Development of a Smart Cradle System for Enhanced Infant Care Using Raspberry Pi
Concordia University
Electrical and Computer engineering
COEN 6711
Microprocessors And Their Applications
Fall 2023

Prithvik Adithya Ravindran
40195464

Rushik Shingala
40221905

Kartik Kishor Faldu
40221660

***Abstract*—The "Development of a Smart Cradle System for Enhanced Infant Care" project introduces an innovative approach to infant care by integrating advanced technologies into a modern cradle system. This project aims to ensure infant safety, enhance sleep quality, and enable remote monitoring for parents and caregivers. By utilizing hardware components such as Raspberry Pi, sensors for temperature, humidity, motion, sound, and vital signs, and the potential application of machine learning techniques, this Smart Cradle System will maintain optimal conditions for infants and offer gentle rocking for improved sleep. A dedicated mobile application will provide real-time monitoring and control. This project draws upon related works in IoT-based baby monitoring and machine learning for emotion recognition, emphasizing its potential to innovate in the field of infant care technology. The content in the report is our work, and all sources of information, ideas, or words that are not my own have been appropriately credited and referenced.**

I. Introduction

In recent years, the field of infant care has been undergoing a profound transformation, driven by the relentless march of technological innovation. This era of progress has given rise to innovative solutions that are redefining the standards for infant well-being, safety, and the overall convenience of caregiving. Among these groundbreaking developments, the focus of this research paper is the "Development of a Smart Cradle System for Enhanced Infant Care." This system is designed to revolutionize the way we nurture and safeguard our youngest family members.

The rapid expansion of technology, with its various facets like automation, sensor technology, and mobile applications, has played a pivotal role in reshaping the landscape of infant care. These technological strides have not only created new possibilities but also presented an opportunity to create a modern cradle system that surpasses traditional models. This next-generation cradle system aspires to ensure the safety and comfort of infants, while simultaneously affording parents and caregivers a higher degree of reassurance.

The proposed Smart Cradle System is a testament to the marriage of technology and infant care. It is a fusion of advanced components, including ARM-based microcontrollers, a range of sensors, and the magic of automation. The resulting system is a versatile and adaptive cradle that diligently monitors and maintains the optimal environment for infants. It achieves this by continuously overseeing crucial factors such as temperature, humidity, air quality, and the subtleties of a baby's sleep patterns.

A primary objective of this project is to elevate infant safety to a new level. It does so by deploying smart sensors that can detect changes in the infant's surroundings and respond accordingly, ensuring a secure and nurturing environment. Furthermore, the system incorporates soothing elements like gentle rocking to promote an environment conducive to high-quality infant sleep.

In a world that often demands the simultaneous management of numerous responsibilities, the importance of remote monitoring cannot be overstated. The Smart Cradle System addresses this need by developing a dedicated mobile application that connects parents and caregivers to real-time updates and alerts. This connectivity ensures they can maintain constant vigilance over their baby's well-being even when they are not physically present.

In summary, the "Development of a Smart Cradle System for Enhanced Infant Care" represents a groundbreaking leap in the realm of infant care technology. By seamlessly integrating

advanced hardware components and potentially incorporating machine learning capabilities, this project strives to provide parents and caregivers with peace of mind, knowing their infants are in the best of hands. It not only paves the way for a safer and more comfortable caregiving experience but also sets a new standard for infant well-being in an increasingly tech-driven world.

II. Literature Review

A. IoT Based Smart Baby Monitoring System with Emotion Recognition Using Machine Learning [1]

The paper introduces a comprehensive and innovative solution in the form of an IoT-based smart baby monitoring cradle system with integrated emotion recognition, catering to the needs of busy parents who struggle to maintain constant vigilance over their infants. The system's architecture centres around a combination of controllers, particularly the NodeMCU and Raspberry Pi, which play a pivotal role in connecting and managing various essential sensors. These sensors encompass devices for detecting sounds, monitoring temperature, and capturing live video footage using cameras.

One of the standout features of this system is the user-friendly mobile application, which empowers parents to monitor their babies in real-time with ease. The application offers several critical functionalities. For example, it provides live streaming capabilities, allowing parents to keep an eye on their baby's activities from virtually anywhere. Additionally, it offers a cradle-swinging feature, enabling parents to remotely comfort their crying baby by initiating soothing motions.

The system excels in sound detection, instantly identifying when the baby is crying and promptly sending notifications to the parent's mobile device. This is a remarkable advancement in easing parental concerns about their child's well-being, especially when they are not in the immediate vicinity.

The system goes further by incorporating sophisticated emotion recognition technology. By leveraging machine learning, it can effectively analyze the infant's facial expressions, providing insights into the baby's emotional state. The paper describes how the system has been trained on a dataset with predefined emotional categories like anger, happiness, fear, crying, disgust, and surprise. It identifies these emotions through facial feature analysis, making it a valuable tool for understanding and responding to the baby's needs.

As part of the experimental validation of the system, the paper demonstrates its effectiveness in various scenarios. It connects the sensors to the controllers, ensuring that sensor data is consistently available through the Blynk server and corresponding mobile application.

Notably, the system delivers real-time humidity and temperature monitoring for the baby's surroundings. Should the temperature deviate from the predefined threshold, parents are instantly notified, ensuring that the baby remains in a comfortable environment. The paper also confirms the system's ability to recognize a crying baby and swing the cradle into action. By playing a simulated crying sound, the system reacts promptly by notifying the parents and initiating the cradle's soothing movements.

Moreover, the paper addresses face recognition, ensuring that only authorized individuals can access the baby's environment. The system sends notifications when an unknown face is detected, bolstering security and providing parents with peace of mind.

The emotion recognition aspect of the system was rigorously tested by using a machine-learning model. By analyzing video frames of the baby's face, the system can successfully identify and notify parents of various emotions, such as happiness, sadness, and surprise. This feature has the potential to deepen the parent's understanding of their baby's emotional state, allowing them to respond more effectively.

In terms of future work, the paper highlights several avenues for improvement. It suggests streamlining the system by consolidating all sensors onto a single controller, potentially reducing cost and complexity. The user interface of the system's mobile application could be further enhanced, taking into account the limitations of existing applications. Additionally, the paper proposes the development of custom dashboards for various devices to augment monitoring capabilities.

The paper concludes by indicating potential research directions. These include refining the emotion and crying detection aspects through the implementation of different machine learning algorithms to pinpoint the specific cause of a baby's distress. Furthermore, the incorporation of wearable sensors is suggested to provide more accurate monitoring of the infant's health and well-being. Overall, this paper offers a detailed and promising approach to addressing the challenges of modern parenting, offering parents a valuable tool for ensuring the safety, comfort, and emotional well-being of their babies.

