safety analysis

Using Cast iron, the gear had a 60 Factor of safety, so we used PLA plastic because it's lighter, cheaper, quiet, and still withing the safe side having a minimum of 7 Factor of safety.

Crib Handles

These handles are responsible for holding the metal chassis. In the pictures below we can see the crib handle along with factor of safety analysis on a 40kg load.

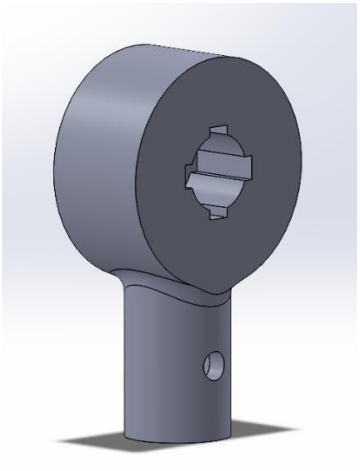


Figure 29: Crib handle

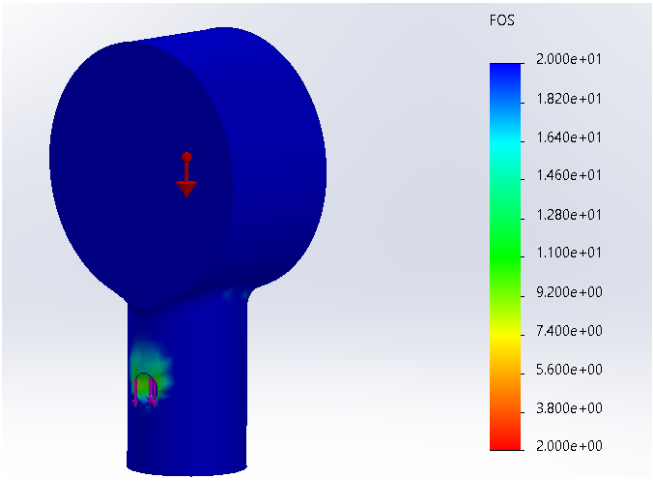


Figure 30: Crib handle factor of safety analysis

The gear has a minimum factor of safety of 60 if they were made from cast iron. But since the factor of safety is very high and since they are made of cast iron the weight of the piece will add unnecessary load onto the motor. To solve this problem, we were able to print this part using PLA.

Crib chassis

The chassis of the crib was made from cold drawn iron rods with an outer diameter of 23mm and inner diameter of 18mm. The simulation of the crib chassis indicates a minimum factor of safety of 12 times on a 40kg load distributed evenly across the chassis' bottom as shown in the figures below.

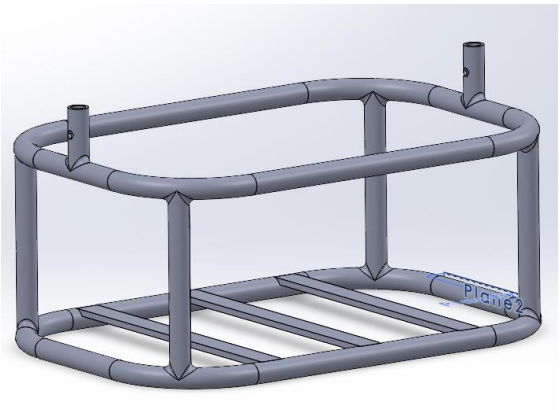


Figure 31: Chassis design

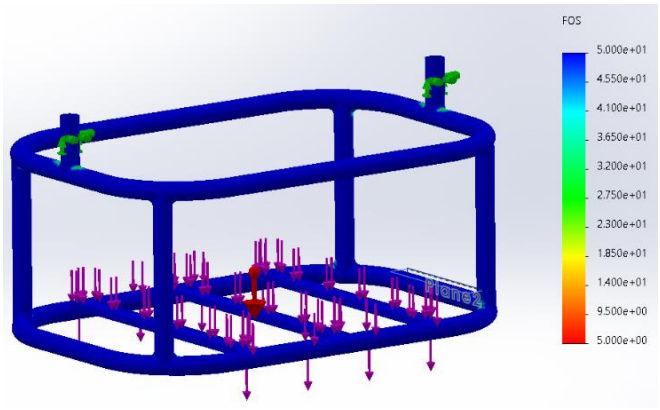


Figure 32: FOS analysis on the chassis

Crib pins

On the other side of the crib, we made a pin responsible for sharing the weight load with the lower gear. And both are placed inside of ball bearings number 6008 2RS to eliminate as much

friction as possible and to reduce the load on the motor. The pin is made of PLA and analysis are shown in the figures below.

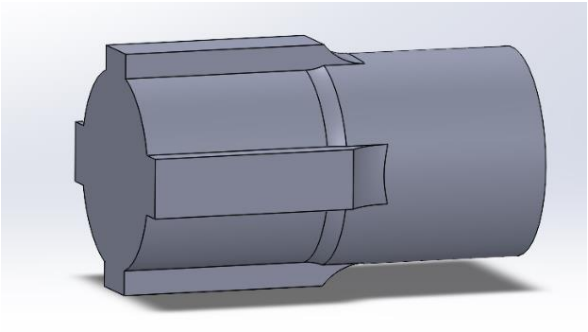


Figure 33: Crib Pin

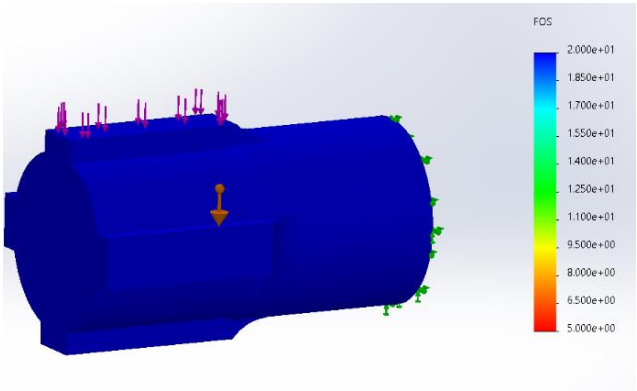


Figure 34: Factor of safety analysis of the pin

Crib base

The base of the crib is made of Oak wood and is responsible for the steadiness of the system when swinging. The simulation below shows base's design under 400N of weight along with the factor of safety analysis. The minimum factor of safety is higher than 100 with a max displacement of 0.0029 mm as shown in the figures below

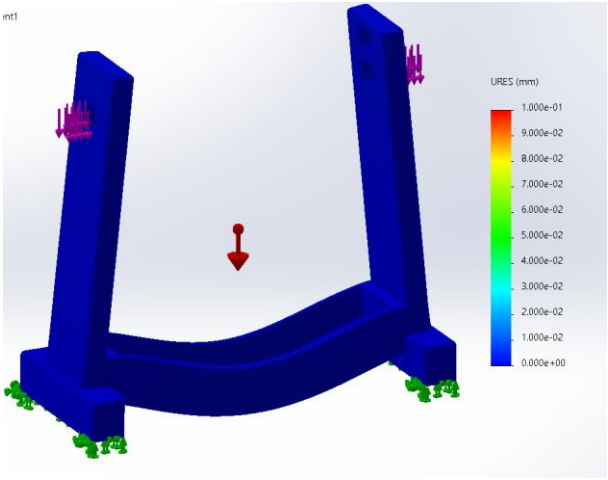


Figure 35: Displacment analysis of the base

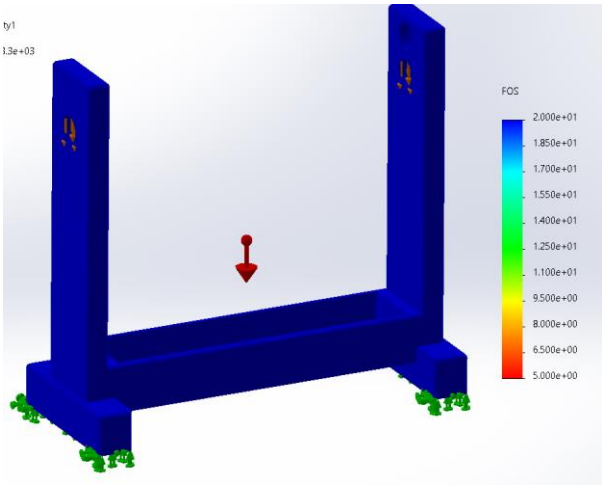


Figure 36: FOS analysis of the Base

In the figure below we can thee the assembly of the crib inside of SolidWorks along with all the forces and constraints that simulate real life conditions.

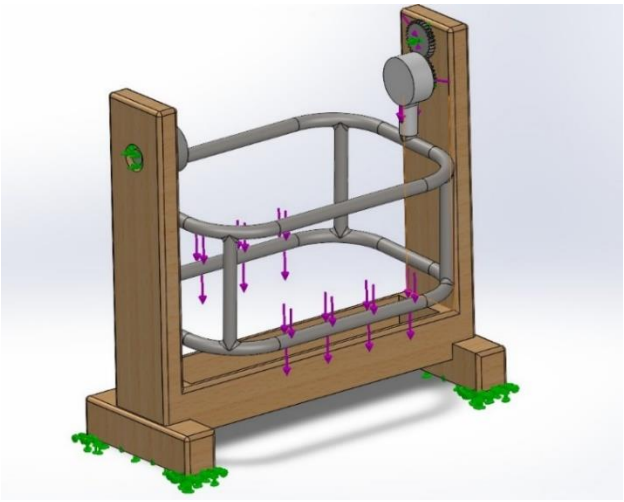


Figure 37: Assembly Simulation

Implementation

The Base and Chassis are custom made. All Plastic parts were created using the Ender 3 V3 SE 3d printer and ESUN PLA+ plastic filament, with layer lines parallel to the load's axis to avoid layer separation. The table below shows the exact specification of each pc.

Piece	Material	Specification	Weight
Crib Chassis	Carbon steel	Coated-Painted, against rust	8Kg
Base	Oak Wood	Sanded	10Kg
Handles	Brown-PLA+	70% infill density with 4 walls layers for extra shock resistance and durability	700g
Lower Gear	Gray-PLA+	70% infill density with 4 walls layers for extra shock resistance and durability	120g
Motor Gear	Gray-PLA+	70% infill density with 4 walls layers for extra shock resistance and durability	170g
Crib Pin	Gray-PLA+	100% infill density for maximum durability and fatigue resistance	100g

Table 3: Material and specifications

Design Discussion:

This final design was able to solve multiple problems, first the motor's shaft is now separated from the weight supporting shaft and only responsible for rotation. There are no critical points where stresses are concentrated. And the system's overall safety factor is higher. In the figures below we can see the parts of the project before assembly.

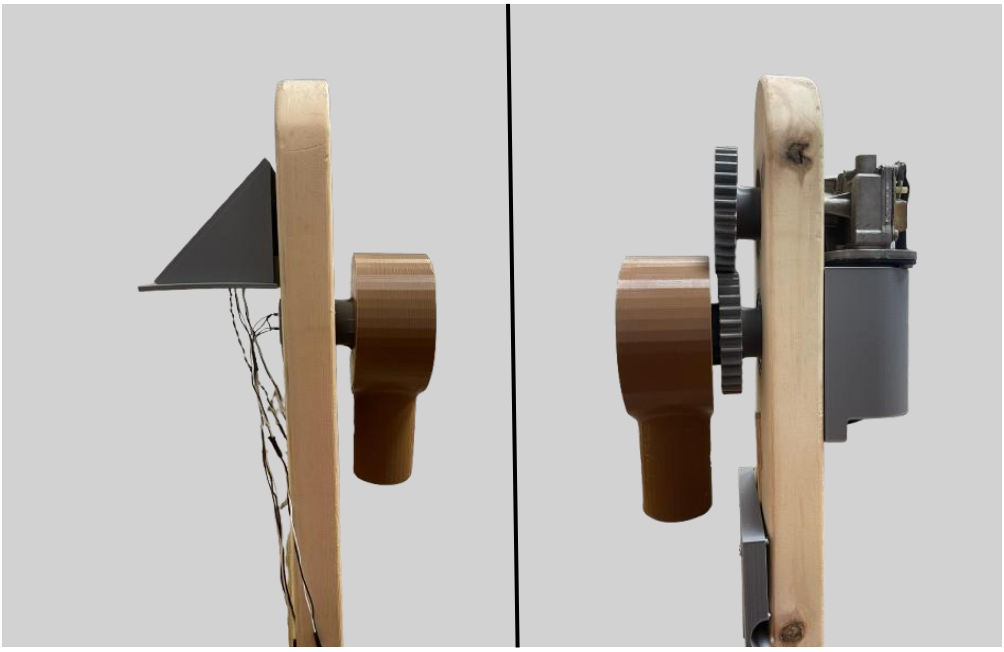


Figure 38: Implemented design

CHAPTER 3 FINAL DESIGN VALIDAION

3.1 Technical validation of Final prototype (process)

3.1.1 Electric system Schematic

Let us categorize our system into two main subsystems. The first subsystem is responsible for reading the health state of the baby.

For this purpose, we designed a wireless bracelet. For health readings, 2 main sensors are used to measure the heart rate, oxygen rate, and body temperature of the baby; those sensors are the MAX30102 and the MLX90614. Those sensors are compact, precise, and cost-effective, allowing for acceptable reading and monitoring. Figure 39 shows the Two sensors and its position in the bracelet.

Figure 39: Sensors of the Bracelet

An ESP32C3 Mini is used as a main controller. It provides compact size, low power consumption, and wireless communication.

To power the bracelet, a lithium-polymer battery (3.7V 1100mAh) is used with a TP4056 lithium battery charger module. This battery capacity can power the system for up to 6 h. The charger module is set to provide 0.5c of the battery capacity to provide average charging time with long life for the battery, Equation number 2 is used to find the compatible resistor to provide 0.5c. We find that the Rprog value is equal to 1.8kΩ. We implement a voltage divider circuit to determine the battery amount, ensuring that the voltage doesn't exceed 3.3 V. To provide the necessary voltage to the system, the MT3608 step-up module is utilized. It delivers a stable 5V by converting the variable voltage range of 4.2V to 3.3V into 5V. A standard on/off switch is utilized to turn the system on and off. Furthermore, the significance of this switch lies in its ability to ensure that the system does not draw power from the battery while charging. A Resettable Fuse |RXE 160 with 3A value is used to protect the circuit from any short circuit.
Important note: the user must not charge the battery and turn on the system simultaneously.

$$I_{CHG} = \frac{1000}{R_{PROG}} \quad (\text{in mA})$$

Equation 1 TP4056 Charging current change

Finally, for visualization, a 0.96" I2C IIC OLED LCD module is used. It is a compact size, and it’s easy to control. All the health data and battery details will occur at the display.to control. All the health data and battery details will occur at the display.

Figure 40 shows the components and their position in the system.

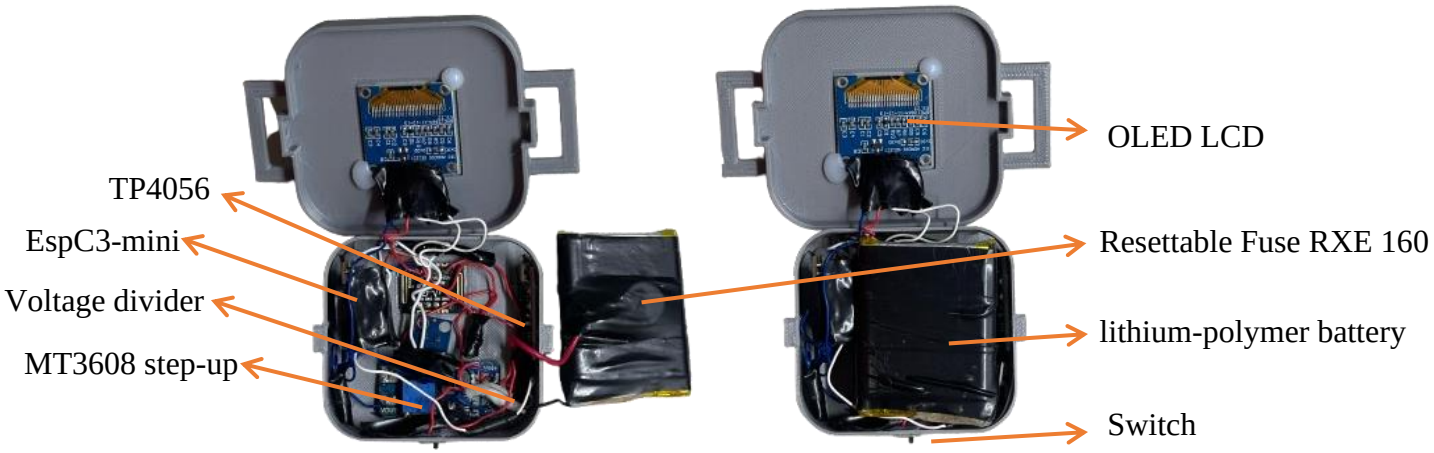


Figure 40: Bracelet components

Figure 41 illustrates the complete connection scheme, highlighting the three categories of the system: sensors/actuators, power, and microcontroller.

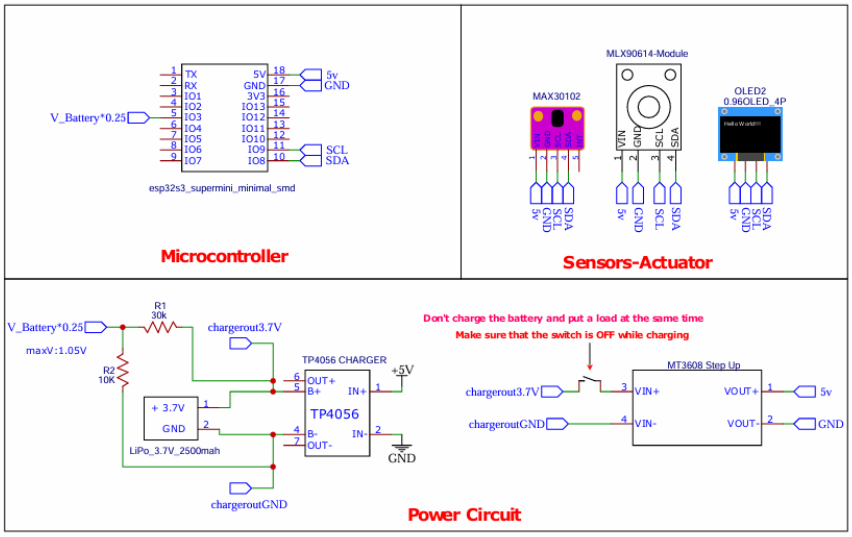


Figure 41: Bracelet Electric Schematic

Electrical System Schematic and Power Architecture of the Smart Crib

The electrical system of the smart crib is engineered to balance high-performance actuation, low-voltage embedded control, and electrical safety, all within a unified and efficient architecture.

Given the complexity and diversity of the hardware involved, the electrical design was carefully planned to ensure power integrity, voltage regulation, current isolation, and protection mechanisms, particularly because the system operates in close proximity to an infant.

1. Power Supply Architecture

At the heart of the crib's power distribution network is a 240V AC to 12V DC power supply unit. This unit performs the initial AC-DC conversion, transforming the mains voltage into a usable low-voltage DC source. The selection of 12V as the base system voltage is deliberate, as it is high enough to drive electromechanical components like motors, yet still within the safe range for low-voltage electronics.

From this 12V DC source, the power is split and regulated through two primary paths:

High-current 12V Line: Directly powers the Bosch C300 wiper motor via the BTS7960 motor driver. The motor is a high torque brushed DC motor, typically rated for automotive use, and capable of withstanding heavy loads. Since the total load (crib chassis + baby) may exceed 15 kg, the driver must sustain high peak currents. The BTS7960 is well suited for this, offering dual full-bridge drivers with up to 43A peak current capability and internal current sensing features. This line is unregulated in terms of voltage, but all components on this path are rated for 12V operation.

Low-voltage 5V Regulation Line: A DC-DC buck converter steps down the 12V to 5V to supply power to the Arduino controller, rotary encoder, 16x2 I2C LCD, AS5600 encoder, sound sensor, DFPlayer module, relay, and SHT30 sensor. The regulator is chosen with sufficient output current capacity (typically 2A or higher) to prevent voltage drops during peak activity.

Additionally, the Arduino board provides a 3.3V output, which is used to power the ESP32 module. This level shifting is necessary as the ESP32 operates strictly at 3.3V logic levels and is not tolerant to 5V.

2. Electrical Safety and Protection

Given the involvement of mains electricity (240V AC) and vulnerable end-users, the system integrates essential electrical safety features:

Circuit Breaker (Disjoncteur): Positioned before the AC-DC power supply, this breaker acts as a first line of defense. In the event of a short circuit, overload, or insulation fault, the disjoncteur will interrupt the electrical supply, preventing fire hazards, electrical shock, or component damage. Its inclusion is vital, especially in continuous-operation systems installed in domestic environments.

Current Isolation & Grounding: The 240V input and 12V output are electrically isolated via a transformer-based PSU, ensuring that the high-voltage side does not come into contact with the low-voltage logic circuits. Proper grounding and insulation ensure user safety and system stability.

Relay Isolation: The LED lighting system under the crib is controlled using a mechanical relay, which provides electrical isolation between the control logic (5V) and the LED power circuit, preventing high current from feeding back into the Arduino in case of a fault.

Voltage Level	Purpose	Powered Components
240V AC	Mains input	Enters PSU for conversion
12V DC	Motor drive and base voltage	Bosch C300 motor, BTS7960 driver
5V DC	Control logic and sensors	Arduino, LCD, DFPlayer, sensors
3.3V DC	Wireless module	ESP32

Table 4: Voltage Levels and Power Distribution

The design considers the maximum current draw of each component. For instance:

The BTS7960 and motor may draw several amps under load — requiring thicker gauge wiring (14–16 AWG) and active cooling if sustained.