B. IoT based Smart Baby Cradle System using Raspberry Pi B+ - Review [2]

The research paper titled "IoT-based Smart Baby Cradle System using Raspberry Pi B+ – Review [2] presents an innovative Internet of Things (IoT) based smart baby cradle monitoring system, designed primarily for busy or working parents. This system enables parents to monitor their infant's activities remotely, ensuring their safety and well-being. Key features include the ability to detect and recognize the baby's movements and sounds, such as crying. The system incorporates a video feed that displays the baby's current position and movements, allowing parents or caregivers to observe the infant from a distance.

Central to the system's functionality is the Raspberry Pi B+ module, which serves as the control hub for the various hardware components. The system uses a condenser microphone for detecting the baby's cries, a crucial feature for alerting parents to their infant's needs. Additionally, a Passive Infrared (PIR) motion sensor is employed to detect the baby's movements, adding another layer of monitoring.

The use of a Pi camera is particularly noteworthy, as it captures the infant's condition and motion, providing real-time visual feedback to the parents. This visual output is displayed

on a screen monitor, offering a live video feed of the sleeping baby.

Moreover, the system is equipped with various parameters for comprehensive monitoring. These include live video and sound streaming, audio playback features, and the tracking of the leisure movements of the infant. Importantly, the system also measures environmental factors such as room temperature and humidity, which are essential for assessing the baby's comfort and identifying potential issues like sleeplessness.

A standout characteristic of the system is its cry detection feature, enabling parents to listen and respond promptly to their baby's cries. This feature underscores the system's aim to provide a holistic and responsive baby monitoring solution.

Overall, the paper discusses a technologically advanced and multifaceted system that integrates various sensors and modules to offer a comprehensive monitoring solution for infants, addressing the needs of modern parenting.

III. Project Components and Software

In this implementation, we have used the following aspects and components:

A. Raspberry PI

The Raspberry Pi is a small, affordable, single-board computer developed in the United Kingdom by the Raspberry Pi Foundation. Its primary purpose is to promote the teaching of basic computer science in schools and developing countries. Since its introduction, however, it has become popular among hobbyists, DIY enthusiasts, and educators for a wide range of projects and applications beyond its educational intent.

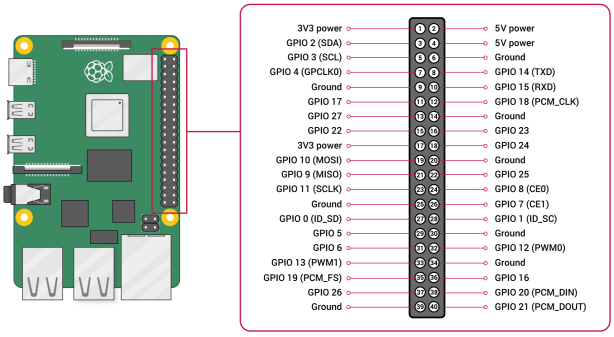

Fig. 1. Pin diagram of Raspberry Pi [5]

Key Features of Raspberry Pi: 1. Size and Form Factor: Raspberry Pi models are compact, often no larger than a credit card. This makes them highly portable and suitable for embedded applications.

2. Processing Power: Depending on the model, the Raspberry Pi comes equipped with a capable CPU, often a Broadcom system-on-a-chip (SoC) that integrates the processor, graphics, and other functionalities.

3. Memory and Storage: It includes RAM and supports external storage through SD cards or USB devices. The amount of RAM varies by model.

4. Connectivity: Raspberry Pis typically offers USB ports, HDMI output for display, and an Ethernet port for networking. Some models also include Wi-Fi and Bluetooth capabilities.

5. GPIO Pins: General Purpose Input/Output (GPIO) pins are a distinctive feature, allowing for interfacing with electronics, sensors, and actuators, which makes them ideal for physical computing projects and IoT applications.

6. Operating System: Raspberry Pi primarily runs on various Linux distributions, with Raspberry Pi OS (formerly Raspbian) being the official one. It also supports other operating systems including some versions of Windows 10, Ubuntu and IoT.

Functionality and Uses: 1. Education: Initially designed to teach computer science, it serves as a practical tool for learning programming languages like Python and Scratch.

2. DIY Projects and Hobbyist Applications: The Raspberry Pi is widely used for home automation, building media centres, gaming consoles, and personal servers.

3. Prototyping and Industry Applications: In professional settings, it's used for prototyping electronic devices, industrial automation, and IoT deployments.

4. Robotics and Electronics: Its GPIO pins allow for the creation of robots and electronic gadgets, including sensors and control systems.

5. Media and Entertainment: With the ability to run media-centre software, the Raspberry Pi is often used as a low-cost solution for streaming media. [5]

B. Camera [7]

Using a webcam with a Raspberry Pi provides a versatile and practical solution for incorporating video and audio functionalities into various projects. Webcams, which typically connect to the Raspberry Pi via USB, are highly compatible and require minimal setup. They vary in resolution and features, with many models offering HD video, and some even include autofocus and low-light enhancement.

```
/* start to read the input, push pictures into global buffer */
DBG("starting %d input plugin\n", global.incnt);
for(i = 0; i < global.incnt; i++) {
    syslog(LOG_INFO, "starting input plugin %s", global.in[i].plugin);
    if(global.in[i].run(i)) {
        LOG("can not run input plugin %d: %s\n", i, global.in[i].plugin);
        closelog();
        return 1;
    }
}

DBG("starting %d output plugin(s)\n", global.outcnt);
for(i = 0; i < global.outcnt; i++) {
    syslog(LOG_INFO, "starting output plugin: %s (ID: %02d)", global.out[i].plugin
    global.out[i].run(global.out[i].param.id);
}
```

Fig. 2. Video Stream Code

These make them ideal for a range of applications like video streaming, surveillance systems, and computer vision tasks. The Raspberry Pi's software ecosystem robustly supports webcams, offering various tools for image and video capture. The mjpg-streamer is the API used to start video feed from the webcam using WebSocket protocol [7]. The input plugin sets the device which is being used as the camera for the recording. The output plugin is responsible for run the live feed on the localhost server on a particular port address

i.e., 8080.

C. Microphone

On the audio front, a 3.5mm headphone microphone, typically connected through a USB sound card or an adapter, offers a simple solution for audio input, as the Raspberry Pi lacks a built-in microphone jack. While the audio quality depends on the microphone's specifications, this setup is generally adequate for basic recording, voice recognition, and command-based applications.

The compact and portable nature of the 3.5mm headphone mic makes it a convenient choice for small-scale and mobile projects. Together, the webcam and the headphone mic form a cost-effective and accessible means to enhance a Raspberry Pi project with audio-visual capabilities, suitable for everything from hobbyist DIY projects to more complex, interactive systems.

```
model = Model(
    [
        layers.Input(shape=(freq_bins,)),  # Input layer
        layers.Dense(int(2 * freq_bins), activation='relu'),  # First intermediate layer
        layers.Dense(int(0.75 * freq_bins), activation='relu'),  # Second intermediate layer
        layers.Dense(len(labels), activation='softmax'),  # Output layer
    ],
    labels=labels,
    low_freq=datasets[0].low_freq,
    high_freq=datasets[0].high_freq
)
# Train the model
for epoch in range(epochs):
    for i, dataset in enumerate(datasets):
        print(f'[epoch {epoch+1}/{epochs}] [audio sample {i+1}/{len(datasets)}]')
        model.fit(dataset)
        evaluation = model.evaluate(dataset)
        print(f'Validation set loss and accuracy: {evaluation}')
# Save the model
model.save(model_dir, overwrite=True)
```

Fig. 3. Code for the mic to detect crying Machine learning model is used(Training Model).

1) Machine learning algorithms for audio detection [6]: Prepare Neural Network: The model architecture is created using TensorFlow's Keras API through the micmon library.Input layer has shape (freq-bins,) representing the number of frequency bins in each sample. Two intermediate dense layers with ReLU activation functions. Output layer with softmax activation for binary classification ('negative' or 'positive'). The architecture is customizable, and the layer sizes are defined based on frequency bins.

Model Training: The model is trained for a specified number of epochs (epochs). Nested loops iterate over epochs and datasets. For each epoch and dataset, the model is trained using model.fit. Evaluation metrics (loss and accuracy) are printed for each validation set using model.evaluate.

The trained model is saved to the specified output directory (model-dir) using model.save. The trained model is saved to the specified output directory (model-dir) using model.save.

This script uses Micmon's AudioDevice to capture continuous live audio from a USB microphone connected to port 1 through ALSA. The pre-trained sound detection model is loaded, and predictions are made on the live audio.

```
model_dir = os.path.expanduser('models/sound-detect')
model = Model.load(model_dir)
audio_system = 'alsa'         # Supported: alsa and pulse
audio_device = 'plughw:1,0'   # Get list of recognized input devices with arecord -l

with AudioDevice(audio_system, device=audio_device) as source:
    for sample in source:
        # Pause recording while we process the frame
        source.pause()
        prediction = model.predict(sample)
```

Fig. 4. Code for capturing continuous live audio from mic connected to port thru ALSA

D. Tempeature Sensor

We utilized two distinct temperature sensors in our project - one for software simulation using Proteus and another for hardware implementation. This variation in sensor choice was necessitated by the availability of the sensors.

a) LM35 Temperature sensor: The LM35 temperature sensor is renowned for its precision and ease of use in a wide array of applications. It is an integrated circuit sensor that outputs a voltage linearly proportional to the Celsius temperature, eliminating the need for complex conversions. With a temperature coefficient of 10 mV/°C, it offers a straightforward interpretation of temperature readings. The sensor operates over a broad range, typically from -55°C to +150°C, and maintains high accuracy, usually within ±0.5°C at room temperature. Its low power consumption makes it ideal for battery-powered applications.

The LM35's analogue output easily interfaces with most microcontrollers' analogue-to-digital converters, facilitating its integration into various systems. This three-pin sensor (Vcc, Output, Ground) is user-friendly, requiring no external calibration or trimming, which is particularly appealing for both professional and hobbyist projects, such as environmental monitoring and HVAC systems. The LM35's combination of accuracy, simplicity, and low power consumption has cemented its status as a go-to choice for temperature measurement needs.

b) DHT11 Sensor: The DHT11 is a commonly used digital temperature and humidity sensor, appreciated for its compactness, reliability, and relatively low cost, making it a popular choice in various electronics and environmental monitoring projects. It combines a thermistor and a capacitive humidity sensor in one package, with a single digital output that can be easily read using a microcontroller like Arduino or Raspberry Pi.

The DHT11 provides a temperature range of 0 to 50°C with a ±2°C accuracy and a humidity range of 20 to 90 per cent with 5 per cent accuracy. It operates on a 3 to 5V power supply, making it compatible with most control systems. While it's not the most precise sensor available, especially when compared to its more advanced counterpart, the DHT22, the DHT11's simplicity and affordability make it an excellent choice for hobbyist projects, beginner-level experiments, and applications where extreme precision is not critical. Its single-wire serial interface simplifies the connection and data acquisition process, further contributing to its widespread use in DIY weather stations, home automation systems, and other humidity-sensitive applications. To imple-

ment the DHT11 sensor temperature and humidity values, the library Adafruit-DHT is used. The Adafruit-DHT.read-retry(DHT11, pin-number) function is responsible for reading the temperature and humidity values from the sensor. The values are then transmitted to blynk server by using BlynkLib library. Blynk.virtual-write() is the function to write the data on the virtual pins of the gauge used in the blynk server. The notification are triggered when the values goes beyond the threshold value using the blynk.log-event() function

```
while True:
    humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)

    if humidity is not None and temperature is not None:
        current_time = time.strftime('%Y-%m-%d %H:%M:%S')
        with open(log_file, 'a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow([current_time, temperature, humidity])
        print(f'Recorded: {current_time}, Temperature: {temperature:.1f}°C,
        blynk.virtual_write(1, temperature)  # Send temperature data to V1
        blynk.virtual_write(0, humidity)     # Send humidity data to V0
```

Fig. 5. DHT11 Code

E. Servo Motor (Cradle)

A servo motor is a type of rotary actuator that provides precise control of angular position, velocity, and acceleration. It consists of a suitable motor coupled to a sensor for position feedback, along with a sophisticated controller. Servo motors are widely used in applications such as robotics, CNC machinery, and automated manufacturing for their ability to accurately control the movement of mechanical components. Unlike a standard electric motor, which rotates continuously, a servo motor is capable of precise position control.

```
import RPi.GPIO as GPIO
import time

# Set GPIO numbering mode
GPIO.setmode(GPIO.BOARD)

# Set pin 11 as an output, and set servo1 as pin 11 as PWM
GPIO.setup(13,GPIO.OUT)
servo1 = GPIO.PWM(13,50) # Note 11 is pin, 50 = 50Hz pulse
```