The 5V rail supports low-power devices (<100 mA each) but collectively can reach up to 1A or more.

Proper capacitor decoupling and EMI suppression techniques are implemented near sensitive components to minimize ripple and noise from motor switching.

4. Electrical Control and Communication Interfaces

The Arduino board acts as the primary control unit, handling logic-level signals, sensor data processing, and user interaction. It communicates with:

I2C Devices: The 16x2 LCD, SHT30 sensor, and AS5600 encoder share the I2C bus for data exchange.

Digital Pins: Used to control the relay, read the rotary encoder pulses, and communicate with the DFPlayer and sound sensor.

ESP32: Powered by the 3.3V line, this module manages wireless functions (e.g., Wi-Fi, Bluetooth) and could eventually support mobile app communication or cloud logging of sensor data.

5. Design Considerations for Long-Term Operation

The system is designed for continuous usage, meaning the electrical components must remain stable, cool, and protected during operation. Key considerations include:

- Thermal Management: Heat sinks are mounted on the BTS7960 and step-down regulators to dissipate heat under full load conditions.
- Load Testing: The power system has been tested under full-load scenarios to ensure voltage stability and protection response times.

- Fail-Safe Operation: In case of unexpected shutdowns (e.g., power outage), the motor driver and relay default to an OFF state, preventing uncontrolled motion or lighting.

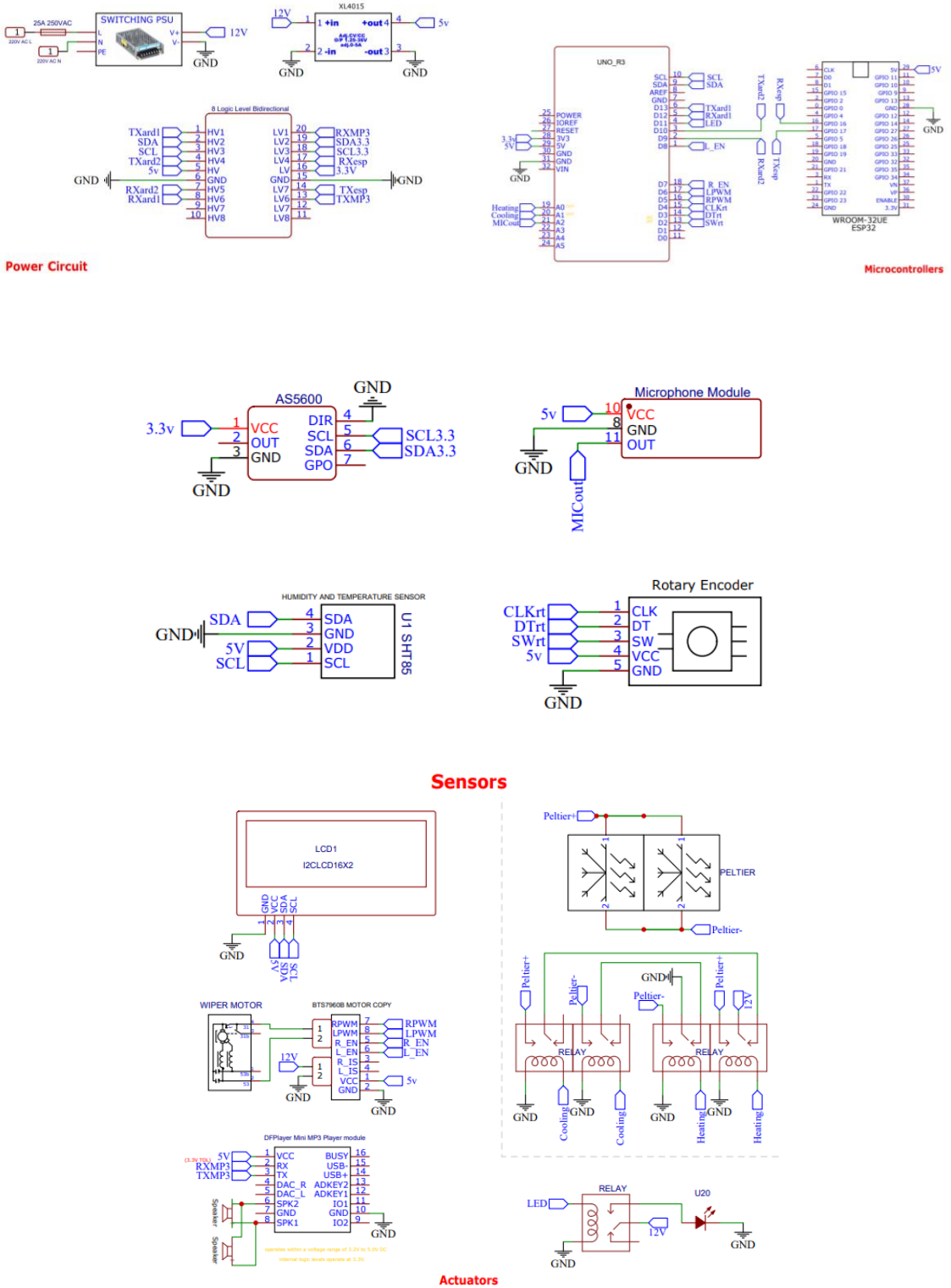


Figure 42: Crib Electric Schematic

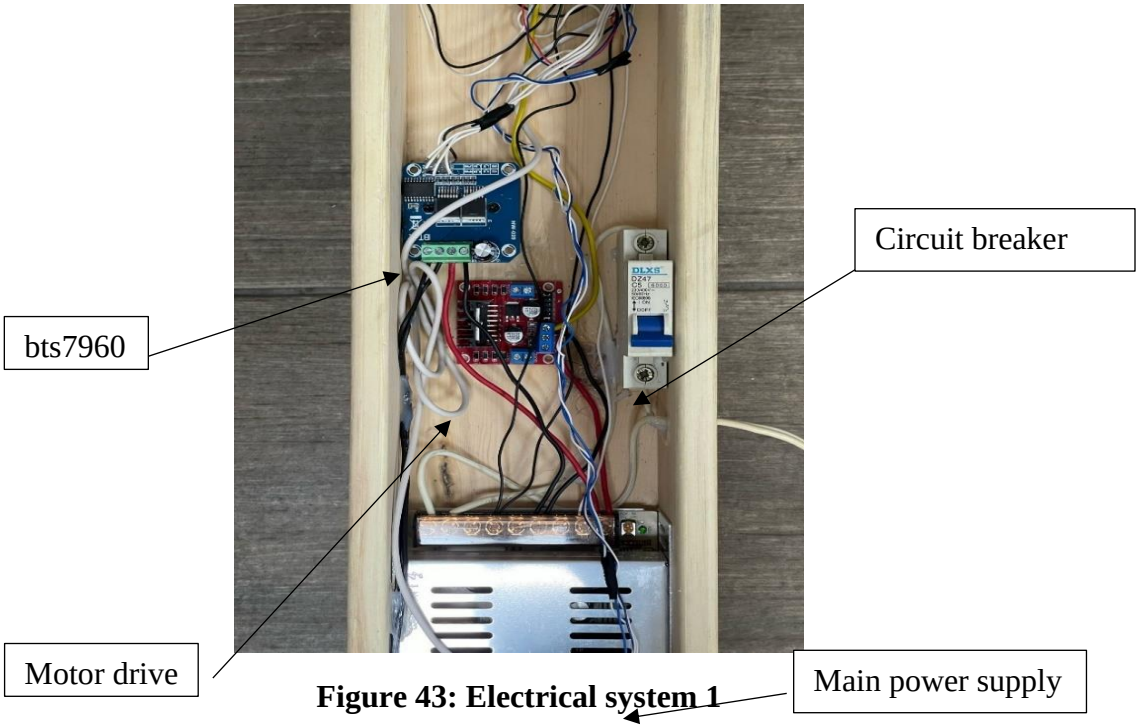


Figure 43: Electrical system 1

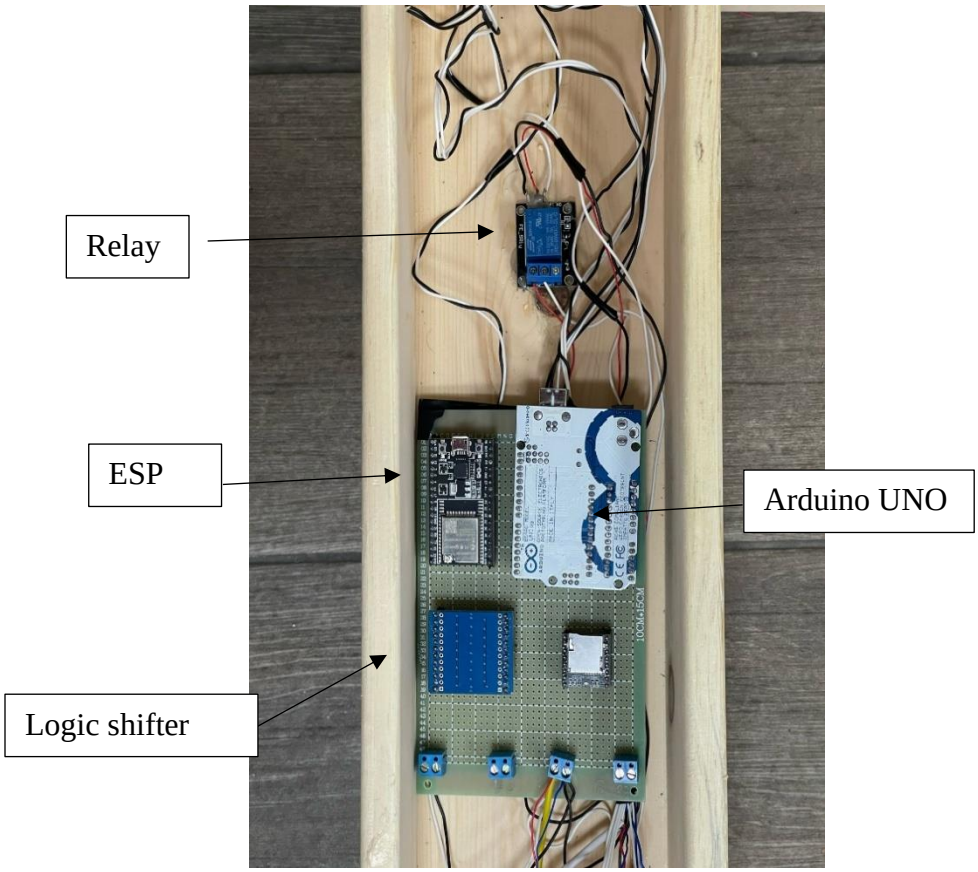


Figure 44: Electrical system 2

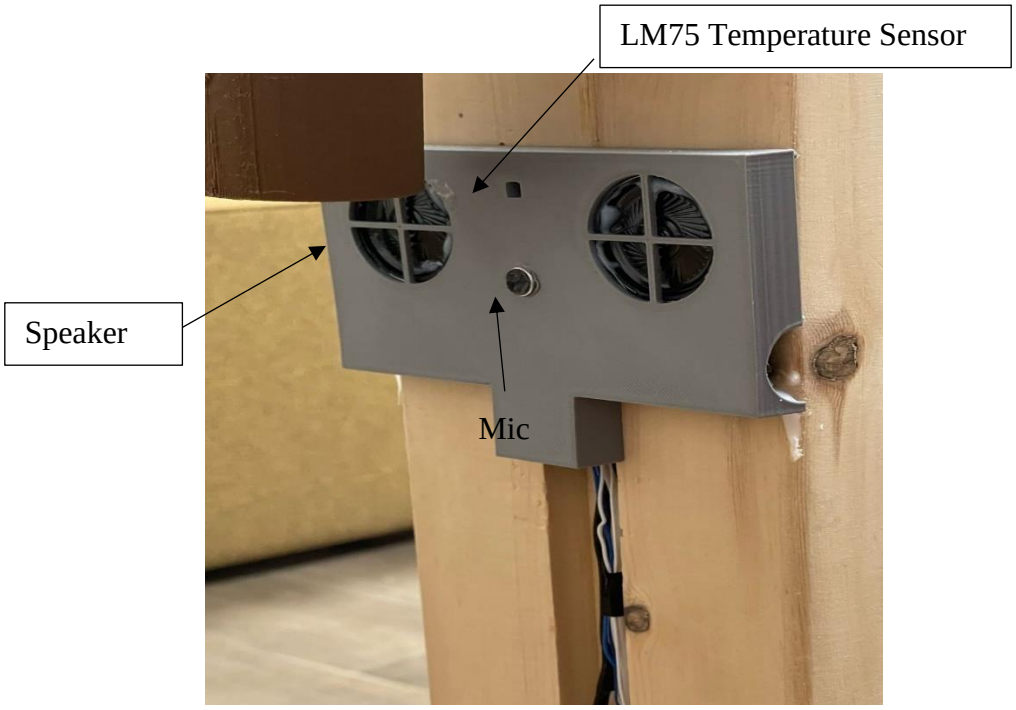


Figure 45: Speakers, Mic, and Temp Sensor

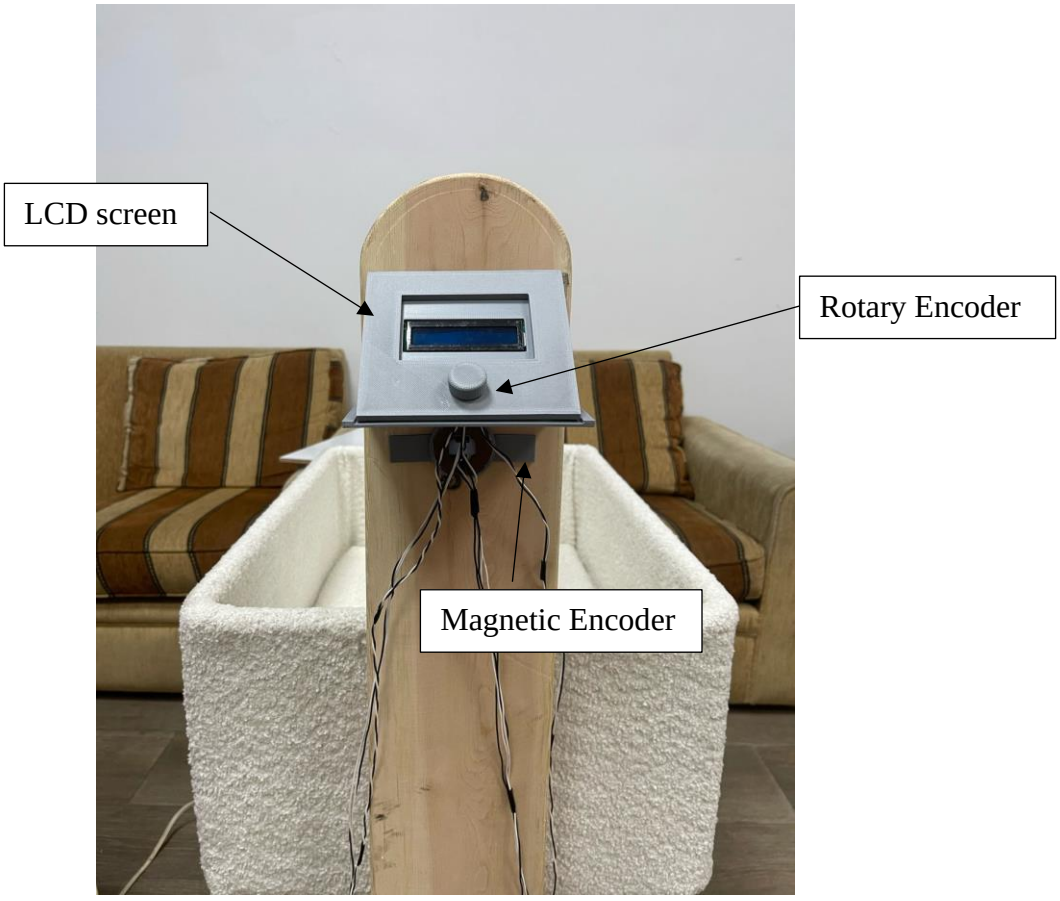


Figure 46: HMI and magnetic Encoder

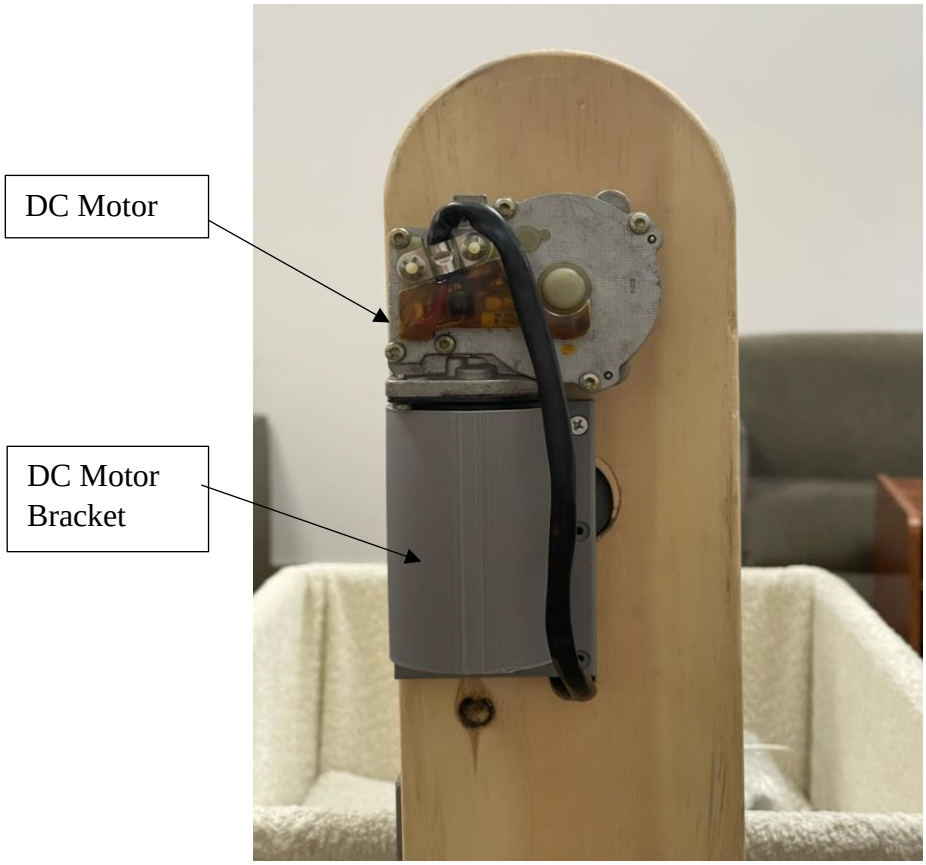


Figure 47: Motor and bracket

3.1.2 Embedded System Communication Architecture

The communication system will be divided into 3 subsystems, made to ensure a robust, flexible, and reliable communication system.

The first subsystem represents communication between the bracelet and the crib system. The aim of this communication is to transmit health data about the baby to the main controller. To do so, the system will start by using the ESP-NOW protocol between the EDS32C3-mini and ESP32-Wriim-32D. This protocol is compatible with our task because it provides direct, peer-to-peer communication without Wi-Fi infrastructure, low latency and fast data transmission, low power consumption (the bracelet is battery-based; the less power consumption, the better), flexible communication topologies, reliability and robustness in congested environments, communication ranges up to 220 meters under ideal conditions or 50–100 meters in typical indoor environments, and easy pairing and device provisioning. This makes ESP-NOW a perfect choice in our scenario.

In our system, we send sensor data from one ESP32 to another using ESP-NOW, and to ensure efficient and reliable transmission, we use a packed structure composed of fixed-size unsigned integer types, specifically uint8_t and uint16_t. These types represent 8-bit and 16-bit unsigned integers, offering predictable and compact data sizes. For example, uint8_t can store values from 0 to 255 using just 1 byte, while uint16_t handles values from 0 to 65,535 using 2 bytes. This

approach significantly reduces memory usage and transmission overhead compared to using floating-point types. To maintain precision for values like temperature, we multiply them by 10 and store them as integers (e.g., 36.5°C becomes 365). This method avoids floating-point issues and keeps the data structure simple and consistent. Overall, using a packed structure with these types makes the data transmission faster, more memory-efficient, and easier to decode on the receiving ESP32. Figure 48 shows the structure and the type of sensory data 1.

```
// Optimized structure to send data via ESP-NOW that's already compatible with Arduino
typedef struct __attribute__((packed)) {
  uint16_t bodyTemp;     // Store as integer (value * 100) for 2 decimal places
  uint16_t ambientTemp;  // Store as integer (value * 100) for 2 decimal places
  uint8_t heartRate;     // BPM as uint8_t (0-255)
  uint8_t oxygenLevel;   // SpO2 as uint8_t (0-100%)
  uint8_t batteryLevel;  // Battery percentage as uint8_t (0-100%)
  uint8_t batteryVolt;   // Battery voltage * 10 for 1 decimal place
} struct_message;
```

Figure 48: packed structure of the sensory data 1

After receiving the data from the ESP32C3 Mini, the ESP32-WROOM-32D needs to transmit this data to the Arduino (main microcontroller). To do so, a UART protocol is used to provide an easy, robust, and easy-to-implement protocol, especially in the short distances (the ESP32-WROOM-32D is very close to the Arduino).

Figure 49 illustrates the first subsystem communication lines, protocols, the data that is being sent, and their type.