Fig. 6. GPIO pin 13 is set for servo motor input pin at 50 Hz frequency.

```
print(prediction)
if prediction == "positive":
    source.resume()
    # start PWM running, but with value of 0 (pulse off)
    servo1.start(0)
    print("Waiting for 2 seconds")
    time.sleep(2)

    # Let's move the servo!
    print("Rotating 180 degrees in 10 steps")

    # Define variable duty
    duty = 2

    # Loop for duty values from 2 to 12 (0 to 180 degrees)
    while duty <= 12:
        servo1.ChangeDutyCycle(duty)
        time.sleep(1)
        duty = duty + 1
```

Fig. 7. Here If the prediction is positive detected servo motor will start from 0 to 180 degrees in 10 steps, by changing the ChangedutyCycle function input parameter from 2 to 12.

This is achieved through a control signal that determines the movement of the motor shaft to a specific position, which the motor then holds until a new command is received. Servo motors are typically small, efficient, fast, and available in different sizes and torque ratings to suit a variety of applications. They are characterized by their three-wire connection: power, ground, and control. The control wire receives pulse width modulation (PWM) signals, which dictate the motor's position and speed. Due to their precision and reliability, servo motors are essential in many advanced technological applications, from simple hobbyist projects to complex industrial systems.

The servo motor is a replica of the cradle which we have meant to use in real life.

F. Blynk Server and IOT application

Blynk is a prominent Internet of Things (IoT) platform known for providing a comprehensive suite for building and managing IoT applications. It offers a user-friendly environment where users can create, deploy, and manage IoT applications with minimal programming knowledge. The platform includes a digital dashboard where users can design interfaces for their projects by dragging and dropping widgets. These widgets can display or control various IoT devices and can be linked to a wide range of hardware models via the Blynk server, which bridges the user's smartphone and hardware.

A key feature of Blynk is its server, which can be either cloud-based or locally hosted, providing flexibility in terms of data security and accessibility. The platform supports a vast array of hardware platforms, including Arduino, Raspberry Pi, ESP8266, ESP32, and many others, making it highly versatile.

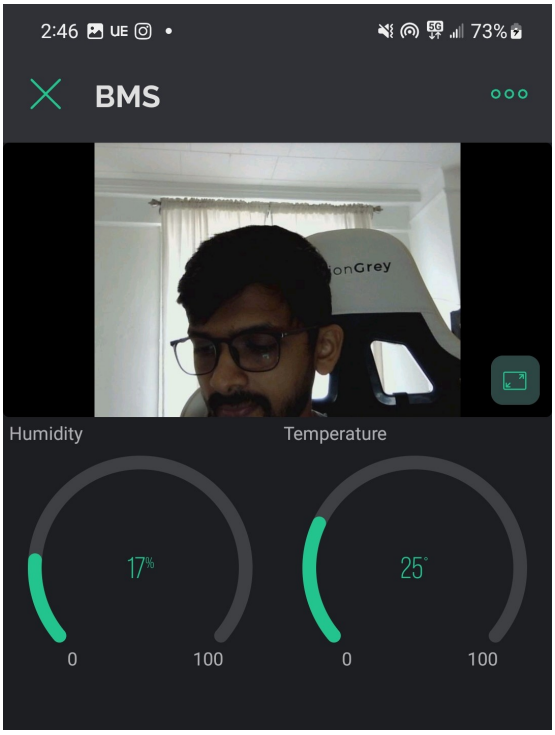

Fig. 8. IoT App

In comparison to its competitors, Blynk stands out for its exceptional ease of use, making IoT accessible even to

beginners. Its visually intuitive approach to app creation, combined with a robust server-client architecture, allows for rapid development and deployment of IoT projects. This ease of use does not come at the expense of power or flexibility, as Blynk also provides advanced features for seasoned developers. The ability to choose between a cloud server or a private server offers a balance between convenience and control over data privacy, catering to a wide range of user preferences. Additionally, Blynk's active community and comprehensive documentation provide strong support to its users, further enhancing its appeal in the rapidly growing field of IoT.

G. Proteus Simulation Software 8.12

Proteus 8.12 is a comprehensive software suite used for the simulation and design of electronic circuits and systems. Developed by Labcenter Electronics, it stands out for its ability to simulate the interaction between software running on a microcontroller and any analog or digital electronic device connected to it. This version, like its predecessors, combines a powerful schematic capture module with an advanced PCB design tool and a unique feature set for simulating microcontroller-based projects. Users can simulate not only the circuitry but also the firmware within a graphical environment.

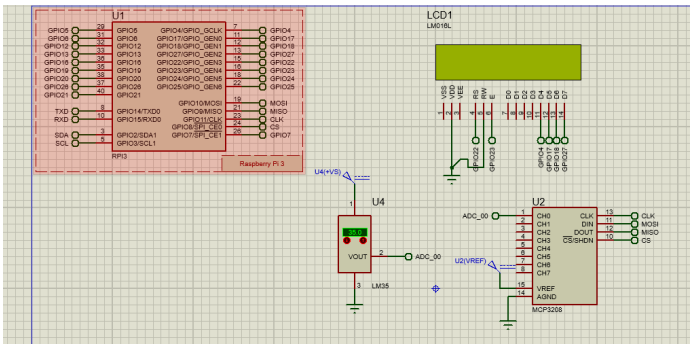

Fig. 9. Proteus Software Simulation

Proteus 8.12 supports a wide range of popular microcontrollers and is particularly known for its user-friendly interface, making it accessible to both professionals and hobbyists. It is widely used in the educational sector for teaching electronics and microcontroller systems, as well as by professionals for designing, testing, and troubleshooting complex circuits before physical prototyping. The ability to simulate real-time interactions between software and hardware makes it a valuable tool in the development process, reducing both the time and costs associated with physical prototyping.

IV. Project Working

To create this baby monitoring system using a Raspberry Pi 4 module, we have tried to simulate this setup both virtually on Proteus IDE and in real-time using appropriate hardware configurations with the components i.e. DHT11 sensor, Raspberry Pi Cam/Webcam, Mic and Servo motor. The connections between the sensors and the Raspberry Pi were made i.e., The

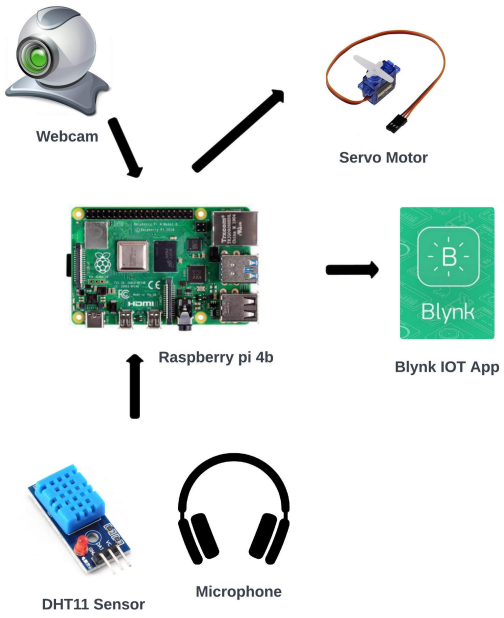

Fig. 10. Hardware Setup

DHT11 sensor's data pin is connected to GPIO4 and Webcam and Microphone are connected with USB ports.

The following steps were followed in the implementation of the system on Proteus IDE:

- Create a project file on the Proteus IDE for the programming of the microcontroller unit.
- Connecting all the components i.e. temperature sensor, and LCD, with the virtual connecting lines drawn to the respective GPIO pins on the micro-controller unit.
- Check if the connections were appropriately made according to the system configurations.
- Upload the respective code with the defined derivative that is to be taken from the sensors with which the output can be accessed and defined accurately.

The system is also implemented in real-time hardware setup. The following steps were followed to implement on hardware:

- Connect the sensors and audio/visual components correctly on the Raspberry Pi.
- Check for on which device card is the audio component connected by using "record -l" in the terminal of Raspberry Pi.
- Check for the video driver at which the webcam is connected by using the "v4l2-ctl –list-devices" command on the terminal of Raspberry Pi.
- Now execute the scripts of the DHT11 sensor, controlling motor based on crying detection and mjpeg-streamer, to start the whole baby monitoring system.

V. Challenges

a) ***Power Management***

Challenge: Ensuring a consistent and stable power supply

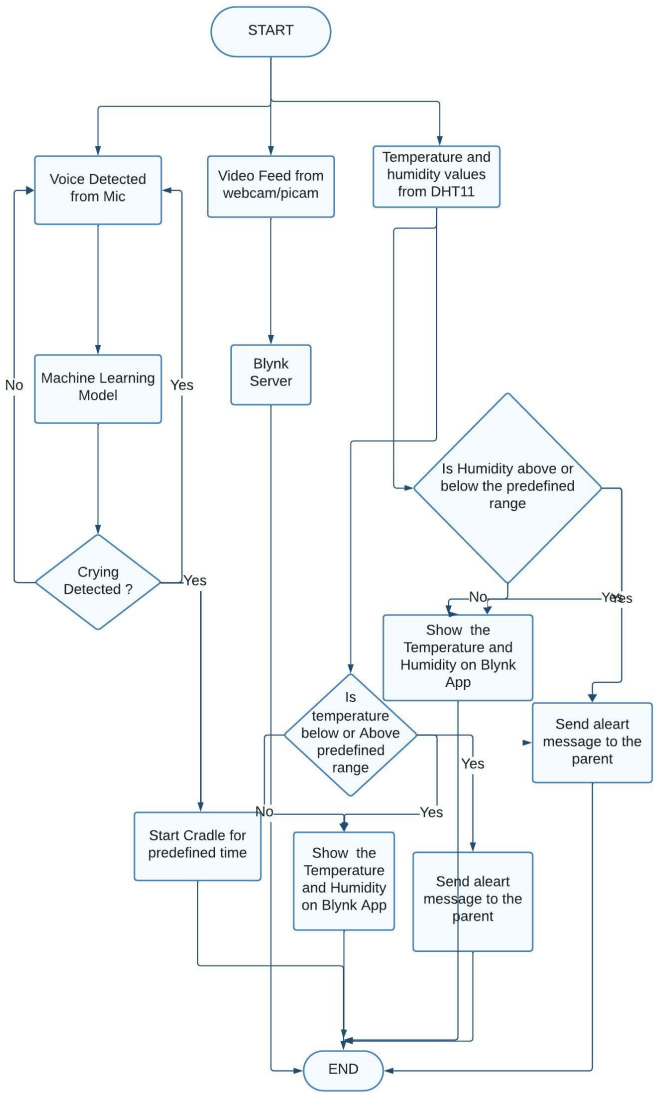

Fig. 11. Dataflow

for the servo motor is crucial in any robotics or automation project. Servo motors can draw a significant amount of current, especially under load, and an unstable power supply can lead to erratic behaviour or even damage to the motor.

Solution: This involved selecting a power supply with adequate current rating and stability. Additionally, we had to carefully choose the correct GPIO pins for the DHT11 sensor, which is sensitive to voltage levels for accurate temperature and humidity readings.:

b) ***Lack of Dataset Availability for Machine Learning*** *Challenge: Machine learning models, particularly in unique applications, often require large and diverse datasets for training. Finding or creating a suitable dataset for the specific use case like crying detection was challenging.*

Solution: We created a dataset, albeit with only 10 samples. While this is small for a machine learning model and might not provide high accuracy or generalization, it's a starting point for proof of concept.:

c) ***Compatibility Issues with Raspbian OS*** *Challenge: Raspbian OS, while widely used for Raspberry Pi projects, might have limitations or compatibility issues with certain advanced machine learning libraries like TensorFlow and TensorFlow Lite.*

Solution: Switching to Ubuntu OS, which has broader support for these libraries, indicates a strategic pivot to ensure compatibility and performance of the machine learning algorithms.:

d) ***Integration Issues with Pi Cam*** *Challenge: The initial idea to use a Pi Cam for the surveillance system faced a roadblock due to the lack of available libraries in our case.*

Solution: Substituting the Pi Cam with a standard webcam, while a compromise, demonstrates flexibility. This change likely required adjustments in our code and hardware setup to accommodate the different specifications and capabilities of a webcam.:

e) ***Adafruit Module Library Issue in Proteus*** *Challenge: In the simulation environment of Proteus 8, the unavailability of the Adafruit library posed a problem. This library is often used for interfacing various sensors and modules with microcontrollers in a simulated environment.*

Solution: We adapted by using the LM35 module for simulation purposes. This required us to write different codes for the software simulation and the actual hardware implementation, showcasing the challenges of aligning simulated environments with real-world conditions.:

f) ***Improving the model efficiency*** *Challenge: Sound sensors, especially for specific tasks like detecting crying, can be prone to false positives due to background noise or other sounds. This can lead to unnecessary triggering of the cradle (servo motor).*

We analyzed our crying detection system using a confusion matrix. Parameters for the matrix are:
True Positive (TP): *The system correctly identifies a crying sound.*
True Negative (TN): *The system correctly identifies a non-crying sound.*
False Positive (FP): *The system incorrectly identifies a non-crying sound as crying.*
False Negative (FN): *The system fails to identify a crying sound.*
The high number of false positives underscores the need for a more robust and sophisticated approach to sound detection.