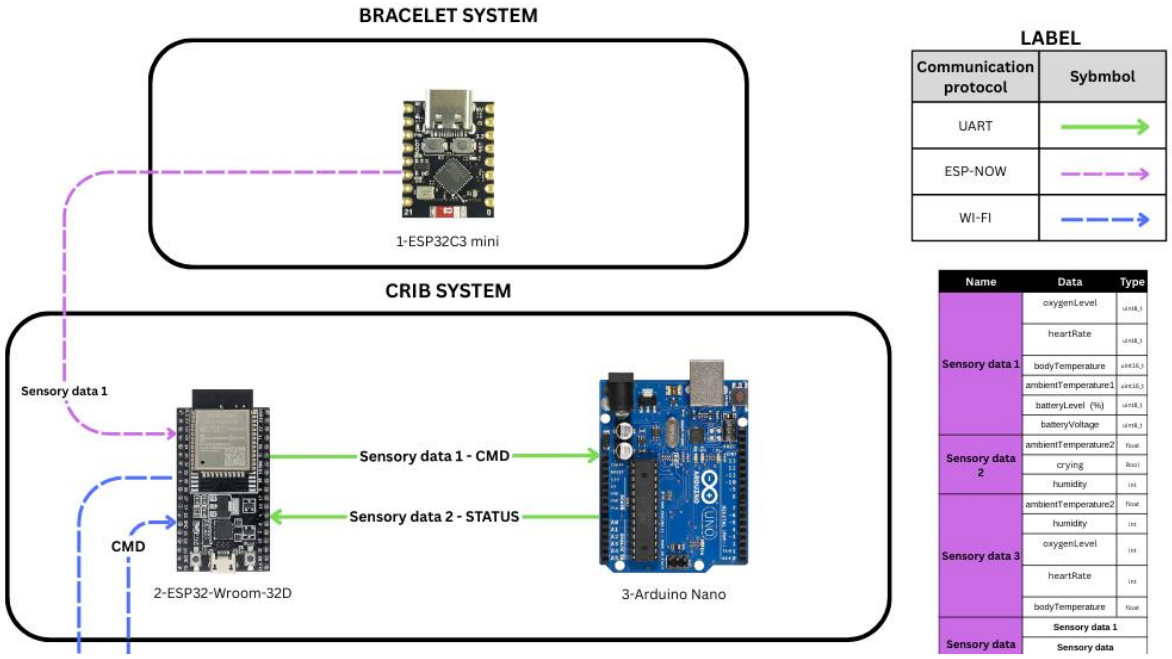


Figure 49: Communication line between bracelet and crib

Figure 50 shows how the data in the bracelet and in the crib is synchronized, representing the previous communication line .

Figure 50: Data transmission from bracelet to crib

The second subsystem is the one that connects the Arduino Uno (main microcontroller) to the telephone application; this communication will give the user the access to read and control all the data via the telephone. The Firebase works as a bridge between the mobile app and the crib system; it's an online database that ensures stability and an easy-to-work environment.

In this subsystem we will have two types of data: readings (sensory, from Arduino to Firebase) and commands (CMD, from mobile to Arduino).

Starting with the Readings line, after gathering all the readings of the sensors and the status of the crib (motor mode, light mode, etc.), the Arduino is going to send those sensory data and status data to the ESP32-WROOM. After receiving this data, the ESP32-WROOM is going to send them to the Firebase via Wi-Fi communication, and now the Firebase will save those values. The mobile application will read those values and display them.

Figure 51 illustrates the readings line, their protocol, data type, and root.

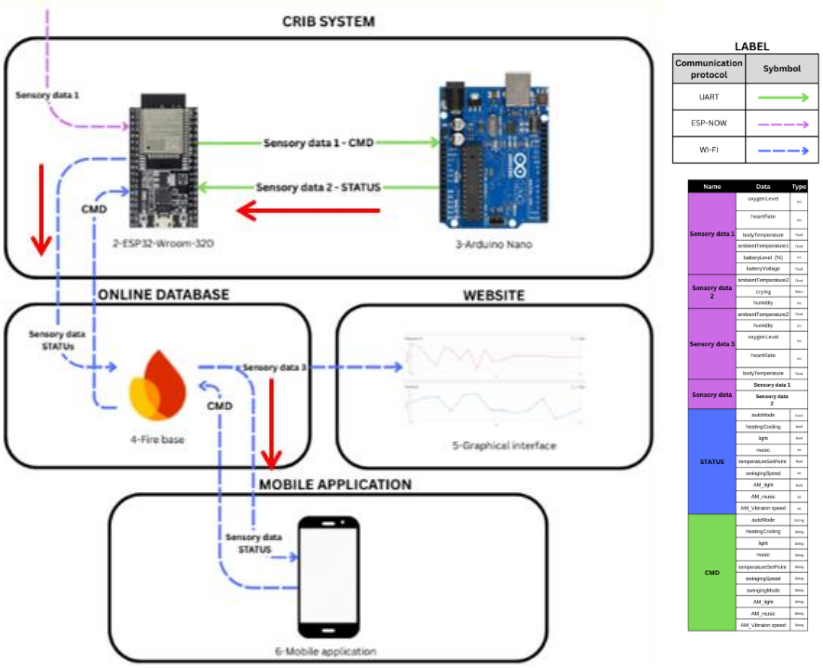


Figure 51: Reading line communication schematic

Figure 52: Data Base

The second line in the second subsystem is the command line; this works exactly in the reverse way of the first line (readings line). The command will appear when the user clicks some button on the mobile app, and this will go to the Firebase. The ESP32 WROOM will read it and send it to the Arduino. The Arduino will take action based on the command.

To avoid problems, a flag variable is used for each command line. This flag will be set to 1 (string) each time a new command occurs. The ESP32-WROOM will check which flag is triggered and send the command to the Arduino and reset the flag after this process. Using the flag method is essential in our system to avoid any possible interlock between some old and new commands and between the main HMI screen and the mobile application.

Figure 53 illustrates the command line, their protocol, data type, and root.

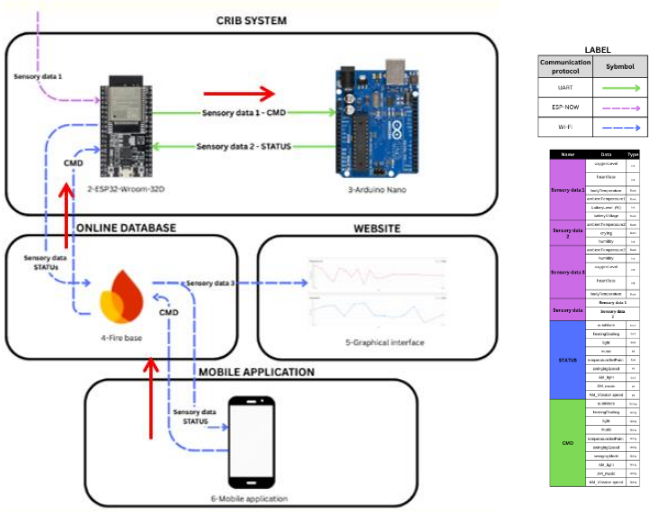


Figure 53: Command line communication schematic

Figure 54 shows the command variables in the database with their flags.

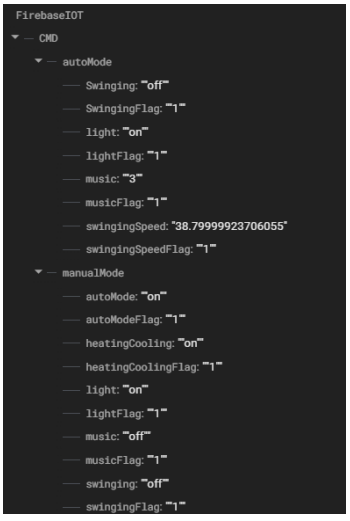


Figure 54: CMD data in the database

Finally, the third subsystem is a simple, yet effective setup designed to visualize health data using a chart. This subsystem consists of the Firebase Realtime Database and a hosted web server. The web page reads values such as body temperature, humidity, heart rate, and blood oxygen level (SpO_2), and displays them in a dynamic chart with respect to time and date. The chart shows data from the last seven days, and any older entries are automatically deleted to maintain a clean and manageable database. To accurately timestamp each reading, the ESP32-WROOM module utilizes an NTP (Network Time Protocol) library to retrieve real-time data. It then attaches the corresponding timestamp to each reading before sending it to Firebase.

Figure 55 shows the web page that contains the graph

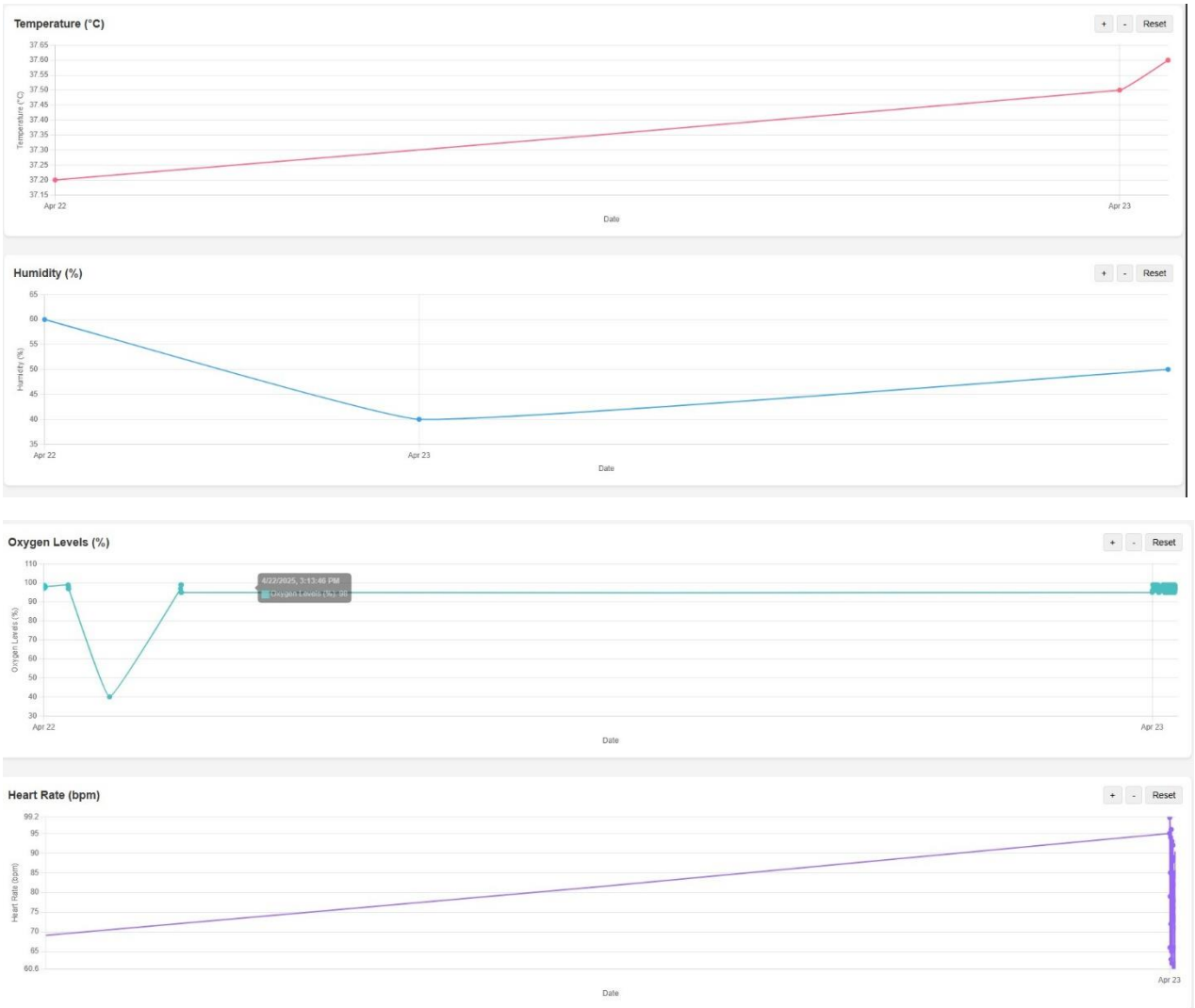


Figure 55: Graphs of the health condition of the baby

Figure 56 shows the timed data and how they are categorized in the database.

Figure 56: shows the timed data format and path in the firebase

Figure 57 shows the summary schematic of the Embedded System Communication Architecture with the communication protocol, data transferred with its type.

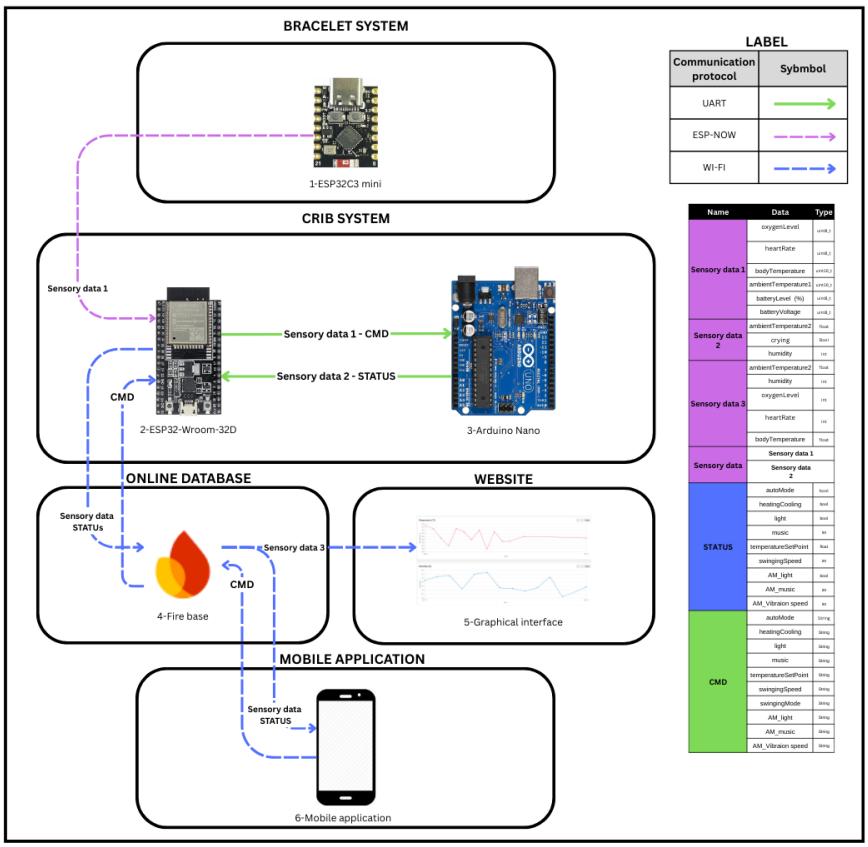


Figure 57: Full communication Architecture

3.1.3 Mode of operation

Bracelet

The main aim of the bracelet is to read and transmit those readings to the main controller. To do so, the algorithm is straightforward; it follows the following order: reading sensory values, printing them, and sending them via ESP-NOW. Figure 58 represents a flow chart that shows the full operation of the bracelet.

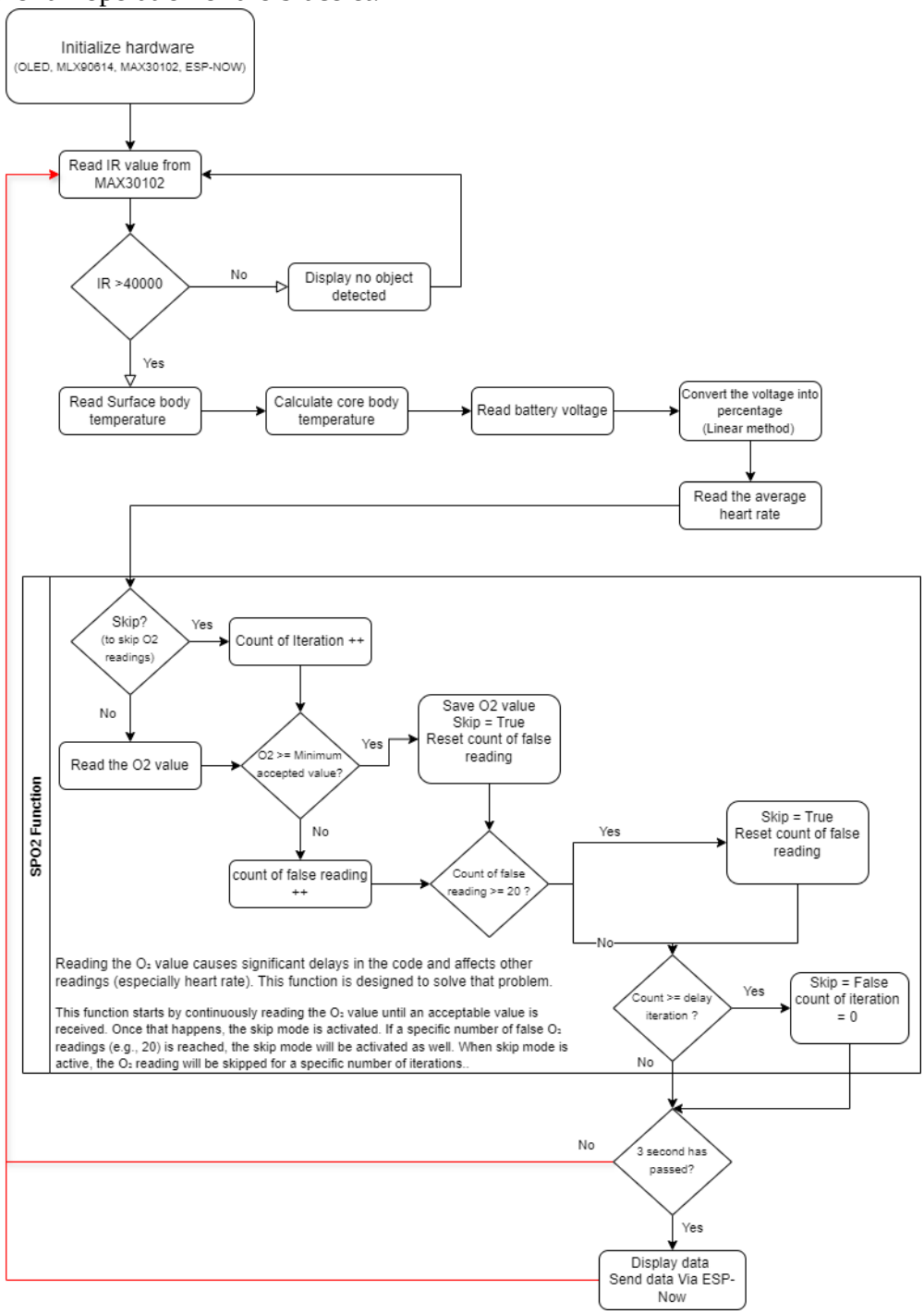


Figure 58: Bracelet algorithm flow chart

To start the readings and the entire operation, the baby shall put the bracelet on his hand. When the MAX30102 IR readings become less than 40000, that means that the baby is putting the bracelet on, and he is close enough to start the readings. **: Bracelet- no object detected screen**, represents the screen that will occur when the the reading of IR sensor is more then 40000.

Figure 59: Bracelet- no object detected screen

There are two types of data in our system: one that's fast reading and others that are long ones. Those long ones need to take a lot of samples to start giving an output.

Starting with the fast readings system, first, the system will read the surface skin temperature of the baby using the MLX90614 sensor. To calculate the internal body temperature, a linear calibration curve is implemented based on the ambient and surface body of the baby (to get more reliable readings, we can edit at the calibration curve and apply more testing).

The user shall be notified about the battery level all the time; to do so, the system will now read the average voltage value of the battery and multiply it by the value of the voltage divider circuit. A linear conversion is applied to change this voltage into a percentage; 3.3 V represents 0%, and 4.2 V represents 100%. One important note is that the battery voltage is actually not linear, but we are going to use a linear curve for simplicity.

Going to the next type of reading, which is the slow readings, the system will start to take the required samples of the heart rate, but without applying any while loops or delays. But one major problem in the system is that the SpO2 readings and the heart rate readings affect each other, and we can't get the two values together. We won't see any heart rate values until SpO2 has a value.

The SpO2 contains a lot of loops that are built in its library, leading to delaying the system too much. That's why we let the SpO2 readings happen once every specific number of iterations. This method can help to stabilize the system and will let us get the HR values. There is some filtration happening at the HR, and the SpO2 values—every value that is below 40 or 90 will be rejected.

After finishing the readings, now we are going to check if three seconds have passed or not using the Millis function. If yes, we are going to present the health readings at the OLED display. readings Shows the values of all the readings at the screen.

Figure 60: Bracelet- screen readings

After printing them, we now need to prepare to send them via ESP-Now. To do so, we are going to fill our structure value with the required data and in the right type. The data that we are going to send will be of the Unit type, so the integer readings will stay the same and the float values will be multiplied by 10 if uint_8 or 100 if uint16 (higher accuracy). This step is important to convert the float value into an integer; the value will be converted back into the received microcontroller. **: Transmitted data from bracelet to esp32 Via Esp-now** Shows how the data is being transferred to the main controller using ESP-now protocol.

Figure 61: Transmitted data from bracelet to esp32 Via Esp-now

Now we are going to start the loop again and repeat all the previous steps.

Important Considerations: The SpO2 and heart rate readings are not so accurate, but comparing the accuracy with the cost, they are pretty good and compatible with the system. The heart rate readings need close to 1 minute to start giving good value, and it accumulates until it reaches the correct value.

Now let's talk about the algorithm and how our crib will function:
First, as a user, we will only be controlling the crib and reading the value from sensors using the lcd and rotary encoder, so let see the interface of the lcd screen step by step.

We will start by O2 Level this value will be taken from the bracelet and passed it on the screen not only O2 level but also Heart rate, Body temperature, ambient temperature and battery level all these value will be appear on the screen and even if the bracelet is turn off it will still print the last value of the sensor, this step is made as safety mode so if for example the heart rate are abnormal and the mother didn't recognize this when she will come back and check the screen it will take into consideration this point.