We conducted a test on live audio, making it impractical to generate a confusion matrix in Python. Instead, we manually performed the assessment 30 times and identified a noteworthy

issue with false positives. Specifically, there were instances where playing a song resulted in declassification as crying. Conversely, false negatives were negligible in comparison. In our efforts to address this concern, we implemented the following changes:

Improvements: *First of all, we used a frequency sample instead of raw audio so the model is better than the traditional one.*

Reduce Frequency range: *We have fine-tuned the frequency range to focus specifically on the bandwidth of interest, which is between 250 Hz and 2.5 kHz.*
Increase Sample duration: *As we have longer audio data we have increased sample duration.*
Increase no of bins: *If the number of bins is increased, more frequency samples,so the model can give a more accurate result.*
Increase size of datasets: *Dataset is a key factor in ML model for accuracy.*
Improve the quality of dataset: *Quality means a dataset is created in a silent or noisy environment.*
Following the implementation of the aforementioned improvements, we observed a substantial reduction in false negatives. This positive outcome signals that our model has reached a level of readiness suitable for real-time applications.:

VI. Novelty and difference from other papers

a) IoT Based Smart Baby Monitoring System with Emotion Recognition Using Machine Learning:
Feature Analysis:
The current system does not employ a voice-based machine learning algorithm. This implies that while the system may have the capability to monitor various parameters, it is not taking advantage of the potential that lies in analyzing an infant's vocal patterns. Voice patterns can be incredibly telling, providing insights into an infant's emotional state or needs. The absence of such a feature means the system might miss out on subtle cues that could indicate distress, discomfort, or other emotional states of the baby that are often expressed through vocalization.
Advancement Proposal:
The proposed advancement is to incorporate machine learning algorithms that can process and analyze voice data. By implementing these models, the system would likely be able to detect not just cries but the nuances in those cries, potentially distinguishing between different needs or states of distress. This could significantly enhance the system's responsiveness, allowing for more nuanced and appropriate reactions from caregivers or automated systems, thereby offering a more advanced level of infant care.:

b) IoT-based Smart Baby Cradle System using Raspberry Pi B+:
Feature Analysis:
This cradle system is designed to use GSM technology for communication, which can be seen as a feature that adds extra hardware requirements. GSM modules enable cellular communication, allowing the cradle to send alerts through SMS or calls. However, relying on GSM could introduce complexity and dependence on cellular network availability, which might not be consistent in all locations. Additionally, GSM modules can consume more power and may require separate SIM management.
Advancement Proposal:
To modernize communication capabilities, the table suggests using built-in WiFi as an alternative to GSM. WiFi is ubiquitous in many homes and has the advantage of being easily integrated into existing networks without the need for additional cellular service plans. It would enable the system to send alerts through the internet, including push notifications to an app, emails, or other web-based communication methods. This change would likely make the system more accessible, cost-effective, and easier to maintain while providing real-time updates and alerts.:

c) Machine learning-based Smart Surveillance System: : **Feature Analysis:**
The existing surveillance system is described as recording continuously, which may lead to excessive storage consumption. Traditional surveillance systems often record all footage, regardless of its relevance, which can result in large volumes of unnecessary data. This not only consumes substantial storage space but also makes the process of reviewing footage for important events time-consuming and inefficient.
Advancement Proposal:
The advancement aims to make the surveillance system more intelligent by incorporating 24/7 tracking that is both trackable and viewable. This suggests the use of machine learning algorithms to actively analyze the video feed in real time, allowing the system to identify and record only those events that are out of the ordinary or require attention. Such a system would significantly reduce the amount of data stored and make it easier for users to access and review important footage. It would also allow for features such as event-based notifications and the ability to remotely access live or recorded video, enhancing the system's utility and user-friendliness.

The advancements adapted aim to leverage the capabilities of machine learning and WiFi technology to overcome the limitations of current IoT systems in infant care and surveillance. By doing so, they propose to create systems that are not only smarter and more responsive but also more efficient and easier to integrate into users' daily lives.

VII. Scope of Further improvements

a) Involving more sensors: To enhance the system's efficiency and functionality, we can pulse sensor, water sensor and even an oxygen monitoring sensor. The inclusion of these sensors can help to monitor the health of the baby and can alert the health authorities immediately in case of any emergency.

b) Speaker Intergration: When a baby starts crying it's difficult to calm them down so the addition of a speaker can be a useful improvement. Whenever the crying of the baby is detected the speaker will play a piece of calm music, which will help the baby fall asleep. This speaker even will have a set of playlists, so that the baby doesn't get bored by listening to one song every time.

c) ML model enhancement: The current model was trained with 10 audio samples of baby crying in an ideal environment with very good efficiency of the model. If we train the model with around 100-200 audio samples with both ideal and noisy environments, will enhance the ML model for the cry detection of the baby.

d) Embedding the sensors: The Cradle of the baby has a cloth or a small bed sheet. So we propose that the sensors detecting heartbeat, oxygen and even wetness be installed on the cloth of the cradle, which will give more precise and accurate results than the current model we have developed. This model can also be linked with smart-home appliances to monitor babies on multiple devices.

VIII. Conclusion

The Development of a Smart Cradle System for Enhanced Infant Care project stands at the forefront of technological innovation in infant care. By harmoniously integrating advanced hardware components like ARM-based Raspberry Pi, a variety of sensors, and the potential of machine learning, this system redefines the paradigm of infant safety, comfort, and monitoring.

The Smart Cradle System's ability to maintain optimal environmental conditions, coupled with features like gentle rocking and real-time remote monitoring through a dedicated mobile app, ensures not only the well-being of infants but also offers invaluable peace of mind to parents and caregivers. This project exemplifies the transformative power of technology in enhancing the standards of care and safety in the early stages of life.