Now what about the crying status of the babe, to do so we first try to integrate an ai that detect the crying phase of a baby from other noise but unfortunately we can't implement this on an Arduino UNO so the solution what to adjust the code so we can approach as much to the crying phase of a babe and we give it a delay so the value on the screen will change only after 30 sec to not affect the screen brightness and control.

Finally for the sensor and reading we also have temperature and humidity sensor near speaker and mic, so he takes this value and print it also on the screen

Now the real important part actuators:
In our crib we have a manual mode and an auto mode we will talk first about the manual mode:

Manual Mode:
The user can control four things in the crib:

- **Motor:** when the user press on the motor on the menu items, screen will disappear and we will go into a new page called options screen in this screen the user will be able to turn on or off the motor, so if the motor is on when he press on motor and go to the options screen he will be able to turn off the motor and vice versa, and if the user go the options screen by mistake he can press a long press on the push button for 1.5 s and it will go back to the main screen .When the motor is turn on he will move for a specific angle ±3 degrees with a constant speed and he will only be off when the user press on turn off. If you are asking how a dc motor will know that he rotates 3 degrees, now it's the time to explain about the magnetic encoder. A magnetic encoder is a type of position sensor that detects rotational position, speed, or direction of a shaft using a magnetic field instead of physical contact or optics, so like this when the motor rotate 3 degrees the magnetic will rotate 3 degrees and like this the encoder send data to Arduino to stop the

motor and reverse it to the next side, and when we turn off he motor he will go back to the initial position (middle of the crib).

- **Light:** when the user press on the light on the menu items, screen will disappear and we will go into a new page called options screen in this screen the user will be able to turn on or off the light, so if the light is on when he press on light and go to the options screen he will be able to turn off the light and vice versa, and if the user go the options screen by mistake he can press a long press on the push button for 1.5 s and it will go back to the main screen. The light is controlled by a relay connected to Arduino.

- **Music:** when the user press on the music on the menu items, screen will disappear and we will go into a new page called options screen in this screen the user will be able to choose the song he wants, we upload 3 songs on our Arduino he can choose which one to play by rotating the encoder and press on the track, the song will repeat till the user enter again to the music go to the final option which will be turn off song, and if the user go the options screen by mistake he can press a long press on the push button for 1.5 s and it will go back to the main screen, but how when upload music to the Arduino without having an mp3 audio, you can play music using tone() on a buzzer or speaker connected to a digital pin. In our code we did so we use melody array to put full tone of a certain music so we can hear tones and not music because Arduino has a short memory.

- **Heating/cooling:** When the user presses on the heating and cooling on the menu items, the screen will disappear, and we will go into a new page called options screen in this screen the user will be able to choose if he wants to cool the bed or to heat the bed of the babe. When we did so used Peltier cells connected to 4 relays connected to Arduino and for safety, we put a value where the relay should turn off for both cooling and heating.

Auto mode:

When the user press on auto mode screen will disappear and we will go into a new page called options screen in this screen the user will be able to turn on or off the auto mode. When auto mode is activate the system will be looking into the crying status if the babe cry the motor and music will be turn on for exactly 1 min even if the condition back to babe is calm, and for the music the system will choose randomly one of track or the user can a adjust it so he can choose what to play in auto mode.

3.1.4 Mobile application

To build the mobile application, the MIT App Inventor is used, a block-based mobile application creator. The power of the MIT App Inventor is that it gives the user the ability to categorize the entire mobile application. This website provides all the necessary components, from buttons and layers to text colors and all the logic needed. The advantage of this website is that there is a built-in function for Firebase, which will facilitate the process and make it easier; moreover, it can let the user upload a full webpage inside it. Those features make the MIT app very suitable for our project. The main drawbacks of this application are that it works only in the Android system.

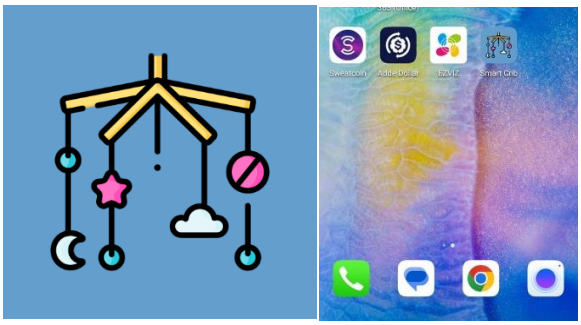


Figure 62: shows the Mobile app icon.

The system will include a mobile application that enables users to monitor and control the Crib via Wi-Fi, as previously mentioned. The app will display real-time sensor readings, crib status, active modes, and actuator operations, providing comprehensive oversight. The user interface consists of four intuitive screens for seamless navigation. Figure 63 shows the different available screens in the mobile application.

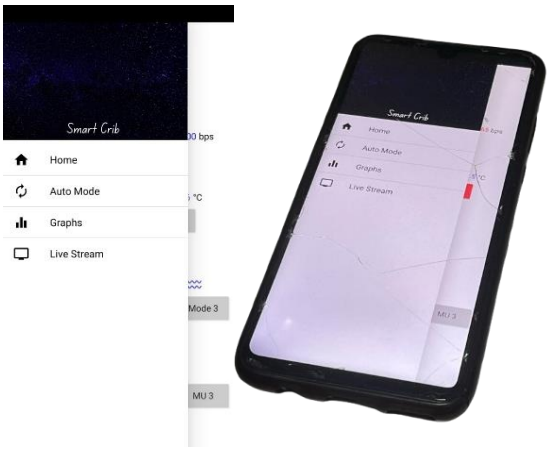


Figure 63: mobile application screens

Screen 1: Home Screen

This serves as the central dashboard, allowing users to view all vital baby data (e.g., health metrics) and crib status briefly. From here, parents can also control the crib's functionalities, such as activating or deactivating heating/cooling, swinging, and the light system. Moreover, the user can set the temperature set point and the swinging speed. And he can switch between music. He can see a live demonstration about what is happening in the system and whether the command that he sends is activated or not.

The home screen control will be locked when the auto mode is activated. The user can only control the cooling and heating system because it's not included in the auto mode.

As illustrated in Figure 64, the home screen presents the baby's health status alongside the crib's operational state. Visual indicators, such as color-changing push buttons and dynamic icons, provide immediate feedback on the crib's current mode and active actuators.

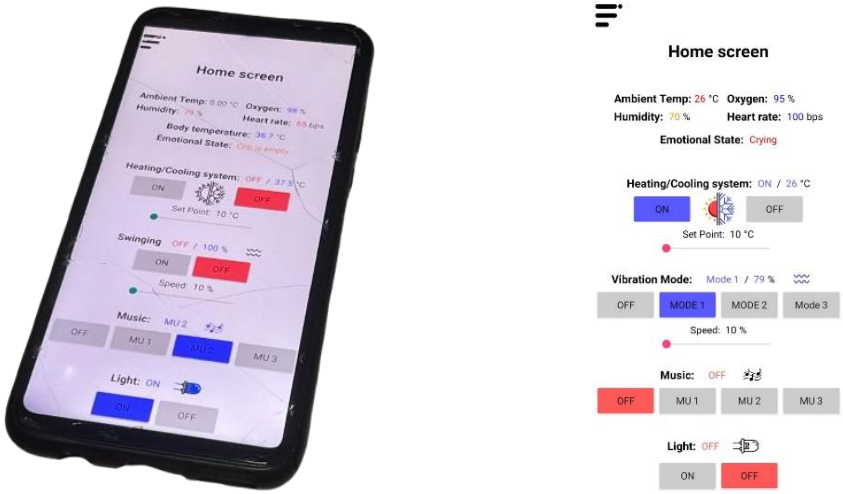


Figure 64: Mobile application- Home Screen-Auto mode is off (two different versions)

Figure 65 illustrates the home screen when the auto mode is activated.

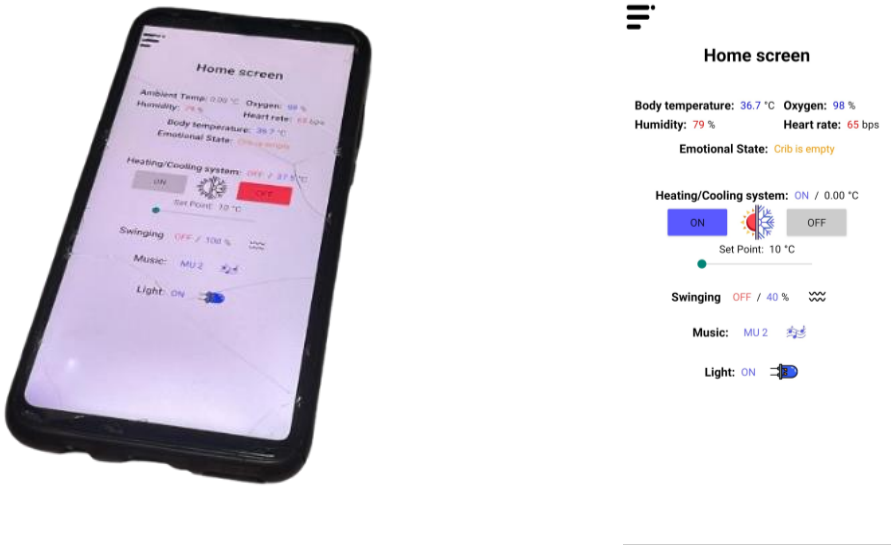


Figure 65: Mobile application- Home screen- auto mode is ON

Screen 2: Auto mode screen

This screen allows the user either to activate or deactivate the auto mode and set it's value plus checking the values available in the system. The user can't change the auto mode parameter if the auto mode is not activated.

Figure 66 shows the auto mode screen in on and off mode.

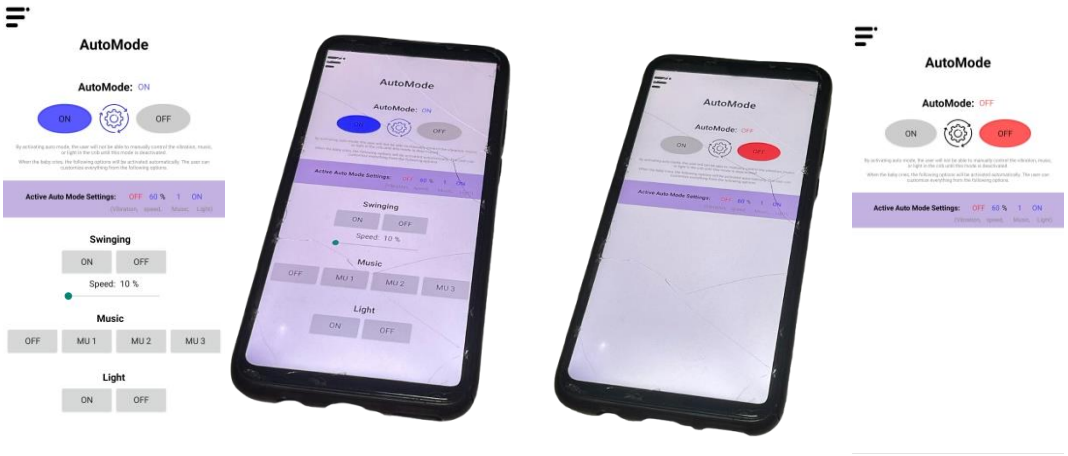


Figure 66: Mobile application- Auto mode screen- auto mode ON/OFF

Screen 3: Graphs screen

This screen shows a web page. This web page shows the graphs of the body temperature, humidity, heart rate, and oxygen rate of the baby for the last seven days. This website is created using JavaScript and is hosted by GitHub. The user can reach those graphs not only from the phone but from any search engine. Those graphs give the user the access to check each point of data and see when exactly it appears.

Figure 67 shows the graphs screen.

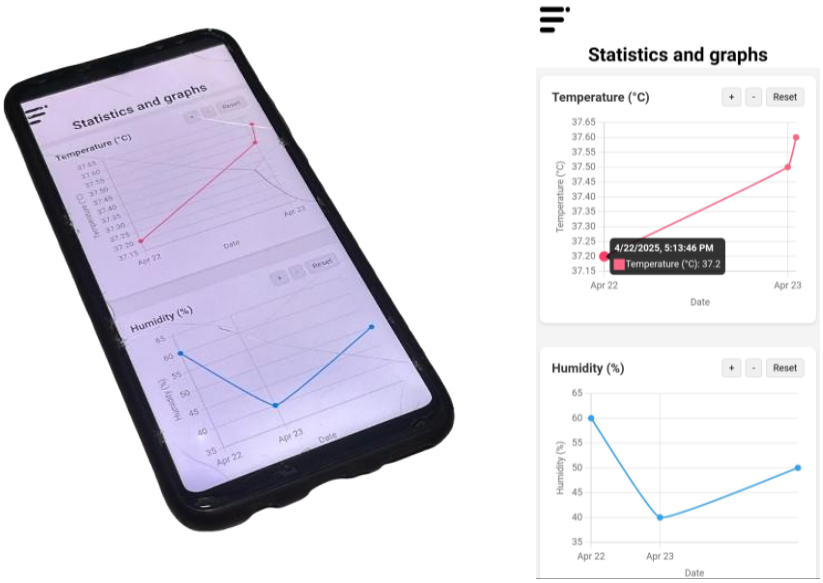


Figure 67: Mobile application- Graphs screen

Screen 3: Live stream screen

This screen is empty for now and it will be for future work , but the main aim of it was to see a live video stream of the baby. Figure 68 illustrates this screen.

Live Stream

Figure 68: Mobile application- Live stream screen

The mobile application code is fully developed and functional. However, the corresponding code on the ESP32 and Arduino side is still under development. As of now, the user can control a few features of the crib via the mobile app, and sensory values are successfully transmitted and displayed. However, the system does not yet support reading or displaying the full operational status of the crib.

3.1.5 Safety & Codes and standers

To ensure the safety, reliability, and robustness of the Smart Baby Crib, the design has been reviewed against internationally recognized standards. The following table summarizes the relevant mechanical, electrical, and communication standards, along with how each has been considered or applied in the current system.

1. Mechanical Safety Standards

Standard	Title	Application in Project
ASTM F1169-19	Standard Consumer Safety Specification for Full-Size Baby Cribs	Ensured safe slat spacing, no sharp edges, and stable mattress support. Physical structure is designed to prevent entrapment and collapse.
ISO 7175-1:2019	Children's Furniture – Beds and Cots – Safety Requirements	Used as a guideline to prevent hazards such as head/limb entrapment, tip-over risks, and mechanical pinch points.

EN 716-1:2017 + A1:2019	Children's Cots – Safety Requirements (EU)	Referenced for structural durability, safe locking mechanisms, and compliant dimensions, especially relevant to the crib's swinging feature.

Table 5: Mechanical safety standards

2. Electrical Safety Standards

Standard	Title	Application in Project
IEC 62368-1:2020	Audio/Video and ICT Equipment – Safety Requirements	The system uses low-voltage (3.3V/5V) components. Protection against overcurrent and overheating has been implemented, with components enclosed in a non-conductive casing.
IEC 60335-1:2020	Household Electrical Appliances – General Requirements	Motorized swinging system and electronic circuits are isolated, with secure wiring and safe shutdown behavior in the event of a power failure.
EN 62115:2020	Electric Toys – Safety	Sound output levels are kept below 80 dB for infant safety. All electronic components are shielded and inaccessible to the child. Lighting effects are soft and non-flickering.

Table 6: Electrical Safety Standards

3. Communication Standards

Standard	Title	Application in Project
IEEE 802.11	Wireless LAN (Wi-Fi) Standards	The ESP32 microcontroller communicates with Firebase using a secure 2.4 GHz Wi-Fi connection, protected with WPA2 encryption.
IEEE 802.15.4 *(optional/future)*	Low-Rate Wireless Personal Area Networks (Zigbee, ESP-NOW)	Used when employing ESP-NOW for device-to-device communication. Data packets are small, fast, and encrypted using MAC-based pairing.
ISO/IEC 7498-1:1994	OSI Network Reference Model	Used as a reference framework to structure communication: Firebase functions at the application layer, ESP32-UART at transport and physical layers.

Table 7: Communication Standards

CHAPTER 4 Heating and Cooling

Preliminary Design and Idea Formulation

This project's initial idea was to improve baby comfort during sleep by maintaining optimal temperature control. Our focus was on developing a system that could provide both heating and cooling to always ensure the baby sleeps comfortably.

The first step in the design process was the selection of the baby bed and mattress dimensions. These measurements were very important for the design of the system components thermal regulation system for the baby bed and mattress.

The first proposed solution was to build an air conditioning system for the baby's mattress. However, this concept was not pursued due to several reasons. The use of a compressor added a considerable amount of noise that could interrupt baby sleep. Furthermore, the cost of manufacture and the complexity of the unit made the design unfeasible for a compact and safe infant bed.

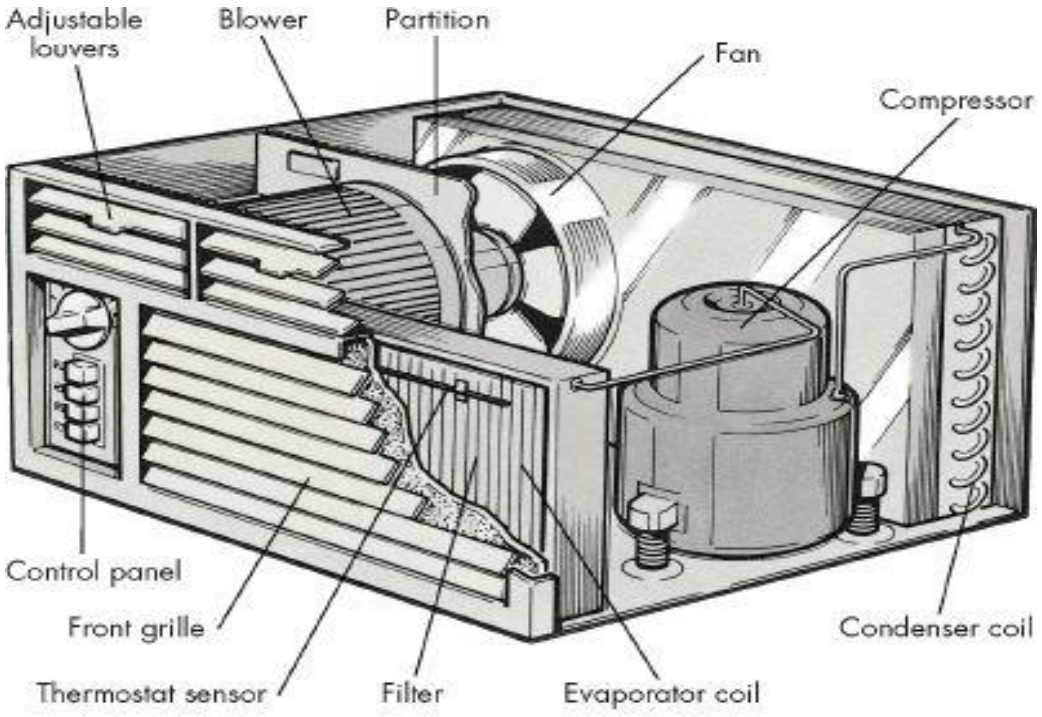


Figure 69: Air conditioning system (package system)

The "Bed Jet-Type Climate System" system is the second proposed solution. As an alternative, we looked at Bed-Jet systems which are marketed as climate control appliances for beds. They consist of a fan and air tubing that blows warm or cool air directly into the bedding. It is less intricate in design than a full air conditioning unit and

Nevertheless, we identified several limitations that made it unsuitable for our project. Although quieter than a compressor, the Bed Jet still produces noticeable noise, particularly at higher fan

speeds, which could interfere with a baby's rest. Moreover, the use of an external air hose posed a potential safety concern when used near infants (A reviewer from The Strategist noted that the Bed-Jet frequently tripped their bedroom's circuit breaker, requiring regular resets. Although customer service provided a workaround, this indicates potential compatibility issues with certain electrical systems. Additionally, a user on Reddit shared a troubling experience where they received damage), and the presence of moving parts added to the overall system complexity and potential maintenance requirements.

Figure 70: Bed-Jet Climate System

In search of a more efficient and baby-friendly alternative, we explored other technologies and ultimately identified Peltier Cell as the most suitable solution. Peltier cells offer a quieter operation, simpler design, lower cost, and safer for the baby, all while meeting the project's core objective of providing a comfortable sleeping environment for the baby.

So, what are Peltier cells?

Peltier cells, or thermoelectric coolers (TECs), are solid-state heat pumps that heat and cool based on the Peltier effect. When an electric current is passed through the junction of two unlike types of conductors or semiconductors in the cell, one will take up heat and the other release heat. This creates a temperature gradient: one side gets cold (ideal for cooling devices), and the other side gets hot (which may be used for heating or must be dissipated). The direction of heat flow can be reversed by merely reversing the direction of the current, thus one can use the same device as a heater or a cooler. Peltier cells are efficient, light, and compact for local thermal control, and they are well suited to be used in applications including portable coolers, cooling electronic components, and even heated or cooled beds or seats.

Figure 71:Peltier Cell

System Implementation through Simulation

After selecting Peltier Cells as the optimal solution for temperature regulation in the baby mattress, the next step was to translate this concept into a practical and functional design. To achieve this, we chose to begin with simulation-based development, allowing us to model and analyze the thermal behavior of the system before proceeding to physical implementation. In order to properly distribute the heat along the selected mattress we noticed that using a thin layer of aluminum foil will be helpful. The selection of the mattress dimensions, knowing the constants and boundary conditions, and selecting the number of Peltier cells was the first step to go into the simulation. We used Ansys Workbench as software for the heat transfer simulation.