It is a significant step towards a future where technology and caregiving merge seamlessly, offering solutions that are not only innovative but also deeply attuned to the fundamental needs of infant care. As we embrace this era of technological progress, the Smart Cradle System emerges as a beacon of advancement in infant care, setting new benchmarks for safety, convenience, and overall quality of life for our youngest and most vulnerable.

References

[1] Hina Alam, Muhammad Burhan, Anusha Gillani, Ihtisham ul Haq, Muhammad Asad Arshed, Muhammad Shafi, Saeed Ahmad, "IoT Based Smart Baby Monitoring System with Emotion Recognition Using Machine Learning," Wireless Communications and Mobile Computing, vol. 2023, Article ID 1175450, 11 pages, 2023. [DOI](https://doi.org/10.1155/2023/1175450).

[2] N. Saude and P. A. H. Vardhini, "IoT-based Smart Baby Cradle System using Raspberry Pi B+," 2020 International Conference on Smart Innovations in Design, Environment, Management, Planning and Computing (ICSIDEMPC), Aurangabad, India, 2020, pp. 273-278. [DOI](https://doi.org/10.1109/ICSIDEMPC49020.2020.9299602) .

[3] Kaur, Amandeep, and Ashish Jasuja. "Health monitoring based on IoT using Raspberry PI." 2017 International conference on computing, communication and automation (ICCCA). IEEE, 2017.

[4] Headphone Microphone and Webcam Used: https://www.logitech.com/en-ca/products/headsets/pro-personal-video-collaboration-kit.html

[5] Raspberry PI Description and Pinout: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

[6] Machine Learning Algorithm Reference: https://github.com/BlackLight/micmon

[7] Video Streaming API: https://github.com/jacksonliam/mjpg-streamer

Appendix

Contents

Machine learning model for detecting crying and start cradle

```
import os
from micmon.audio import AudioDirectory, AudioFile
from micmon.dataset import DatasetWriter

basedir = os.path.expanduser('~/datasets/sound-detect')
audio_dir = os.path.join(basedir, 'audio')
datasets_dir = os.path.join(basedir, 'data')
desired_num_frequency_bins = 200
cutoff_frequencies = [250, 2500]

# Scan the base audio_dir for labelled audio samples
audio_dirs = AudioDirectory.scan(audio_dir)

# Save the spectrum information and labels of the samples to a
# different compressed file for each audio file.
for audio_dir in audio_dirs:
    dataset_file = os.path.join(datasets_dir, os.path.basename(audio_dir.path) +
'.npz')
    print(f'Processing audio sample {audio_dir.path}')

    with AudioFile(audio_dir.audio_file, audio_dir.labels_file) as reader, \
            DatasetWriter(dataset_file,
                          low_freq=cutoff_frequencies[0],
                          high_freq=cutoff_frequencies[1]) as writer:
        # Calculate the frequency bin width based on the desired number of bins
        bin_width = (cutoff_frequencies[1] - cutoff_frequencies[0]) /
desired_num_frequency_bins
        for sample in reader:
            # Use the calculated bin width to set the frequency bins
            writer.set_bins(bin_width)
```

```
            writer += sample
```

Explanation: Iterate through each audio directory, and for each one:
Define the path for storing the dataset file for that audio sample (dataset_file), and Increased num_frequency_bins to 200. Display a message indicating the current audio sample being processed. Use the AudioFile and DatasetWriter from micmon to handle reading and writing audio data. Iterate through each audio sample in the current audio directory (for sample in reader). Append the audio sample and its associated labels to the dataset writer (writer += sample).
This code essentially organizes and processes audio samples, converting them into a format suitable for training machine learning models. The provided micmon library handles the intricacies of reading and writing audio data.

Training the dataset

```
import os
from tensorflow.keras import layers

from micmon.dataset import Dataset
from micmon.model import Model

# This is a directory that contains the saved .npz dataset files
datasets_dir = os.path.expanduser('~/datasets/sound-detect/data')

# This is the output directory where the model will be saved
model_dir = os.path.expanduser('~/models/sound-detect')

# This is the number of training epochs for each dataset sample
epochs = 10

# Load the datasets from the compressed files.
# Use 100% of the data points for training
datasets = Dataset.scan(datasets_dir)
labels = ['negative', 'positive']
freq_bins = len(datasets[0].samples[0])

# Create a network with 4 layers (one input layer, two intermediate layers, and
one output layer).
# The first intermediate layer in this example will have twice the number of
units as the number
# of input units, while the second intermediate layer will have 75% of the number
of
# input units. We also specify the names for the labels and the low and high-
frequency range
# used when sampling.
```

```
model = Model(
    [
        layers.Input(shape=(freq_bins,)),  # Input layer
        layers.Dense(int(2 * freq_bins), activation='relu'),  # First
intermediate layer
        layers.Dense(int(0.75 * freq_bins), activation='relu'),  # Second
intermediate layer
        layers.Dense(len(labels), activation='softmax'),  # Output layer
    ],
    labels=labels,
    low_freq=datasets[0].low_freq,
    high_freq=datasets[0].high_freq
)
# Train the model
for epoch in range(epochs):
    for i, dataset in enumerate(datasets):
        print(f'[epoch {epoch+1}/{epochs}] [audio sample {i+1}/{len(datasets)}]')
        model.fit(dataset)
        evaluation = model.evaluate(dataset)
        print(f'Validation set loss and accuracy: {evaluation}')
# Save the model
model.save(model_dir, overwrite=True)
```

Prepare Neural Network: The model architecture is created using TensorFlow's Keras API through the micmon library.Input layer has shape (freq_bins,) representing the number of frequency bins in each sample. Two intermediate dense layers with ReLU activation functions. Output layer with softmax activation for binary classification ('negative' or 'positive'). The architecture is customizable, and the layer sizes are defined based on frequency bins.
Model Training: The model is trained for a specified number of epochs (epochs). Nested loops iterate over epochs and datasets. For each epoch and dataset, the model is trained using model.fit. Evaluation metrics (loss and accuracy) are printed for each validation set using model.evaluate. The trained model is saved to the specified output directory (model_dir) using model.save.

```
import os
from micmon.audio import AudioDevice
from micmon.model import Model

# Import libraries
import RPi.GPIO as GPIO
import time

# Set GPIO numbering mode
GPIO.setmode(GPIO.BOARD)
```

```
# Set pin 11 as an output, and set servo1 as pin 11 as PWM
GPIO.setup(13,GPIO.OUT)
servo1 = GPIO.PWM(13,50) # Note 11 is pin, 50 = 50Hz pulse


model_dir = os.path.expanduser('models/sound-detect')
model = Model.load(model_dir)
audio_system = 'alsa'        # Supported: alsa and pulse
audio_device = 'plughw:1,0'  # Get list of recognized input devices with arecord
-l

with AudioDevice(audio_system, device=audio_device) as source:
    for sample in source:
        # Pause recording while we process the frame
        source.pause()
        prediction = model.predict(sample)
        print(prediction)
        if prediction == "positive":
            source.resume()
            # start PWM running, but with value of 0 (pulse off)
            servo1.start(0)
            print("Waiting for 2 seconds")
            time.sleep(2)

            # Let's move the servo!
            print("Rotating 180 degrees in 10 steps")

            # Define variable duty
            duty = 2

            # Loop for duty values from 2 to 12 (0 to 180 degrees)
            while duty <= 12:
                servo1.ChangeDutyCycle(duty)
                time.sleep(1)
                duty = duty + 1

            # Wait a couple of seconds
            time.sleep(2)
        else:
            print("Crying not detected.........")
            source.resume()
```