In the first trial of bed heating simulation, we used the following data:

Mattress dimensions	**Constants and boundary conditions**
$lenght$: 70 cm	$k_{aluminum\ foil} = 210\frac{w}{m}.k$ $thickness_{aluminum\ foil} = 0.2cm$
$Width$: 45 cm	$k_{mattress\ sponge} = 0.06\frac{w}{m}.k$
$Thickness$: 6 cm	$T_{ambient} = 17℃$ $T_{initial} = 60℃$ $h_{air} = 5\frac{w}{m}.k$

Table 8: Constants and boundary conditions

In the first trial we used 2 cells under the mattress so the result on the upper surface was about 23℃ which is not suitable for baby warming.

The second simulation trial has some modifications in the mattress thickness (we consider it to be 3 cm). The other data was kept the same as trial 1.

The results of the second trial were acceptable for baby warming (about 29℃). The results are shown below:

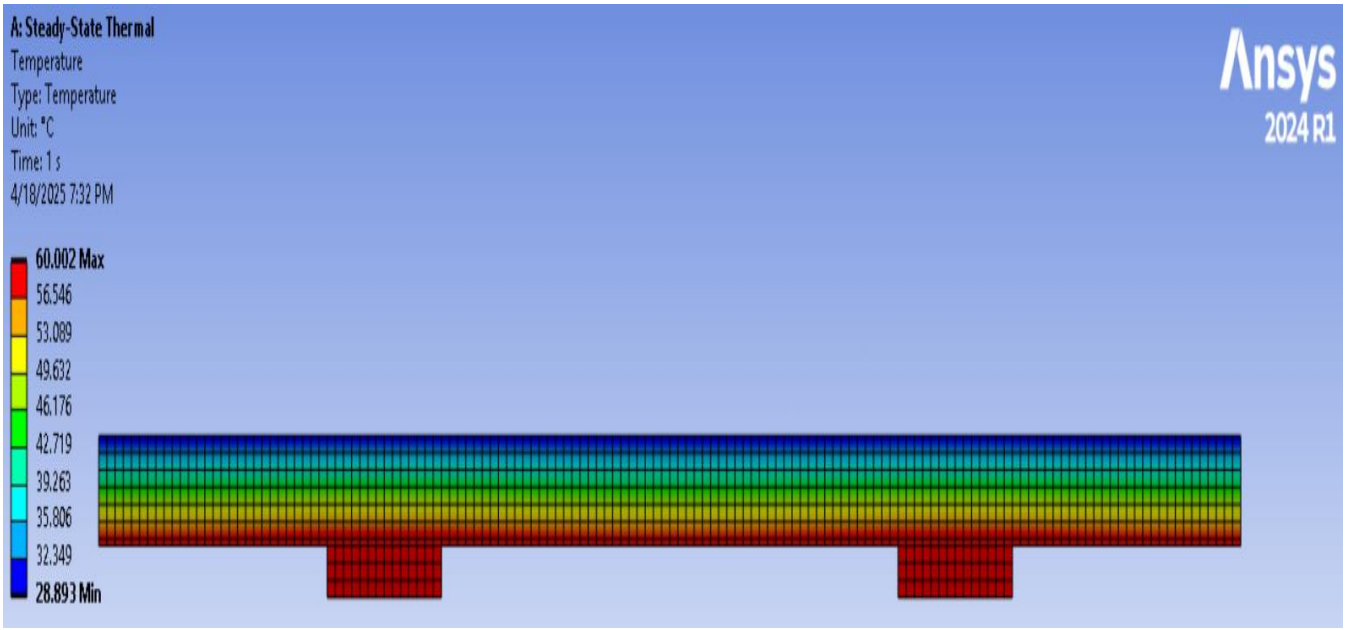


Figure 72: Ansys analysis

The results of Ansys give that the temperature at the top surface of the mattress is about 29℃. The research shows that for an adult a temperature between 30 and 35 ℃ is acceptable for heating so for a baby a 28-29 ℃ temperature is compatible.

Physical Prototype and Testing

As the simulation was done, we chose to use 2 Peltier Cells as heating and cooling sources. The mattress dimensions were:

$lenght$: 70 cm

$Width$: 45 cm

$Thickness$: 6 cm

To optimize heat transfer, we essentially split the mattress into two thermally identical sections, each measuring 3 cm thick. This arrangement enhances the provision of evenly distributed temperature regulation in the mattress body. The adjoining two sponge layers were sandwiched between a thin layer of aluminum foil (0.1 $cm\ thick$) to help evenly distribute heat across the contact area with the baby. It chose aluminum for its high thermal conductivity, which helps to dissipate heat evenly and prevents the formation of hot spots. From a safety perspective, too, the

aluminum foil is a protective factor. This is triggered in case of any electrical short circuit or power strike, the foil helps to dissipate heat and prevent direct ignition of the surrounding sponge material. This dual-purpose design enhances both the thermal performance and the safety of the mattress, making it suitable for use in infant sleep environments.

The Peltier cells we used were of $12\ V$ input and gave a temperature of $60\ ℃$ each.

The Cells will be connected directly to the power supply. The figures below will show where and how it is located.

Figure 73: Peltier cells Prototype

CHAPTER 5 Conclusion & Future work

4.1 Future Work: Enhancing Intelligence and Usability

As the Smart Baby Crib continues to evolve, several promising directions have emerged that can significantly enhance its capabilities and user experience. One notable enhancement is the addition of a live streaming system, allowing parents to visually monitor their child in real-time through the mobile application. This feature will not only improve peace of mind but also expand the crib's role as a complete monitoring solution.

Another key improvement involves enhancing sensor accuracy, particularly for health parameters like heart rate and SpO_2 levels. More reliable readings will contribute to better decision-making and ensure the system's effectiveness in critical situations.

The integration of a machine learning model for cry detection presents an exciting step toward intelligent responsiveness. By training an algorithm to distinguish between various types of infant sounds, such as hunger cries versus discomfort cries, the system could take more targeted and appropriate soothing actions. This adaptive intelligence would elevate the crib from a responsive device to a predictive caregiving assistant.

On the user interface front, upgrading to a larger screen on the main system could significantly improve usability and visibility, especially for real-time status updates and manual control. Meanwhile, developing an iOS-compatible version of the mobile application would expand

accessibility to a broader range of users, ensuring a seamless experience across all major platforms.

Together, these future upgrades represent an exciting roadmap for turning the Smart Baby Crib into a fully integrated, intelligent, and user-friendly solution for modern parenting.

4.2 Conclusion

In summary, this project has detailed the design, implementation, and validation of a Smart Baby Crib—an innovative solution that merges health monitoring, environmental control, and automated care features into one cohesive system. From wireless wearable sensors to IoT-enabled functionalities and a custom mobile app, the crib has been designed to offer both convenience and reliability to caregivers, while prioritizing the comfort and safety of infants.

The development process was marked by continuous iteration and problem-solving, whether in optimizing mechanical structures, managing power distribution, or refining communication protocols. Each design challenge led to a creative solution, ultimately resulting in a working prototype capable of monitoring vital signs, reacting to sound cues, and providing comforting responses through movement, lighting, and music.

This project embodies the spirit of applied mechatronics—bringing together mechanical design, embedded systems, wireless communication, and user interface development. It stands as a testament to what interdisciplinary collaboration and engineering innovation can achieve.

As the baton is passed to future development, the Smart Baby Crib is well-positioned to grow into an even more intelligent and essential tool for modern childcare, combining compassion with technology to meet the needs of both infants and parents in the digital age.

References

Back Then History. (n.d.). *History of baby monitors*. Back Then History. Retrieved from https://www.backthenhistory.com/articles/the-history-of-baby-monitors

HelloBaby. (n.d.). *When were baby monitors invented?* HelloBaby. Retrieved from https://www.hellobaby-monitor.com/blogs/news/when-were-baby-monitors-invented

National Center for Biotechnology Information. (2023). *The five "S's" and the "SNOO" Smart Sleeper—non-pharmacological interventions for infant soothing*. National Center for Biotechnology Information. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10485641/

Transparency Market Research. (n.d.). *Smart crib market size, industry verticals & overview - 2034*. Transparency Market Research. Retrieved from https://www.transparencymarketresearch.com/smart-crib-market.html

Healthline. (2019). *The 5 S's for baby: A guide for soothing your little one*. Healthline. Retrieved from https://www.healthline.com/health/baby/5-s-baby

Ashton, K. (2009). That 'Internet of Things' thing. *RFID Journal*. Retrieved from https://www.rfidjournal.com/expert-views/that-internet-of-things-thing

Zulkey, C. (2024, February 20). *BedJet Review 2024*. The Strategist. https://nymag.com/strategist/article/bedjet-review.html?utm_source

APPENDIX A-Student Outcome

How was it addressed in your SLP?	Where was it addressed in your SLP?
1. An ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics	
1.1 Identify complex engineering problems	Recognized the need to improve infant safety and sleep quality by automating crib monitoring and environmental control.
1.2 Formulate complex engineering problems	Applied control systems, sensor integration, and embedded system principles to develop a smart crib model.
1.3 Solve complex engineering problems	Solutions incorporated knowledge from courses such as Control Systems, Embedded Systems, and Mechatronics.
2. An ability to apply engineering design to produce solutions that meet specified needs	
2.1 Design a system to meet specific needs	Designed a baby crib with features that enhance safety, comfort, and usability, considering real-world constraints.

2.2 Modify a system to meet specific needs	Modified sensor layout and control logic after initial testing to optimize performance.
3. An ability to communicate effectively with a range of audiences	
3.1 Write a formal report	Developed a comprehensive technical report covering all stages of the project from planning to testing.
3.2 Deliver a formal presentation	Delivered progress presentations during SLP1 and SLP2 to faculty and peers.
4. An ability to recognize ethical and professional responsibilities and make informed judgments	
4.1 Consider ethical impact	Considered infant safety and privacy concerns in system design; followed safety standards.
4.2 Apply engineering standards and codes	Used standards like IEEE and ISO guidelines where applicable (e.g., sensor use, electrical safety).
5. An ability to function effectively on a team	
5.1 Plan and organize team tasks	Gantt chart and minutes document planning, task allocation, and deadline tracking.
5.2 Carry out tasks assigned by team	Individual responsibilities were met as outlined in team role assignments and weekly logs.

6. An ability to develop and conduct appropriate experimentation, analyze and interpret data	
6.1 Design experiments	Designed experiments to test sensor accuracy, environmental monitoring, and motor response.
6.2 Conduct experiments	Performed real-time testing in various conditions; data collected from sensors and microcontroller.
6.3 Draw evidence-based conclusions	Interpreted sensor data and performance metrics to validate system functionality.
7. An ability to acquire and apply new knowledge as needed	
7.1 Identify necessary skills/tools	Required learning new tools like Arduino IDE, C++, and sensor calibration techniques.
7.2 Apply self-learned skills/tools	Implemented custom code and control logic for real-time environmental monitoring.

\

APPENDIX B-Bills of Materials

SMART CRIB COMPONENTS PRICES

MAIN COMPONENTS				
COMPONENT	**Quantity**	**total Price**	**Store**	**Status**
ESP32-WROOM-32U	1	$ 9.00	robot Pi Shop	Available
SD CARD 2GB	1	$ 2.00		Available
USB TYPE B MINI	1	$ 2.00		Available
SHT30-D TEMPERATURE HUMIDITY SENSOR	1	$ 3.00	Robot Pi Shop	Available
HIGH SENSITIVITY MICROPHONE SENSOR MODULE	1	$ 3.00	Electros	Available
USB 2.0 MINI MICRO SD TF M2 MEMORY CARD READER	1	$ 1.00	Electros	Available
MINI SD CARD MP3 SOUND MODULE	1	$ 4.00	Electros	Available
SPEAKER 8 OHM 0.5W	2	$ 1.20	robot Pi Shop	Available
WIPER MOTOR 12V (NEW)	1	$ 30.00	Beb el ramel	Available
BTS7960 MOTOR DRIVER	1	$ 7.00	robot Pi Shop	Available
MAGNETIC SENSOR ENCODER | INDUCTION ANGLE MEASUREMENT SENSOR | AS5600	1	$ 1.50	Katranji	Available
THERMOELECTRIC COOLER PELTIER 12705 12V 5A CELL	2	$ 11.00	Electros	Available
RELAY				Available
ENCODER MODULE | ROTARY	1	$ 0.90	Katranji	Available
16BY 2 LCD DISPLAY	1	$ 5.00	Electros	Available
SWITCHING POWER SUPPLY 12V 30A	1	$ 12.00	robot Pi Shop	Available
CIRCUIT BREAKER				Available
WIRES THAT CAN HANDLE 20-30 A				Available

8 CHANNEL LOGIC LEVEL CONVERTER BI-DIRECTIONAL MODULE	1	$ 2.00	robot Pi Shop	Available
PCB	1	$ 3.50	robot Pi Shop	Available
40PIN 2.54MM ROUND MALE PIN HEADER	3	$ 0.30	robot Pi Shop	Available
2.54MM SINGLE ROW FEMALE 1X40 PIN HEADER STRIP BLACK	3	$ 0.99	Electros	Available
CONNECTOR | STRAIGHT FEMALE PIN HEADER | 40X1 | 3X2.54MM	2	$ 1.20	Katranji	Available
KF300 2PIN WIRE TERMINAL COPPER	4	$ 1.20	robot Pi Shop	Available
SOLDER WIRE 0.8MM 50G	1	$ 2.00	Electros	Available
CP-2515 SOLDERING DESOLDERING WIRE SOLDER WICK	1	$ 3.00	robot Pi Shop	Available
1 MM COPPER WIRED BLACK COLOR	6 m	$ 2.00	Abou samra	Available
0.5 MM COPPER WIRE BLACK AND WHITE	6m	$ 6.00		
3 IN ONE WIRE (0.5MM EACH)	5 m	$ 1.40	Abou samra	Available
TINNED COPPER JUMPER WIRE 1M	6	$ 0.90	robot Pi Shop	Available
ESP32-C3 MINI DEVELOPMENT BOARD	1	$ 5.00	Robot Pi Shop	Available
E-HEALTH SENSOR | OXIMETER | HEART-RATE | MAX30102	1	$ 1.50	Katranji	Available
MLX90614 NON-CONTACT IR TEMPERATURE SENSOR	1	$ 13.00	Electros	Available
0.96" INCH BLUE I2C IIC OLED LCD MODULE 4PIN	1	$ 4.00	Electros	Available
LITHIUM-POLYMER BATTERY | 3.7V 1100MAH | 50X40X6MM	1	$ 6.00	Katranji	Available
TP4056 LITHIUM BATTERY CHARGER MODULE	1	$ 2.50	Robot Pi Shop	Available
RESETTABLE FUSE | RXE 160 1.6A	2	$ 2.00	Katranji	Available
RESISTOR CARBON | 2W | 5% | 2K2	2	$ 0.20	Katranji	Available
RESISTOR 1.8K*2 - 1K - 4.7K - 1.2K*2	6	$ 0.30	Robot Pi Shop	Available
MT3608 2A MAX DC-DC STEP UP POWER MODULE	1	$ 1.50	Electros	Available
THREE RESISTOR (12K -30K -4.8K)	3	$ 0.15	Robot Pi Shop	Available
CRIB CONNECTOR (750 G) (20H)	2	$ 48.00		Available

BIG PIN (40 G) (1.5H)	3	$ 3.00		Available
ENCODER HOUSING (40G) (2H)	1	$ 1.50		Available
MOTOR GEAR (140G) (4.5H)	1	$		Available
MOTOR STAND (140G) (6.5H)	1	17.70		Available
SPEAKER HOLDER (100G) (3.5H)	1			Available
CRIB HANDLES (40G)(1.5 H)	2			Available
SCREEN HOLDER	1	$		Available
SMALL PARTS	-	8.00		
LOWER GEAR (100G 5.5H)	1	$ 3.00		Available
BEARINGS 6008-2RS	3	$ 15.00		Available
BEARING STANDARD 22 MM	1	$ -		Available
METAL CHASSIS	1	$ 70.00		Available
WOOD AND CRIB INTEGRATION	1	$ 200.00	201	Available
تنجيد	-	$ 80.00		Available
TOTAL	VAT	$ 1.47		
	Shipping	$ 37.00		
		$ 637.91		

APPENDIX C- MEETING MINUTES

MINUTES OF MECH 595A MEETING (1)

COLLEGE OF ENGINEERING – MME Department – RHU

Group I

ON SEPTEMBER 24th, 2019 AT 03:30 PM

Present: Ahmad Hammoudeh- Ahmad Achaji- Ziad Zok- Sleiman Smaili

Absent: none

The meeting came to order at 09:30 am.

1. Discussions and Updates

- About the type of mechanism that will be used for the soothing up and down motion
- About the smart bracelet device.
- About detecting the presence of the baby in the crib
- About the HMI system
- About the deadline

2. Advisor Comments and Recommendations

- Keep 1 mechanism out of the 2 (preferably the swing mechanism)
- Start working on the Human Machine Interface
- Approval on the smart bracelet device.
- Usage of ultrasonic sensors with the up and down mechanism (if used)
- Using the infra-red temperature sensors for (future features)

3. Expected Deliverables for Next Meeting

- Simulation for the used mechanism
- Initial control algorithm and components used (sensors/ actuators)
- HMI configuration
- (Bracelet testing)

4. Assessment

The meeting was adjourned at 09:48AM.

Minutes taken by: Ahmad Hammoudeh

MINUTES OF MECH 595A MEETING (2)

COLLEGE OF ENGINEERING – MME Department – RHU

Group 5

ON January 24th, 2025, AT 09:30 PM

Present: Ahmad Achaji - Ahmad Hammoudeh - Ziad Zok

Absent: Sleiman Smaili

The meeting came to order at 09:30 am.

5. Discussions and Updates

- About 1 swinging mechanism instead of two
- About the coupling between the metal gears and the 3d printed part
- About the motor used for the mechanism
- About the bracelet components (How to measure body temperature)
- About the Mobile application

6. Advisor Comments and Recommendations

- Using a Bigger battery in the bracelet system
- Calibrating and finding an algorithm to measure the body temperature
- Agreed on the 20 N.m actuator for the swinging mechanism and the feedback on the rotation using a magnetic encoder
- Start printing and fabricating plastic and metal components

7. Expected Deliverables for Next Meeting

- Printed part of the mechanism
- Algorithms to measure body temperature
- Entire bracelet algorithm
- Initial crib algorithm + HMI

8. Assessment

The meeting was adjourned at 09:33 AM.

Minutes taken by: Ahmad Achaji

MINUTES OF MECH 595A MEETING (3)

COLLEGE OF ENGINEERING – MME Department – RHU

Group 5

ON January 24th, 2025, AT 09:30 PM

Present: Ahmad Achaji - Ahmad Hammoudeh - Sleiman Smaili

Absent: Ziad Zok

The meeting came to order at 09:30 am.

9. Discussions and Updates

- About the number and configuration of Peltier cells for cooling and heating
- About the swing Motor's driver
- Testing of the temperature sensor and core temperature algorithm
- About the power supply system
- About the HMI

10. Advisor Comments and Recommendations

- Number of Peltier cells required and configuration
- Use of a 45Nm dc motor for swinging
- Agreed on the design and confirmation to the start implementing the system

11. Expected Deliverables for Next Meeting

- Printed part of the mechanism
- Entire bracelet algorithm
- Initial crib algorithm + HMI
- Metal and wooden parts

12. Assessment

The meeting was adjourned at 10 :00 AM.

Minutes taken by: Ahmad hammoudeh

MINUTES OF MECH 595A MEETING (4)

COLLEGE OF ENGINEERING – MME Department – RHU

Group 5

ON February 13th, 2025, AT 09:30 PM

Present: Ahmad Achaji - Ahmad Hammoudeh

Absent: Sleiman Smaili - Ziad Zok

The meeting came to order at 09:30 am.

13. Discussions and Updates

- Testing of the wiper motor (speed, direction, and power consumption)

14. Advisor Comments and Recommendations

- Changing the motor to another one that can rotate in two directions
- Agreed on the design and confirmation to the start implementing the system

15. Expected Deliverables for Next Meeting

- New wiper motor
- Entire bracelet algorithm
- Initial crib algorithm + HMI
- Metal and wooden parts

16. Assessment

The meeting was adjourned at 10 :15 AM.

Minutes taken by: Ahmad Achaji

MINUTES OF MECH 595A MEETING (5)

COLLEGE OF ENGINEERING – MME Department – RHU

Group 5

ON February 20th, 2025, AT 09:40 PM

Present: Ahmad Achaji - Ahmad Hammoudeh- Ziad Zok

Absent: Sleiman Smaili

The meeting came to order at 09:40 am.

17. Discussions and Updates

- Testing the magnetic encoder
- Approval of the metal chassis

18. Advisor Comments and Recommendations

- Testing the motor driver and the new motor.
- Testing the screen and touch feature

19. Expected Deliverables for Next Meeting

- Entire bracelet algorithm
- Initial crib algorithm + HMI
- Wooden parts

20. Assessment

The meeting was adjourned at 10 :05 AM.

Minutes taken by: Ahmad Hammoudeh

MINUTES OF MECH 595A MEETING (7)

COLLEGE OF ENGINEERING – MME Department – RHU

Group 5

ON February 27th, 2025, AT 09:30 PM

Present: Ahmad Achaji - Ahmad Hammoudeh

Absent: Sleiman Smaili - Ziad Zok

The meeting came to order at 09:30 am.

21. Discussions and Updates

- Bracelet complete algorithm
- Design updates

22. Advisor Comments and Recommendations

- Approval of the bracelet algorithm

23. Expected Deliverables for Next Meeting

- Initial crib algorithm + HMI
- Completed base and mechanism

24. Assessment

The meeting was adjourned at 09:45 AM.

Minutes taken by: Ahmad Achaji

MINUTES OF MECH 595A MEETING (8)

COLLEGE OF ENGINEERING – MME Department – RHU

Group 5

ON March 6th, 2025, AT 10:15 PM

Present: Ahmad Achaji - Ahmad Hammoudeh

Absent: Sleiman Smaili - Ziad Zok

The meeting came to order at 10:15 am.

25. Discussions and Updates

- Crib structure and mechanical part demonstration

26. Advisor Comments and Recommendations

- Approval of the mechanical prototype
- Recommendation to start with merging all the subsystems together

27. Expected Deliverables for Next Meeting

- Initial crib algorithm + HMI

28. Assessment

The meeting was adjourned at 10:30 AM.

Minutes taken by: Ahmad Hammoudeh

MINUTES OF MECH 595A MEETING (9)

COLLEGE OF ENGINEERING – MME Department – RHU

Group 5

ON March 20h, 2025, AT 9:30 PM

Present: Ahmad Achaji - Ahmad Hammoudeh

Absent: Sleiman Smaili - Ziad Zok

The meeting came to order at 9:30 am.

29. Discussions and Updates

- Bracelet integration and final design
- System integration

30. Advisor Comments and Recommendations

- Designing a PCB design for the bracelet to make it more reliable and robust
- Recommendation to start with merging all the subsystems together

31. Expected Deliverables for Next Meeting

- Initial crib algorithm + HMI
- Part of the integrated system

32. Assessment

The meeting was adjourned at 9:45 AM.

Minutes taken by: Ahmad Achaji

MINUTES OF MECH 595A MEETING (10)

COLLEGE OF ENGINEERING – MME Department – RHU

Group 5

ON March 17h, 2025, AT 9:30 PM

Present: Ahmad Achaji - Ahmad Hammoudeh- Sleiman Smaili- Ziad Zok

Absent:

The meeting came to order at 9:30 am.

33. Discussions and Updates

- System integration
- HMI problem
- Final system iteration

34. Advisor Comments and Recommendations

- Changing the HMI screen to another one that requires fewer wires
- Managing cables and electrical components in the crib.

35. Expected Deliverables for Next Meeting

- A fully functional system that represents the final iteration of the project

36. Assessment

The meeting was adjourned at 10:30 AM

Minutes taken by: Ahmad Achaji

APPENDIX D-Bills of Materials

Bracelet ESP32 Code

```
#include <Wire.h>

#include <Adafruit_GFX.h>

#include <Adafruit_SSD1306.h>

#include <Adafruit_MLX90614.h>

#include "MAX30105.h"

#include "heartRate.h"

#include "spo2_algorithm.h"

#include <esp_now.h>

#include <WiFi.h>


MAX30105 particleSensor;


#define Battery_adcPin 3

#define SCREEN_WIDTH 128

#define SCREEN_HEIGHT 64


Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

Adafruit_MLX90614 mlx = Adafruit_MLX90614();


struct SensorData {

  float ambientTempC;
```

```
  float objectTempC;
  float coreTempC;  // Calibrated core (axillary) temperature in Celsius
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////
// ADC characteristics

#define VoltageBattery_Calibration 0.71  // the lowest the higher the voltage
const int Battery_adcMaxValue = 4096;  // Arduino (1023) - Esp (4096)
const float referenceVoltage = 3.3;    // Arduino (5V) - Esp (3.3 v)
//const int numReadings_Battery = 10;    // Number of readings for averaging
#define numReadings_Battery 100  // Number of stored readings
// Voltage divider ratio
const float Battery_voltageDividerRatio = 0.25;
// LiPo battery characteristics
const float minBatteryVoltage = 3.0;  // Minimum voltage (0% charge)
const float maxBatteryVoltage = 4.2;  // Maximum voltage (100% charge)

float voltageReadings[numReadings_Battery] = {0};  // FIFO buffer for voltage values
int currentIndex = 0;  // Track the current position in the buffer
int sampleCount = 0;   // Counts the number of readings taken
float sumVoltage = 0;  // Running sum of stored values

//////insde code variables////////////////////////////////////////////////////////////////////////////////////
```

```
int HeartRate = 0;

int32_t O2 = 0;

int batteryLevel = 0;

float batteryVoltage_FV =0 ;

SensorData IR_data;

bool isConnected = false;


int Minimum_accepted_o2 = 90;

int Minimum_accepted_HR = 40;


unsigned long previousMillis = 0;  // Store the last time the function was called


////////////////////////////////////////////////////////////////////////////////////////

long irValue = 0;

//presence sensing ////////////////////////////////////////////////////////////////////////////////

bool presenceState = 0;

//Heart rate////////////////////////////////////////////////////////////////////////////////

bool Startup1 = 0;

const byte RATE_SIZE = 16;  // Increased window for averaging

byte rates[RATE_SIZE];      // Array to store heart rate samples

byte rateSpot = 0;

long lastBeat = 0;  // Time at which the last beat occurred
```

```
float beatsPerMinute = 0;
int beatAvg = 0;
// Variables for exponential moving average (low-pass filter)
float filteredBPM = 0;
const float alpha = 0.9;  // Smoothing factor (closer to 1 gives smoother, slower response)
//////o2/////////////////////////////////////////////////////////////////////////////////////////////
bool Startup2 = 0;
int count = 0;
bool skip = 0;
int false_reading_count =0;
#define MAX_BRIGHTNESS 255
#if defined(__AVR_ATmega328P__) || defined(__AVR_ATmega168__)
//Arduino Uno doesn't have enough SRAM to store 100 samples of IR led data and red led data
        in 32-bit format
//To solve this problem, 16-bit MSB of the sampled data will be truncated. Samples become 16-
        bit data.
uint16_t irBuffer[100];   //infrared LED sensor data
uint16_t redBuffer[100];  //red LED sensor data
#else
uint32_t irBuffer[100];   //infrared LED sensor data
uint32_t redBuffer[100];  //red LED sensor data
#endif
int32_t bufferLength;   //data length
```

```
int32_t spo2;           //SPO2 value
int8_t validSPO2;       //indicator to show if the SPO2 calculation is valid
int32_t heartRate;      //heart rate value
int8_t validHeartRate;  //indicator to show if the heart rate calculation is valid
//////////////////////////////////////////////////////////////////////////////

// REPLACE WITH YOUR RECEIVER MAC Address
uint8_t broadcastAddress[] = { 0xe8, 0x6b, 0xea, 0xd4, 0x7b, 0x10 };

// Optimized structure to send data via ESP-NOW that's already compatible with Arduino
typedef struct __attribute__((packed)) {
  uint16_t bodyTemp;     // Store as integer (value * 100) for 2 decimal places
  uint16_t ambientTemp;  // Store as integer (value * 100) for 2 decimal places
  uint8_t heartRate;     // BPM as uint8_t (0-255)
  uint8_t oxygenLevel;   // SpO2 as uint8_t (0-100%)
  uint8_t batteryLevel;  // Battery percentage as uint8_t (0-100%)
  uint8_t batteryVolt;   // Battery voltage * 10 for 1 decimal place
} struct_message;

// Create a struct_message called myData
struct_message myData;
esp_now_peer_info_t peerInfo;
```

```
// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup() {

  Wire.begin();  // for esp32 Wire.begin(SDA,SCL) (9,10)
  Serial.begin(11520);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("OLED initialization failed");
    while (1)
      ;
  }
  ///mlx.writeEmissivity(0.98); // Set emissivity to 0.98 for skin
  if (!mlx.begin()) {
    Serial.println("Error connecting to MLX sensor. Check wiring.");
    while (1)
      ;
  }
  Serial.println("MAX30102 Presence Sensing Example");
  if (particleSensor.begin(Wire,100000) == false)  //Use default I2C port, 100kHz speed
```

```
{
  Serial.println("MAX30102 was not found. Please check wiring/power. ");
  while (1)
    ;
}

WiFi.mode(WIFI_STA);
// Init ESP-NOW
if (esp_now_init() != ESP_OK) {
  Serial.println("Error initializing ESP-NOW");
  return;
}

// Once ESPNow is successfully Init, we will register for Send CB to
// get the status of Transmitted packet
esp_now_register_send_cb(OnDataSent);

// Register peer
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = 0;
peerInfo.encrypt = false;

// Add peer
```

```
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }
  particleSensor.setup();
  particleSensor.setPulseAmplitudeRed(0x0F);  // Increased from 0x0A
  particleSensor.setPulseAmplitudeGreen(0);   // Turn off green LED

  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(32, 32);
  display.print("Hello");
  display.display();
}
void loop() {
  irValue = particleSensor.getIR();
  presenceState = presenceSensing();
  if (presenceState) {
    IR_data = readSensorData();
    batteryVoltage_FV =batteryVoltage_2();
    batteryLevel = batteryPercentage_L();
    HeartRate = heartRateSensing();
```

```
  O2 = O2_reading_with_delay(2000);
 // Serial.print("IR=");
  //Serial.print(irValue);
  //Serial.print("  Avg BPM=");
  //Serial.print(HeartRate);
  //Serial.print("  O2=");
  //Serial.println(O2);

  //Serial.print("Ambient T=");
  //Serial.print(IR_data.ambientTempC);
  //Serial.print("  Core T=");
  //Serial.print(IR_data.coreTempC);
  //Serial.println(" *C");

  //Serial.print("V | Battery Percentage: ");
  //Serial.print(batteryLevel, 1);
  ///Serial.println("%");
  Display_and_send_with_Delay(3000);  //sending and displayin data every 3 seconds
}

else {
  display.clearDisplay();
  display.setTextSize(1.5);
```

```
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(10, 32);
    display.print("No Object detected");
    display.display();
  }
}

void Display_and_send_with_Delay(const unsigned long interval) {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;  // Update the last execution time
    drawDisplay();
    EspNow_sending();
  }
}
void drawDisplay() {

  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.print("Bat:");
  display.print(batteryLevel);
```

```
  display.print("%");


  display.setCursor(90, 0);
  display.print(batteryVoltage_FV);
  display.print("V");
 // display.print(isConnected ? "Connected" : "Disconnected");


  display.setCursor(30, 20);
  display.print("Temp:");
  display.print(IR_data.coreTempC);
  display.print(" C");


  display.setCursor(36, 35);
  display.print("HR:");
  display.print(HeartRate);
  display.print(" bpm");


  display.setCursor(37, 50);
  display.print("SpO2:");
  display.print(O2);
  display.print("%");


  display.display();
```

```
}

SensorData readSensorData() {  //temperature data
  SensorData data;
  data.ambientTempC = mlx.readAmbientTempC();
  data.objectTempC = mlx.readObjectTempC();
  data.coreTempC = calculateCoreTemp(data.objectTempC, data.ambientTempC);

  //printIRSensorData(data);
  return data;
}
float calculateCoreTemp(float foreheadTemp, float ambientTemp) {
  return 0.296 * foreheadTemp - 0.018 * ambientTemp + 28.335;
}

void printIRSensorData(const SensorData &data) {
  Serial.print("Ambient = ");
  Serial.print(data.ambientTempC);
  Serial.print(" *C\tObject (Forehead) = ");
  Serial.print(data.objectTempC);
  Serial.println(" *C");

  // Print the calibrated core temperature in Celsius
```

```
  Serial.print("Calibrated Core (Axillary) Temperature = ");
  Serial.print(data.coreTempC);
  Serial.println(" *C");
  Serial.println();
}

float batteryVoltage_2() {
  // Read the ADC value
  int adcValue = analogRead(Battery_adcPin);

  // Convert to voltage
  float measuredVoltage = (adcValue / (float)Battery_adcMaxValue) * referenceVoltage;
  float batteryVoltage = measuredVoltage / Battery_voltageDividerRatio;
  batteryVoltage /= VoltageBattery_Calibration;  // Apply calibration

  // Add new value and update the sum
  sumVoltage -= voltageReadings[currentIndex];  // Remove the oldest value from the sum
  voltageReadings[currentIndex] = batteryVoltage;  // Store new reading in FIFO buffer
  sumVoltage += batteryVoltage;  // Add the new value to the sum

  // Move to the next position in a circular manner
  currentIndex = (currentIndex + 1) % numReadings_Battery;
```

```
  // Increase sample counter
  sampleCount++;

  // Compute and return the average only when enough samples are collected
  if (sampleCount >= numReadings_Battery) {
    float averageVoltage = sumVoltage / numReadings_Battery;

    // Debugging output
    //Serial.print("ADC: ");
    //Serial.print(adcValue);
    //Serial.print(" | Raw Voltage: ");
    //Serial.print(measuredVoltage, 2);
    //Serial.print("V | Smoothed Battery Voltage: ");
    //Serial.print(averageVoltage, 2);
    //Serial.println("V");

    return averageVoltage;
  }

  return batteryVoltage_FV;  // Return -1 to indicate average is not yet ready
}
float batteryVoltage_1() {
  int totalAdcValue = 0;
```

```
  for (int i = 0; i < numReadings_Battery; i++) {
    totalAdcValue += analogRead(Battery_adcPin) ;
   // totalAdcValue += (analogRead(Battery_adcPin) * 0.9);
  }
  int adcValue = totalAdcValue / numReadings_Battery;  //average
  float measuredVoltage = (adcValue / (float)Battery_adcMaxValue) * referenceVoltage;
  float batteryVoltage = measuredVoltage / Battery_voltageDividerRatio;

  batteryVoltage = batteryVoltage / VoltageBattery_Calibration ;

  Serial.print("ADC Value: ");
  Serial.print(adcValue);
  Serial.print(" | Measured Voltage: ");
  Serial.print(measuredVoltage, 2);
  Serial.print("V | Battery Voltage: ");
  Serial.print(batteryVoltage, 2);

  return batteryVoltage;
}

int batteryPercentage_NL() {
  float batteryVoltage = batteryVoltage_1();
  int batteryPercentage = 0;
```

```
  if (batteryVoltage >= 4.2) batteryPercentage = 100.0;
  else if (batteryVoltage >= 4.15) batteryPercentage = 95 + (batteryVoltage - 4.15) * 5 / 0.05;
  else if (batteryVoltage >= 4.1) batteryPercentage = 90 + (batteryVoltage - 4.1) * 5 / 0.05;
  else if (batteryVoltage >= 4.0) batteryPercentage = 80 + (batteryVoltage - 4.0) * 10 / 0.1;
  else if (batteryVoltage >= 3.9) batteryPercentage = 60 + (batteryVoltage - 3.9) * 20 / 0.1;
  else if (batteryVoltage >= 3.8) batteryPercentage = 40 + (batteryVoltage - 3.8) * 20 / 0.1;
  else if (batteryVoltage >= 3.7) batteryPercentage = 20 + (batteryVoltage - 3.7) * 20 / 0.1;
  else if (batteryVoltage >= 3.5) batteryPercentage = 10 + (batteryVoltage - 3.5) * 10 / 0.2;
  else if (batteryVoltage >= 3.3) batteryPercentage = (batteryVoltage - 3.3) * 10 / 0.2;
  else batteryPercentage = 0;

  //to make the readings more stable
  if ((abs(batteryLevel - batteryPercentage) <= 20))
    batteryPercentage = batteryLevel;

  Serial.print("V | Battery Percentage: ");
  Serial.print(batteryPercentage, 1);
  Serial.println("%");

  return batteryPercentage;
}
int batteryPercentage_L() {
```

```
  float batteryVoltage = batteryVoltage_2() ; // Get measured voltage
  int batteryPercentage = 0;

  // Ensure voltage is within the expected range
  if (batteryVoltage < 3.3) batteryVoltage = 3.3;
  if (batteryVoltage > 4.2) batteryVoltage = 4.2;

  // Linear mapping from 3.3V - 4.2V to 0% - 100%
  batteryPercentage = ((batteryVoltage - 3.3) / (4.2 - 3.3)) * 100;

  // Stability filter (optional)
  if (abs(batteryLevel - batteryPercentage) <= 5) {
    batteryPercentage = batteryLevel;
  }

  //Serial.print("V | Battery Percentage: ");
  //Serial.print(batteryPercentage);
  //Serial.println("%");

  return batteryPercentage;
}

bool presenceSensing() {
```

```
  bool pressenceState_readings = false;
  if (irValue > 40000) {
    //Serial.println("Object detected");
    pressenceState_readings = 1;
  } else {
    pressenceState_readings = 0;
    Serial.println("Put your fingure please");
  }
  return pressenceState_readings;
}

int heartRateSensing() {

  if (Startup1 == 0) {
    // Initialize filtered BPM.
    filteredBPM = 0;
    Startup1 = 1;
  }

  int HeartRate_readings_filtered = 0;
  //long irValue = particleSensor.getIR();
  // Check if a beat is detected.
```

```
if (checkForBeat(irValue) == true) {
  // We sensed a beat!
  long delta = millis() - lastBeat;
  lastBeat = millis();
  // Calculate BPM from the time between beats.
  beatsPerMinute = 60 / (delta / 1000.0);
  // Validate BPM range.
  if (beatsPerMinute < 255 && beatsPerMinute > 20) {
    rates[rateSpot++] = (byte)beatsPerMinute;  // Store this reading.
    rateSpot %= RATE_SIZE;                     // Wrap the array index.

    // Compute average BPM over the stored samples.
    beatAvg = 0;
    for (byte x = 0; x < RATE_SIZE; x++) {
      beatAvg += rates[x];
    }
    beatAvg /= RATE_SIZE;

    // Apply an exponential moving average filter for smoother output.
    if (filteredBPM == 0) {
      filteredBPM = beatsPerMinute;  // Initialize on the first valid reading.
    } else {
      filteredBPM = alpha * filteredBPM + (1 - alpha) * beatsPerMinute;
```

```
    }
   }
  }
  // Serial.print("IR=");
  // Serial.print(irValue);
  // Serial.print(", BPM=");
  // Serial.print(beatsPerMinute, 2);
  // Serial.print(", Filtered BPM=");
  // Serial.print(filteredBPM, 2);
  // Serial.print(", Avg BPM=");
  // Serial.print(beatAvg);
  // Serial.println();

  if (beatAvg >= Minimum_accepted_HR) {  //filtering values that are less then
        'Minimum_accepted_HR'
   HeartRate_readings_filtered = beatAvg;
  }
  return HeartRate_readings_filtered;
}

int32_t O2_reading_with_delay(int Delay_iteration_nb) {  // the aim of this function is to simply
        take a acceptable value of the SPO2 and after that skip the reading of it for specific
        amount of loops
```

```
int32_t O2_radings_filtered = O2;
if (!skip) {
  int32_t O2_readings = O2value();
  if (O2_readings >= Minimum_accepted_o2) {
    O2_radings_filtered = O2_readings;
    skip = true;
    false_reading_count =0;
    Serial.println( "O2 value has captures;    Skip on");
  }
  else false_reading_count ++ ;
}
else {
  count++;
}
 Serial.print("Count false readings: ");
 Serial.println( false_reading_count);
 Serial.print("Count value: ");
 Serial.println( count);

if (false_reading_count >= (20) ){
  false_reading_count=0;
   skip = true;
   Serial.println("Skiping by force");
```

```
    }


  if (count >= Delay_iteration_nb) {
    Serial.println("reset count");
    count = 0;
    skip = false;
  }
  return O2_radings_filtered;
}
int32_t O2value() {


  if (Startup2 == 0) {
    bufferLength = 50;  //buffer length of 100 stores 4 seconds of samples running at 25sps


    //read the first 100 samples, and determine the signal range
    for (byte i = 0; i < bufferLength; i++) {
      while (particleSensor.available() == false)  //do we have new data?
        particleSensor.check();                    //Check the sensor for new data


      redBuffer[i] = particleSensor.getRed();
      irBuffer[i] = particleSensor.getIR();
      particleSensor.nextSample();  //We're finished with this sample so move to next sample
```

```
    Serial.print(F("red="));
    Serial.print(redBuffer[i], DEC);
    Serial.print(F(", ir="));
    Serial.println(irBuffer[i], DEC);
  }

  //calculate heart rate and SpO2 after first 100 samples (first 4 seconds of samples)
  maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2,
        &validSPO2, &heartRate, &validHeartRate);
  Startup2 = 1;
}

//dumping the first 25 sets of samples in the memory and shift the last 75 sets of samples to the
        top
for (byte i = 25; i < 100; i++) {
  redBuffer[i - 25] = redBuffer[i];
  irBuffer[i - 25] = irBuffer[i];
}

//take 25 sets of samples before calculating the heart rate.
for (byte i = 75; i < 100; i++) {
  while (particleSensor.available() == false)  //do we have new data?
    particleSensor.check();                    //Check the sensor for new data
```

```
    redBuffer[i] = particleSensor.getRed();
    irBuffer[i] = particleSensor.getIR();
    particleSensor.nextSample();  //We're finished with this sample so move to next sample

    //Serial.print(F(", SPO2="));
    //Serial.println(spo2, DEC);
  }

  //After gathering 25 new samples recalculate HR and SP02
  maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2,
          &validSPO2, &heartRate, &validHeartRate);
  return spo2;
}
void EspNow_sending() {
  // Convert floating point values to fixed-point integers
  myData.bodyTemp = (uint16_t)(IR_data.coreTempC * 100);     // e.g., 37.25°C becomes 3725
  myData.ambientTemp = (uint16_t)(IR_data.ambientTempC * 100); // e.g., 22.50°C becomes
          2250

  // Convert integers with bounds checking
  myData.heartRate = (HeartRate > 255) ? 255 : (uint8_t)HeartRate;
  myData.oxygenLevel = (O2 > 100) ? 100 : (uint8_t)O2;
```

```
  myData.batteryLevel = (batteryLevel > 100) ? 100 : (uint8_t)batteryLevel;

  // Convert voltage with single decimal place
  myData.batteryVolt = (uint8_t)(batteryVoltage_FV * 10); // e.g., 3.7V becomes 37

  // Send message via ESP-NOW
  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)&myData, sizeof(myData));

  if (result == ESP_OK) {
    Serial.println("Sent with success");
    isConnected = true;
  } else {
    Serial.println("Error sending the data");
    isConnected = false;
  }
}
```

Crib ESP32 Code

```
#include <WiFi.h>
#include <Firebase_ESP_Client.h>
#include <Preferences.h>
#include <time.h>
#include <HardwareSerial.h>
#include <esp_now.h>
```

```
#define TXD2 17
#define RXD2 16
HardwareSerial SerialToArduino(2);


// New structure to receive data

// Use the same structure that the sender uses
typedef struct __attribute__((packed)) {
  uint16_t bodyTemp;     // Already stored as integer (value * 100)
  uint16_t ambientTemp;  // Already stored as integer (value * 100)
  uint8_t heartRate;
  uint8_t oxygenLevel;
  uint8_t batteryLevel;
  uint8_t batteryVolt;   // Already stored as integer (value * 10)
} SensorData;
// Create a structure to hold the received data
SensorData receivedData;
// ESP-NOW callback function that will be executed when data is received
void OnDataRecv(const uint8_t *mac, const uint8_t *incomingData, int len) {
  memcpy(&receivedData, incomingData, sizeof(receivedData));

  // Print received data for debugging (converting back to floating point for display)
  Serial.print("Bytes received: ");
  Serial.println(len);
  Serial.print("Body Temperature: ");
  Serial.println(receivedData.bodyTemp / 100.0, 2);
```

```
  Serial.print("Ambient Temperature: ");
  Serial.println(receivedData.ambientTemp / 100.0, 2);
  Serial.print("Heart Rate: ");
  Serial.println(receivedData.heartRate);
  Serial.print("Oxygen Level: ");
  Serial.println(receivedData.oxygenLevel);
  Serial.print("Battery Level: ");
  Serial.println(receivedData.batteryLevel);
  Serial.print("Battery Voltage: ");
  Serial.println(receivedData.batteryVolt / 10.0, 1);
  Serial.println();
}



// ===== CONFIGURATION =====
// WiFi and Firebase credentials
const char* ssid = "Ahmadac";        //TP-Link_09F8        Ahmadac    .    hawAh
const char* password = "AhmaAd123";  // 18230888    AhmaAd123     ahmadzouhair
#define API_KEY "AIzaSyCa8Zl1aMVWQYy5v_6K0TBR4bgWNRXe9DM"
#define DATABASE_URL "https://fir-iot-9d8f7-default-rtdb.firebaseio.com/"

// NTP configuration
const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = 0;
const int daylightOffset_sec = 0;

// Application settings
const long dataInterval = 10000;  // Send data every 10 seconds
```

```
bool debugMode = true;

// Connection retry settings
const int maxWiFiAttempts = 20;
const int maxNtpRetries = 3;
const int maxFirebaseAttempts = 20;

// ===== GLOBAL OBJECTS =====
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
Preferences preferences;

// ===== STATE VARIABLES =====
// Data point tracking
int heartRateDataPoint;
int oxygenDataPoint;
int temperatureDataPoint;
int humidityDataPoint;

// Timer
unsigned long previousMillis = 0;


// Firebase paths
String CMD_manualPath = "FirebaseIOT/CMD/manualMode/";
String CMD_autoPath = "FirebaseIOT/CMD/autoMode/";
//////////////////////////////Flags
```

```
// Manual Mode Flags (as Strings)
String autoModeFlag;
String heatingCoolingFlag;
String lightFlag;
String musicFlag;
String swingingFlag;
String swingingSpeedFlag;
String temperatureSetPointFlag;
// Auto Mode Flags (as Strings)
String autoSwingingFlag;
String autoLightFlag;
String autoMusicFlag;
String autoSwingingSpeedFlag;
////////////////////////////////////




// ===== CONNECTIVITY FUNCTIONS =====
bool setupWiFi() {
  debug("Connecting to WiFi...");

  WiFi.mode(WIFI_STA);   // Fix: ensure station mode
  WiFi.setSleep(false);  // Fix: disable power save mode
  delay(500);            // Fix: small delay before starting connection
  WiFi.begin(ssid, password);

  int attempts = 0;
```

```
  while (WiFi.status() != WL_CONNECTED && attempts < maxWiFiAttempts) {
    delay(500);
    Serial.print(".");
    attempts++;
  }

  if (WiFi.status() != WL_CONNECTED) {
    debug("\n❌ WiFi connection failed after " + String(attempts) + " attempts");
    return false;
  }

  debug("\n✅ WiFi connected! IP: " + WiFi.localIP().toString());
  return true;
}
bool setupNTP() {
  debug("Configuring NTP time sync...");
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);

  // Try to sync with NTP
  int ntpAttempts = 0;
  struct tm timeinfo;
  bool timeSuccess = false;

  while (!timeSuccess && ntpAttempts < 3) {
    ntpAttempts++;
    debug("NTP sync attempt " + String(ntpAttempts) + "...");

    // Wait for time sync
```

```
    delay(500);

    if (getLocalTime(&timeinfo)) {
      timeSuccess = true;
      char timeStringBuff[30];
      strftime(timeStringBuff, sizeof(timeStringBuff), "%Y-%m-%d %H:%M:%S", &timeinfo);
      debug("✅ Time synchronized with NTP: " + String(timeStringBuff));
    } else {
      debug("⚠️ NTP time sync failed, retrying...");
    }
  }
  if (!timeSuccess) {
    debug("❌ NTP time sync failed after " + String(ntpAttempts) + " attempts");
    return false;
  }

  return true;
}
bool setupFirebase() {
  debug("Setting up Firebase connection...");

  // Configure Firebase
  config.api_key = API_KEY;
  config.database_url = DATABASE_URL;

  // Anonymous sign-in
  if (!Firebase.signUp(&config, &auth, "", "")) {
    debug("❌ Firebase authentication failed: " + fbdo.errorReason());
```

```
  return false;
}

debug("✅ Firebase authentication successful");

// Initialize connection
Firebase.begin(&config, &auth);
Firebase.reconnectWiFi(true);

// Wait for connection
int attempts = 0;
debug("Waiting for Firebase connection");
while (!Firebase.ready() && attempts < maxFirebaseAttempts) {
  Serial.print(".");
  delay(500);
  attempts++;
}

if (!Firebase.ready()) {
  debug("\n❌ Firebase connection failed after " + String(attempts) + " attempts");
  return false;
}

debug("\n✅ Firebase ready!");

// Test write to verify connection
String testPath = "FirebaseIOT/test";
if (Firebase.RTDB.setString(&fbdo, testPath, "ESP32 connected at " + getISOTimestamp())) {
```

```
    debug("✅ Firebase test write successful");
    return true;
  } else {
    debug("❌ Firebase test write failed: " + fbdo.errorReason());
    return false;
  }
}
// ===== UTILITY FUNCTIONS =====
// Print debug messages if debug mode is on
void debug(String message) {
  if (debugMode) {
    Serial.println(message);
  }
}

// Get ISO 8601 timestamp for Firebase data
String getISOTimestamp() {
  time_t now = time(nullptr);
  struct tm timeinfo;
  gmtime_r(&now, &timeinfo);
  char timeStringBuff[30];
  strftime(timeStringBuff, sizeof(timeStringBuff), "%Y-%m-%dT%H:%M:%SZ", &timeinfo);
  return String(timeStringBuff);
}

// ===== PERSISTENT STORAGE FUNCTIONS =====
void initializeCounters() {
  preferences.begin("sensor_data", false);
```

```
  heartRateDataPoint = preferences.getInt("heartRate", 1);
  oxygenDataPoint = preferences.getInt("oxygen", 1);
  temperatureDataPoint = preferences.getInt("temperature", 1);
  humidityDataPoint = preferences.getInt("humidity", 1);

  debug("✅ Counters loaded from storage:");
  debug(" - Heart Rate: " + String(heartRateDataPoint));
  debug(" - Oxygen: " + String(oxygenDataPoint));
  debug(" - Temperature: " + String(temperatureDataPoint));
  debug(" - Humidity: " + String(humidityDataPoint));
}
void resetAllCounters() {  // Reset all counters to 1
  preferences.begin("sensor_data", false);

  // Reset in-memory values
  heartRateDataPoint = 1;
  oxygenDataPoint = 1;
  temperatureDataPoint = 1;
  humidityDataPoint = 1;

  // Reset stored values
  preferences.putInt("heartRate", 1);
  preferences.putInt("oxygen", 1);
  preferences.putInt("temperature", 1);
  preferences.putInt("humidity", 1);

  preferences.end();
}
```

```
void showCurrentCounters() {
  debug("Current counter values:");
  debug(" - Heart Rate: " + String(heartRateDataPoint));
  debug(" - Oxygen: " + String(oxygenDataPoint));
  debug(" - Temperature: " + String(temperatureDataPoint));
  debug(" - Humidity: " + String(humidityDataPoint));
}

// ===== DATA FUNCTIONS =====
// Send data for a single sensor type
bool sendTimedSensorData(String sensorType, float value, String timestamp) {
  // Get the appropriate data point counter
  int dataPoint = 1;  // Default

  if (sensorType == "heartRate") {
    dataPoint = heartRateDataPoint;
  } else if (sensorType == "oxygen") {
    dataPoint = oxygenDataPoint;
  } else if (sensorType == "temperature") {
    dataPoint = temperatureDataPoint;
  } else if (sensorType == "humidity") {
    dataPoint = humidityDataPoint;
  }

  // Create Firebase path
  String path = "FirebaseIOT/readings/sensors/timed/" + sensorType + "/dataPoint" +
String(dataPoint);

  // Send value and timestamp
```

```
  bool success = false;

  if (sensorType == "temperature") {
    // Temperature is a float
    success = Firebase.RTDB.setFloat(&fbdo, path + "/value", value);
  } else {
    // All others are integers
    success = Firebase.RTDB.setInt(&fbdo, path + "/value", (int)value);
  }

  if (success && Firebase.RTDB.setString(&fbdo, path + "/timestamp", timestamp)) {
    // Update counter in preferences
    if (sensorType == "heartRate") {
      preferences.putInt("heartRate", ++heartRateDataPoint);
    } else if (sensorType == "oxygen") {
      preferences.putInt("oxygen", ++oxygenDataPoint);
    } else if (sensorType == "temperature") {
      preferences.putInt("temperature", ++temperatureDataPoint);
    } else if (sensorType == "humidity") {
      preferences.putInt("humidity", ++humidityDataPoint);
    }

    return true;
  } else {
    debug("❌ Failed to send " + sensorType + " data: " + fbdo.errorReason());
    return false;
  }
}
```

```
// Check connections and send all sensor data
void sendDataToFirebase() {
  // Check WiFi connection
  if (WiFi.status() != WL_CONNECTED) {
    debug("❌ WiFi disconnected! Reconnecting...");
    WiFi.begin(ssid, password);
    delay(3000);
    if (WiFi.status() != WL_CONNECTED) {
      debug("WiFi reconnection failed, skipping this data cycle");
      return;
    }
    debug("WiFi reconnected successfully");
  }
  // Check Firebase connection
  if (!Firebase.ready()) {
    debug("❌ Firebase not ready! Skipping this data cycle");
    return;
  }

  // Get current timestamp
  String timestamp = getISOTimestamp();

  // Generate sensor values once to be used for both raw and timed data
  int heartRate = receivedData.heartRate;
  int oxygen = receivedData.oxygenLevel;
  float temperature = receivedData.bodyTemp / 100.0, 2;
  int humidity = random(40, 80);
  bool appearance = random(0, 2) == 1;
```

```
bool crying = random(0, 2) == 1;

// Send timed sensor readings
bool success = true;
success &= sendTimedSensorData("heartRate", heartRate, timestamp);
success &= sendTimedSensorData("oxygen", oxygen, timestamp);
success &= sendTimedSensorData("temperature", temperature, timestamp);
success &= sendTimedSensorData("humidity", humidity, timestamp);

// Update raw sensor values with the same readings
String rawPath = "FirebaseIOT/readings/sensors/raw";
success &= Firebase.RTDB.setBool(&fbdo, rawPath + "/appearanceState", appearance);
success &= Firebase.RTDB.setBool(&fbdo, rawPath + "/cryingState", crying);
success &= Firebase.RTDB.setInt(&fbdo, rawPath + "/oxygen", oxygen);
success &= Firebase.RTDB.setInt(&fbdo, rawPath + "/humidity", humidity);
success &= Firebase.RTDB.setFloat(&fbdo, rawPath + "/temperature", temperature);

if (success) {
  debug("✅ All data sent successfully with consistent values:");
  debug(" - Heart Rate: " + String(heartRate) + " BPM");
  debug(" - Oxygen: " + String(oxygen) + "%");
  debug(" - Temperature: " + String(temperature) + "°C");
  debug(" - Humidity: " + String(humidity) + "%");
  debug(" - Appearance State: " + String(appearance ? "true" : "false"));
  debug(" - Crying State: " + String(crying ? "true" : "false"));
  debug("-------------------------");
  showCurrentCounters();
} else {
```

```
    debug("❌ Some data failed to send");
  }
}
void sendStatusToFirebase() {
  // Check WiFi connection
  if (WiFi.status() != WL_CONNECTED) {
    debug("❌ WiFi disconnected! Reconnecting...");
    WiFi.begin(ssid, password);
    delay(3000);
    if (WiFi.status() != WL_CONNECTED) {
      debug("WiFi reconnection failed, skipping status data");
      return;
    }
    debug("WiFi reconnected successfully");
  }

  // Check Firebase connection
  if (!Firebase.ready()) {
    debug("❌ Firebase not ready! Skipping status data");
    return;
  }

  // Simulated status values — replace with actual ones in your code
  bool automode_status = true;
  bool heatingCooling_status = false;
  bool light_status = true;
  bool swinging_status = false;
  int music_status = 2;
```

```
  float temperatureSetPoint_status = 37.5;

  bool autoModeSettings_light_status = true;
  bool autoModeSettings_swinging_status = false;
  int autoModeSettings_music_status = 1;

  // Base path for status
  String basePath = "/FirebaseIOT/readings/status";

  bool success = true;
  success &= Firebase.RTDB.setBool(&fbdo, basePath + "/automode", automode_status);
  success &= Firebase.RTDB.setBool(&fbdo, basePath + "/heatingCooling",
heatingCooling_status);
  success &= Firebase.RTDB.setBool(&fbdo, basePath + "/light", light_status);
  success &= Firebase.RTDB.setBool(&fbdo, basePath + "/swinging", swinging_status);
  success &= Firebase.RTDB.setInt(&fbdo, basePath + "/music", music_status);
  success &= Firebase.RTDB.setFloat(&fbdo, basePath + "/temperatureSetPoint",
temperatureSetPoint_status);

  // Automode settings
  String autoPath = basePath + "/automodeSettings";
  success &= Firebase.RTDB.setBool(&fbdo, autoPath + "/light",
autoModeSettings_light_status);
  success &= Firebase.RTDB.setBool(&fbdo, autoPath + "/swinging",
autoModeSettings_swinging_status);
  success &= Firebase.RTDB.setInt(&fbdo, autoPath + "/music",
autoModeSettings_music_status);

  if (success) {
    debug("✅ All status values sent successfully.");
```

```
    Serial.println("📤 Sent Status Values:");
    Serial.print("automode: ");
    Serial.println(automode_status);
    Serial.print("heatingCooling: ");
    Serial.println(heatingCooling_status);
    Serial.print("light: ");
    Serial.println(light_status);
    Serial.print("swinging: ");
    Serial.println(swinging_status);
    Serial.print("music: ");
    Serial.println(music_status);
    Serial.print("temperatureSetPoint: ");
    Serial.println(temperatureSetPoint_status);
    Serial.print("automodeSettings/light: ");
    Serial.println(autoModeSettings_light_status);
    Serial.print("automodeSettings/swinging: ");
    Serial.println(autoModeSettings_swinging_status);
    Serial.print("automodeSettings/music: ");
    Serial.println(autoModeSettings_music_status);
  } else {
    debug("❌ Some status values failed to send.");
  }
}

void ReadFlags_SendToArduino() {
  // Read manual mode flags
  readFirebaseString(CMD_manualPath + "autoModeFlag", autoModeFlag);
  readFirebaseString(CMD_manualPath + "heatingCoolingFlag", heatingCoolingFlag);
```

```
  readFirebaseString(CMD_manualPath + "lightFlag", lightFlag);
  readFirebaseString(CMD_manualPath + "musicFlag", musicFlag);
  readFirebaseString(CMD_manualPath + "swingingFlag", swingingFlag);
  readFirebaseString(CMD_manualPath + "swingingSpeedFlag", swingingSpeedFlag);
  readFirebaseString(CMD_manualPath + "temperatureSetPointFlag", temperatureSetPointFlag);
  // Read auto mode flags
  readFirebaseString(CMD_autoPath + "swingingFlag", autoSwingingFlag);
  readFirebaseString(CMD_autoPath + "lightFlag", autoLightFlag);
  readFirebaseString(CMD_autoPath + "musicFlag", autoMusicFlag);
  readFirebaseString(CMD_autoPath + "swingingSpeedFlag", autoSwingingSpeedFlag);
  // Debug log
  debug("📥 Flags updated:");
  debug("Manual -> light: " + lightFlag + ", swing: " + swingingFlag + ", music: " + musicFlag +
", speed: " + swingingSpeedFlag);
  debug("Auto   -> light: " + autoLightFlag + ", swing: " + autoSwingingFlag + ", music: " +
autoMusicFlag + ", speed: " + autoSwingingSpeedFlag);


  SendCMDToArduino();
}
void readFirebaseString(String path, String& targetVariable) {
  if (Firebase.RTDB.getString(&fbdo, path)) {
    targetVariable = fbdo.stringData();


    targetVariable.replace("\"", "");  // Remove quotes
    targetVariable.replace("\\", "");  // Remove backslashes
    targetVariable.trim();             // Remove whitespace or newlines


    debug("✅ Firebase read success - " + path + ": " + targetVariable);
  } else {
```

```
    debug("❌ Firebase read failed - " + path + ": " + fbdo.errorReason());
    targetVariable = "";
  }
}

void checkAndSendCommand_manualMode(String flagValue, String flagPath, String valuePath,
String commandName) {
  if (flagValue == "1") {
    String tempString;
    readFirebaseString(valuePath, tempString);

    if (tempString != "") {
      SerialToArduino.write(0xAA);  // Start byte
      SerialToArduino.write(0x02);  // Tag for command
      SerialToArduino.println("Manual:" + commandName + ":" + tempString);
      Firebase.RTDB.setString(&fbdo, flagPath, "0");  // Reset flag
    }
  }
}
void checkAndSendCommand_autoMode(String flagValue, String flagPath, String valuePath,
String commandName) {
  if (flagValue == "1") {
    String tempString;
    readFirebaseString(valuePath, tempString);

    if (tempString != "") {
      SerialToArduino.write(0xAA);  // Start byte
      SerialToArduino.write(0x02);  // Tag for command
      SerialToArduino.println("Auto:" + commandName + ":" + tempString);
```

```
      Firebase.RTDB.setString(&fbdo, flagPath, "0");  // Reset flag
    }
  }
}
void SendCMDToArduino() {
  // Manual commands
  checkAndSendCommand_manualMode(lightFlag, CMD_manualPath + "lightFlag", CMD_manualPath + "light", "light");
  checkAndSendCommand_manualMode(musicFlag, CMD_manualPath + "musicFlag", CMD_manualPath + "music", "music");
  checkAndSendCommand_manualMode(swingingFlag, CMD_manualPath + "swingingFlag", CMD_manualPath + "swinging", "swinging");
  //checkAndSendCommand_manualMode(swingingSpeedFlag, CMD_manualPath + "swingingSpeedFlag", CMD_manualPath + "swingingSpeed", "speed");
  //checkAndSendCommand_manualMode(temperatureSetPointFlag, CMD_manualPath + "temperatureSetPointFlag", CMD_manualPath + "temperatureSetPoint", "temperatureSetPoint");
  checkAndSendCommand_manualMode(autoModeFlag, CMD_manualPath + "autoModeFlag", CMD_manualPath + "autoMode", "autoMode");



  // Auto mode specific commands (optional)
  checkAndSendCommand_autoMode(autoSwingingFlag, CMD_autoPath + "swingingFlag", CMD_manualPath + "Swinging", "Swinging");
  checkAndSendCommand_autoMode(autoLightFlag, CMD_autoPath + "LightFlag", CMD_manualPath + "Light", "Light");
  checkAndSendCommand_autoMode(autoMusicFlag, CMD_autoPath + "MusicFlag", CMD_manualPath + "Music", "Music");
  //checkAndSendCommand_autoMode(autoSwingingSpeedFlag, CMD_autoPath+"SwingingSpeedFlag", CMD_manualPath + "SwingingSpeed", "SwingingSpeed");
}
```

```
// ===== READ COMMAND FUNCTION =====
String readFirebaseCommand(String path) {
  if (Firebase.RTDB.getString(&fbdo, path)) {
    String value = fbdo.stringData();
    debug("✅ Firebase read success - " + path + ": " + value);
    return value;
  } else {
    debug("❌ Firebase read failed - " + path + ": " + fbdo.errorReason());
    return "";
  }
}

bool offlineMode = false;
// ===== MAIN FUNCTIONS =====
void setup() {
  Serial.begin(115200);
  Serial.println("\n=== ESP32 Firebase Multi-Sensor Data Logger ===");

  // Initialize Serial connection to Arduino
  SerialToArduino.begin(9600, SERIAL_8N1, RXD2, TXD2);

  // Initialize WiFi - restart if failed
  if (!setupWiFi()) {
    debug("Restarting due to WiFi connection failure...");
    delay(1000);
    //ESP.restart();
    offlineMode = true;
```

```
}

// Initialize NTP - retry until successful
bool ntpSuccess = false;
int ntpRetries = 0;
while (!ntpSuccess && ntpRetries < maxNtpRetries) {
  ntpSuccess = setupNTP();
  if (!ntpSuccess) {
    ntpRetries++;
    debug("NTP connection failed, retry " + String(ntpRetries) + "/" + String(maxNtpRetries));
    delay(2000);
  }
}
if (!ntpSuccess) {
  debug("Restarting due to NTP failure...");
  delay(1000);
  // ESP.restart();
}

// Initialize Firebase - restart if failed
if (!setupFirebase()) {
  debug("Restarting due to Firebase connection failure...");
  delay(1000);
  //ESP.restart();
}
if (esp_now_init() != ESP_OK) {
  Serial.println("Error initializing ESP-NOW");
  return;
```

```
  }

  // Once ESPNow is successfully Init, we will register for recv CB to
  // get recv packer info
  esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));



  // Initialize persistent counters
  initializeCounters();
  debug("Setup complete! Starting data transmission...");
}

void loop() {
  if (!offlineMode) {
    unsigned long currentMillis = millis();

    // Send data at the specified interval
    if (currentMillis - previousMillis >= dataInterval) {
      previousMillis = currentMillis;
      sendDataToFirebase();
      sendStatusToFirebase();
    }

    ReadFlags_SendToArduino();

    delay(10);
  } else {
    Serial.println("Offline mode");
```

```
    // offline mode
    delay(10);
  }
}
```

Crib Arduino Code

```
#include <SoftwareSerial.h>
#include "DFRobotDFPlayerMini.h"
#include <Wire.h>

#define light 11

SoftwareSerial espSerial(9, 10);  // RX, TX

typedef struct __attribute__((packed)) {
  uint16_t bodyTemp;     // Already stored as integer (value * 100)
  uint16_t ambientTemp;  // Already stored as integer (value * 100)
  uint8_t heartRate;
  uint8_t oxygenLevel;
  uint8_t batteryLevel;
  uint8_t batteryVolt;  // Already stored as integer (value * 10)
} SensorData;

SensorData incomingData;
int setLightMode = -1;


const int speakerPin = A3;
const int NUM_SONGS = 3;  // Only 3 songs now
```

```
// SONG 0 - "Twinkle Twinkle Little Star"
int melody1[] = {
  262, 262, 392, 392, 440, 440, 392,
  349, 349, 330, 330, 294, 294, 262,
  392, 392, 349, 349, 330, 330, 294,
  392, 392, 349, 349, 330, 330, 294,
  262, 262, 392, 392, 440, 440, 392,
  349, 349, 330, 330, 294, 294, 262,
  294, 294, 262, 262, 294, 294, 330,
  330, 349, 330, 294, 262, 294, 330,
  392, 392, 349, 330, 294, 262, 262
};
int duration1[] = {
  500, 500, 500, 500, 500, 500, 1000,
  500, 500, 500, 500, 500, 500, 1000,
  500, 500, 500, 500, 500, 500, 1000,
  500, 500, 500, 500, 500, 500, 1000,
  500, 500, 500, 500, 500, 500, 1000,
  500, 500, 500, 500, 500, 500, 1000,
  500, 500, 500, 500, 500, 500, 700,
  500, 500, 500, 500, 500, 500, 1000,
  500, 500, 500, 500, 500, 800, 1000
};
// SONG 1 - "Mary Had a Little Lamb"
int melody2[] = {
  330, 294, 262, 294, 330, 330, 330, 294, 294, 294, 330, 392, 392,
  330, 330, 330, 294, 262, 262, 294, 294, 330, 330, 330, 294, 330,
  330, 294, 262, 262, 330, 330, 330, 294, 294, 294, 330, 392, 392,
```

```
  330, 330, 330, 294, 262, 262
};
int duration2[] = {
  400, 400, 400, 400, 400, 400, 800, 400, 400, 800, 400, 400, 800,
  400, 400, 400, 400, 400, 400, 800, 400, 400, 400, 400, 400, 400,
  800, 400, 400, 400, 400, 400, 400, 800, 400, 400, 400, 400, 400,
  400, 800, 400, 400, 400, 400, 400, 400
};
// SONG 2 - "Happy Birthday"
int melody3[] = {
  264, 264, 297, 264, 352, 330, 264, 264, 297, 264, 396, 352,
  264, 264, 297, 264, 352, 330, 264, 264, 297, 264, 396, 352,
  264, 264, 297, 264, 352, 330, 264, 264, 297, 264, 396, 352,
  264, 264, 297, 264, 352, 330
};
int duration3[] = {
  350, 350, 700, 700, 700, 1400, 350, 350, 700, 700, 700, 1400,
  350, 350, 700, 700, 700, 1400, 350, 350, 700, 700, 700, 1400,
  350, 350, 700, 700, 700, 1400, 350, 350, 700, 700, 700, 1400,
  350, 350, 700, 700, 700, 1400
};
// Arrays for all songs
int* melodies[NUM_SONGS] = { melody1, melody2, melody3 };
int* durations[NUM_SONGS] = { duration1, duration2, duration3 };
int melodyLengths[NUM_SONGS] = {
  sizeof(melody1) / sizeof(int), sizeof(melody2) / sizeof(int), sizeof(melody3) / sizeof(int)
};
// Playback state
```

```
int* currentMelody;
int* currentDurations;
int currentSongLength;
int currentNote = 0;
bool isTonePlaying = false;
bool isPlaying = false;
unsigned long previousMillis = 0;


// Define the BTS7960 control pins
const int RPWM = 5;  // Right PWM pin connected to Arduino pin 3
const int LPWM = 6;  // Left PWM pin connected to Arduino pin 5
const int R_EN = 7;  // Right Enable pin connected to Arduino pin 2
const int L_EN = 8;  // Left Enable pin connected to Arduino pin 4

void setup() {
  Serial.begin(115200);
  espSerial.begin(9600);
  pinMode(light, OUTPUT);
  digitalWrite(light, LOW);

  pinMode(speakerPin, OUTPUT);


  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);
  pinMode(R_EN, OUTPUT);
  pinMode(L_EN, OUTPUT);
```

```
  // Enable the motor driver
  digitalWrite(R_EN, HIGH);
  digitalWrite(L_EN, HIGH);



}

void loop() {
  if (espSerial.available()) {
    if (espSerial.read() != 0xAA) return;  // Wait for start byte

    while (!espSerial.available())
      ;
    uint8_t tag = espSerial.read();

    if (tag == 0x01) {
      while (espSerial.available() < sizeof(incomingData))
        ;
      espSerial.readBytes((char*)&incomingData, sizeof(incomingData));

      Serial.println("Arduino: Got sensor struct");
      Serial.print("Body Temp: ");
      Serial.println(incomingData.bodyTemp / 100.0, 2);
      Serial.print("Ambient Temp: ");
      Serial.println(incomingData.ambientTemp / 100.0, 2);
      Serial.print("Heart Rate: ");
      Serial.println(incomingData.heartRate);
```

```
  Serial.print("Oxygen Level: ");
  Serial.println(incomingData.oxygenLevel);
  Serial.print("Battery Level: ");
  Serial.println(incomingData.batteryLevel);
  Serial.print("Battery Voltage: ");
  Serial.println(incomingData.batteryVolt / 10.0, 1);
  Serial.println("----------------------");

} else if (tag == 0x02) {
  String cmd = espSerial.readStringUntil('\n');

  Serial.print("Arduino: Raw cmd string: [");
  Serial.print(cmd);
  Serial.println("]");

  int modeIndex = cmd.indexOf(':');
  int typeIndex = cmd.indexOf(':', modeIndex + 1);

  if (modeIndex != -1 && typeIndex != -1) {
    String mode = cmd.substring(0, modeIndex);
    String type = cmd.substring(modeIndex + 1, typeIndex);
    String action = cmd.substring(typeIndex + 1);
    action.replace("\"", "");  // Remove quotes
    action.replace("\\", "");  // Remove backslashes
    action.trim();             // Remove whitespace or newlines

    action.trim();         // Remove whitespace
    action.toLowerCase();  // Normalize
```

```
Serial.print("Mode: ");
Serial.println(mode);
Serial.print("Type: ");
Serial.println(type);
Serial.print("Action: ");
Serial.println(action);

if (mode == "Manual") {
  if (type == "light") {
    if (action == "on") {
      setLightMode = 1;
      Serial.println(" Turning ON the light.");
    } else if (action == "off") {
      setLightMode = 0;
      Serial.println(" Turning OFF the light.");
    }
  } else if (type == "music") {
    if (action == "0") {
      playSong(0);
      stopPlayback();
      Serial.println(" Turning off the music.");
    } else if (action == "1") {
      playSong(1);
      Serial.println(" Turning ON the music 1.");
    } else if (action == "2") {
      playSong(2);
      Serial.println(" Turning ON the music 2");
```

```
    } else if (action == "3") {
      playSong(3);
      Serial.println("  Turning ON the music 3");
    }
   } else if (type == "swinging") {
    if (action == "on") {

      Serial.println(" swinging is set on .");
    } else if (action == "off") {
      analogWrite(RPWM, 0);
      analogWrite(LPWM, 0);
      Serial.println(" swinging is set off");
    }
   } else if (type == "autoMode") {
    if (action == "on") {
      Serial.println(" automode is set on .");
    } else if (action == "off") {
      Serial.println(" automode is set off");
    }
   }
 } else if (mode == "Auto") {
   if (type == "light") {
    if (action == "on") {
      digitalWrite(light, HIGH);
      Serial.println(" automode_set ON the light.");
    } else if (action == "off") {
      digitalWrite(light, LOW);
      Serial.println(" automode_Turning OFF the light.");
```

```
          }
        } else if (type == "music") {
          if (action == "0") {
            Serial.println("automode_ Turning off the music.");
          } else if (action == "1") {
            Serial.println(" automode_Turning ON the music 1.");
          } else if (action == "2") {
            Serial.println(" automode_Turning ON the music 2");
          } else if (action == "3") {
            Serial.println(" automode_ Turning ON the music 3");
          }
        } else if (type == "swinging") {
          if (action == "on") {
            Serial.println(" automode_swinging is set on .");
          } else if (action == "off") {
            Serial.println("automode_ swinging is set off");
          }
        }
       }
     }
   }
  }
  updatePlayback();
}
// --- FUNCTIONS ---

void playSong(int songNumber) {
  songNumber -= 1; // Adjust so song 1 refers to index 0, song 2 to index 1, etc.
```

```
  if (songNumber >= 0 && songNumber < NUM_SONGS) {
    currentMelody = melodies[songNumber];
    currentDurations = durations[songNumber];
    currentSongLength = melodyLengths[songNumber];
    currentNote = 0;
    isTonePlaying = false;
    isPlaying = true;
  }
}


void updatePlayback() {
  if (!isPlaying) return;

  unsigned long currentMillis = millis();

  if (!isTonePlaying) {
    tone(speakerPin, currentMelody[currentNote]);
    previousMillis = currentMillis;
    isTonePlaying = true;
  }

  if (currentMillis - previousMillis >= currentDurations[currentNote]) {
    noTone(speakerPin);
    currentNote++;
    isTonePlaying = false;
```

```
    if (currentNote >= currentSongLength) {
      stopPlayback();
    }
  }
}

void stopPlayback() {
  noTone(speakerPin);
  isPlaying = false;
}
```

Web Graphs code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sensor Data Graphs</title>
    <!-- Firebase -->
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="https://www.gstatic.com/firebasejs/8.10.0/firebase-app.js"></script>
    <script src="https://www.gstatic.com/firebasejs/8.10.0/firebase-database.js"></script>
    <!-- Date adapter for Chart.js -->
    <script src="https://cdn.jsdelivr.net/npm/chartjs-adapter-date-fns"></script>
    <!-- Plugin for zooming and panning -->
    <script src="https://cdn.jsdelivr.net/npm/chartjs-plugin-zoom"></script>

    <style>
```

```
/* CSS for responsive canvas */
canvas {
    max-width: 100%;
    height: auto;
}

/* General body styling */
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 10px 5px;
    background-color: #f4f4f4;
}

.chart-wrapper {
    margin-bottom: 30px;
    background: white;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
    padding: 15px;
}

.chart-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 10px;
}
```

```
.chart-title {
    font-weight: bold;
    font-size: 18px;
    color: #333;
}

.chart-controls {
    display: flex;
    gap: 5px;
}

.chart-btn {
    padding: 5px 10px;
    background: #f0f0f0;
    border: 1px solid #ddd;
    border-radius: 4px;
    cursor: pointer;
    font-size: 14px;
    transition: all 0.2s;
}

.chart-btn:hover {
    background: #e0e0e0;
}

.chart-container {
    position: relative;
```

```
            width: 100%;
            height: 300px;
        }

        /* Media query for smaller screens */
        @media only screen and (max-width: 600px) {
            .chart-container {
                height: 250px;
            }
            .chart-btn {
                padding: 4px 8px;
                font-size: 12px;
            }
            .chart-title {
                font-size: 16px;
            }
        }
    </style>
</head>
<body>

    <!-- Temperature Chart -->
    <div class="chart-wrapper">
        <div class="chart-header">
            <div class="chart-title">Temperature (°C)</div>
            <div class="chart-controls">
                <button class="chart-btn zoom-in" data-chart="temperatureChart">+</button>
                <button class="chart-btn zoom-out" data-chart="temperatureChart">-</button>
```

```
            <button class="chart-btn reset-zoom" data-chart="temperatureChart">Reset</button>
        </div>
    </div>
    <div class="chart-container">
        <canvas id="temperatureChart"></canvas>
    </div>
</div>

<!-- Humidity Chart -->
<div class="chart-wrapper">
    <div class="chart-header">
        <div class="chart-title">Humidity (%)</div>
        <div class="chart-controls">
            <button class="chart-btn zoom-in" data-chart="humidityChart">+</button>
            <button class="chart-btn zoom-out" data-chart="humidityChart">-</button>
            <button class="chart-btn reset-zoom" data-chart="humidityChart">Reset</button>
        </div>
    </div>
    <div class="chart-container">
        <canvas id="humidityChart"></canvas>
    </div>
</div>

<!-- Oxygen Chart -->
<div class="chart-wrapper">
    <div class="chart-header">
        <div class="chart-title">Oxygen Levels (%)</div>
        <div class="chart-controls">
```

```
            <button class="chart-btn zoom-in" data-chart="oxygenChart">+</button>
            <button class="chart-btn zoom-out" data-chart="oxygenChart">-</button>
            <button class="chart-btn reset-zoom" data-chart="oxygenChart">Reset</button>
        </div>
    </div>
    <div class="chart-container">
        <canvas id="oxygenChart"></canvas>
    </div>
</div>

<!-- Heart Rate Chart -->
<div class="chart-wrapper">
    <div class="chart-header">
        <div class="chart-title">Heart Rate (bpm)</div>
        <div class="chart-controls">
            <button class="chart-btn zoom-in" data-chart="heartRateChart">+</button>
            <button class="chart-btn zoom-out" data-chart="heartRateChart">-</button>
            <button class="chart-btn reset-zoom" data-chart="heartRateChart">Reset</button>
        </div>
    </div>
    <div class="chart-container">
        <canvas id="heartRateChart"></canvas>
    </div>
</div>

<script>
    // Firebase configuration
    const firebaseConfig = {
```

```
    apiKey: "AIzaSyCa8Zl1aMVWQYy5v_6K0TBR4bgWNRXe9DM",
    authDomain: "fir-iot-9d8f7.firebaseapp.com",
    databaseURL: "https://fir-iot-9d8f7-default-rtdb.firebaseio.com",
    projectId: "fir-iot-9d8f7",
    storageBucket: "fir-iot-9d8f7.appspot.com",
    messagingSenderId: "244336984642",
    appId: "1:244336984642:web:751a956f5ee52e718c23db",
    measurementId: "G-H17FJDT07B"
  };

  // Initialize Firebase
  const app = firebase.initializeApp(firebaseConfig);
  const database = firebase.database(app);
  let charts = {};

  // Function to delete data older than 7 days
  function deleteOldData(variable) {
    const dataRef = database.ref('/FirebaseIOT/readings/sensors/timed/' + variable);
    const counterRef = database.ref('/FirebaseIOT/readings/sensors/timed/' + variable +
'/dataPointCounter');
    const currentTime = new Date().getTime();
    const sevenDaysAgo = currentTime - (7 * 24 * 60 * 60 * 1000);

    dataRef.once('value', (snapshot) => {
      let deletePromises = [];

      snapshot.forEach((childSnapshot) => {
        const dataPoint = childSnapshot.val();
        const timestamp = new Date(dataPoint.timestamp).getTime();
```

```
    if (timestamp < sevenDaysAgo) {
      let deletePromise = childSnapshot.ref.remove()
        .then(() => {
          console.log(`Deleted old data for ${variable}`);
          return 1;
        })
        .catch((error) => {
          console.error(`Error deleting old data for ${variable}: `, error);
          return 0;
        });

      deletePromises.push(deletePromise);
    }
});

Promise.all(deletePromises).then((deletedCounts) => {
    let deletedTotal = deletedCounts.reduce((sum, count) => sum + count, 0);

    if (deletedTotal > 0) {
      counterRef.transaction((currentCounter) => {
        return (currentCounter || 0) - deletedTotal;
      }).then(() => {
        console.log(`Updated dataPointCounter for ${variable} by -${deletedTotal}`);
      }).catch((error) => {
        console.error(`Error updating dataPointCounter for ${variable}: `, error);
      });
    }
```

```
        });
    });
}

// Function to fetch data and render chart with timestamp sorting
function fetchDataAndRenderChart(variable, label, unit, chartId, chartColor) {
    const dataRef = database.ref('/FirebaseIOT/readings/sensors/timed/' + variable);

    dataRef.on('value', (snapshot) => {
        const dataPoints = [];
        snapshot.forEach((childSnapshot) => {
            const dataPoint = childSnapshot.val();
            dataPoints.push({
                x: new Date(dataPoint.timestamp),
                y: dataPoint.value
            });
        });

        // Sort data points by timestamp
        dataPoints.sort((a, b) => a.x - b.x);

        renderChart(dataPoints, label, unit, chartId, chartColor);
    }, (error) => {
        console.error(`Error fetching data for ${variable}: `, error);
    });
}

// Function to render the chart with enhanced zoom controls
```

```
function renderChart(dataPoints, label, unit, chartId, chartColor) {
    if (charts[chartId]) {
        charts[chartId].destroy();
    }

    const config = {
        type: 'line',
        data: {
            datasets: [{
                label: `${label} (${unit})`,
                data: dataPoints,
                borderColor: chartColor,
                borderWidth: 2,
                fill: false,
                tension: 0.1,
                pointRadius: 3,
                pointHoverRadius: 6,
                pointBackgroundColor: chartColor
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            interaction: {
                mode: 'nearest',
                axis: 'x',
                intersect: false
            },
```

```
scales: {
    x: {
        type: 'time',
        time: {
            unit: 'day',
            tooltipFormat: 'MMM d, yyyy HH:mm',
            displayFormats: {
                hour: 'HH:mm',
                day: 'MMM d'
            }
        },
        title: {
            display: true,
            text: 'Date'
        },
        ticks: {
            source: 'data',
            maxRotation: 0,
            autoSkip: true
        }
    },
    y: {
        title: {
            display: true,
            text: `${label} (${unit})`
        },
        grace: '5%'
    }
```

```
},
plugins: {
    zoom: {
        limits: {
            x: { min: 'original', max: 'original' },
            y: { min: 'original', max: 'original' }
        },
        pan: {
            enabled: true,
            mode: 'xy',
            threshold: 10
        },
        zoom: {
            wheel: {
                enabled: true,
                speed: 0.1,
                modifierKey: 'ctrl'
            },
            pinch: {
                enabled: true
            },
            mode: 'xy',
            scaleMode: 'xy',
            speed: 0.1
        }
    },
    tooltip: {
        callbacks: {
```

```
                    title: (context) => {
                        return new Date(context[0].parsed.x).toLocaleString();
                    }
                }
            },
            legend: {
                display: false
            }
        }
    }
};

    const ctx = document.getElementById(chartId).getContext('2d');
    charts[chartId] = new Chart(ctx, config);
}

// Initialize on page load
window.onload = function() {
    // Delete old data
    deleteOldData('temperature');
    deleteOldData('humidity');
    deleteOldData('oxygen');
    deleteOldData('heartRate');

    // Initialize charts with different colors
    fetchDataAndRenderChart('temperature', 'Temperature', '°C', 'temperatureChart', 'rgba(255, 99, 132, 1)');
    fetchDataAndRenderChart('humidity', 'Humidity', '%', 'humidityChart', 'rgba(54, 162, 235, 1)');
```

```
    fetchDataAndRenderChart('oxygen', 'Oxygen Levels', '%', 'oxygenChart', 'rgba(75, 192, 192, 1)');
    fetchDataAndRenderChart('heartRate', 'Heart Rate', 'bpm', 'heartRateChart', 'rgba(153, 102, 255, 1)');

    // Add event listeners for all control buttons
    document.querySelectorAll('.zoom-in').forEach(btn => {
        btn.addEventListener('click', () => {
            const chartId = btn.getAttribute('data-chart');
            if (charts[chartId]) {
                charts[chartId].zoom(1.2);
            }
        });
    });

    document.querySelectorAll('.zoom-out').forEach(btn => {
        btn.addEventListener('click', () => {
            const chartId = btn.getAttribute('data-chart');
            if (charts[chartId]) {
                charts[chartId].zoom(0.8);
            }
        });
    });

    document.querySelectorAll('.reset-zoom').forEach(btn => {
        btn.addEventListener('click', () => {
            const chartId = btn.getAttribute('data-chart');
            if (charts[chartId]) {
                charts[chartId].resetZoom();
```

```
            }
        });
    });
};
</script>
</body>
</html>
```

Mobile application

https://github.com/ahmadachaji123/Smart-Baby-Crib-Project/blob/main/Mobile%20Application/Smarrt_Crib_300425.aia