Micmon library used for sound detection, RPi.GPIO for GPIO control, and ALSA for audio input. After that, It configures pin 13 as an output and initializes a PWM (Pulse Width Modulation) object (servo1) on pin 13 with a frequency of 50 Hz. Then, loads the model using Model.load method. Then Sets the audio system to ALSA (audio_system = 'alsa') and specifies the audio device (audio_device = 'plughw:1,0'). The audio device value ('plughw:1,0') which need to change based on port connected. source.pause() pause the recording and start the predicting. And then if positive detected, it will start servo motor for full swing (0 to 180 degree).

Code for webcam-based video streaming. Mjpg-streamer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <signal.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <getopt.h>
#include <pthread.h>
#include <dlfcn.h>
#include <fcntl.h>
#include <syslog.h>
#include <linux/types.h>          /* for videodev2.h */
#include <linux/videodev2.h>

#include "utils.h"
#include "mjpg_streamer.h"

/* globals */
static globals global;

/******************************************************************************
Description.: Display a help message
Input Value.: argv[0] is the program name and the parameter progname
Return Value: -
```

```
******************************************************************************/
static void help(char *progname)
{
    fprintf(stderr, "---------------------------------------------------------------
----------\n");
    fprintf(stderr, "Usage: %s\n" \
            "  -i | --input \"<input-plugin.so> [parameters]\"\n" \
            "  -o | --output \"<output-plugin.so> [parameters]\"\n" \
            " [-h | --help ]........: display this help\n" \
            " [-v | --version ].....: display version information\n" \
            " [-b | --background]...: fork to the background, daemon mode\n",
progname);
    fprintf(stderr, "---------------------------------------------------------------
----------\n");
    fprintf(stderr, "Example #1:\n" \
            " To open an UVC webcam \"/dev/video1\" and stream it via HTTP:\n" \
            "  %s -i \"input_uvc.so -d /dev/video1\" -o \"output_http.so\"\n",
progname);
    fprintf(stderr, "---------------------------------------------------------------
----------\n");
    fprintf(stderr, "Example #2:\n" \
            " To open an UVC webcam and stream via HTTP port 8090:\n" \
            "  %s -i \"input_uvc.so\" -o \"output_http.so -p 8090\"\n",
progname);
    fprintf(stderr, "---------------------------------------------------------------
----------\n");
    fprintf(stderr, "Example #3:\n" \
            " To get help for a certain input plugin:\n" \
            "  %s -i \"input_uvc.so --help\"\n", progname);
    fprintf(stderr, "---------------------------------------------------------------
----------\n");
    fprintf(stderr, "In case the modules (=plugins) can not be found:\n" \
            " * Set the default search path for the modules with:\n" \
            "   export LD_LIBRARY_PATH=/path/to/plugins,\n" \
            " * or put the plugins into the \"/lib/\" or \"/usr/lib\" folder,\n"
\
            " * or instead of just providing the plugin file name, use a
complete\n" \
            "   path and filename:\n" \
            "   %s -i \"/path/to/modules/input_uvc.so\"\n", progname);
    fprintf(stderr, "---------------------------------------------------------------
----------\n");
}

/******************************************************************************
```

```
Description.: pressing CTRL+C sends signals to this process instead of just
              killing it plugins can tidily shutdown and free allocated
              resources. The function prototype is defined by the system,
              because it is a callback function.
Input Value.: sig tells us which signal was received
Return Value: -
******************************************************************************/
static void signal_handler(int sig)
{
    int i;

    /* signal "stop" to threads */
    LOG("setting signal to stop\n");
    global.stop = 1;
    usleep(1000 * 1000);

    /* clean up threads */
    LOG("force cancellation of threads and cleanup resources\n");
    for(i = 0; i < global.incnt; i++) {
        global.in[i].stop(i);
        /*for (j = 0; j<MAX_PLUGIN_ARGUMENTS; j++) {
            if (global.in[i].param.argv[j] != NULL) {
                free(global.in[i].param.argv[j]);
            }
        }*/
    }

    for(i = 0; i < global.outcnt; i++) {
        global.out[i].stop(global.out[i].param.id);
        pthread_cond_destroy(&global.in[i].db_update);
        pthread_mutex_destroy(&global.in[i].db);
        /*for (j = 0; j<MAX_PLUGIN_ARGUMENTS; j++) {
            if (global.out[i].param.argv[j] != NULL)
                free(global.out[i].param.argv[j]);
        }*/
    }
    usleep(1000 * 1000);

    /* close handles of input plugins */
    for(i = 0; i < global.incnt; i++) {
        dlclose(global.in[i].handle);
    }

    for(i = 0; i < global.outcnt; i++) {
        int j, skip = 0;
```

```
        DBG("about to decrement usage counter for handle of %s, id #%02d, handle:
%p\n", \
            global.out[i].plugin, global.out[i].param.id, global.out[i].handle);

        for(j=i+1; j<global.outcnt; j++) {
          if ( global.out[i].handle == global.out[j].handle ) {
            DBG("handles are pointing to the same destination (%p == %p)\n",
global.out[i].handle, global.out[j].handle);
            skip = 1;
          }
        }
        if ( skip ) {
          continue;
        }

        DBG("closing handle %p\n", global.out[i].handle);

        dlclose(global.out[i].handle);
    }
    DBG("all plugin handles closed\n");

    LOG("done\n");

    closelog();
    exit(0);
    return;
}

static int split_parameters(char *parameter_string, int *argc, char **argv)
{
    int count = 1;
    argv[0] = NULL; // the plugin may set it to 'INPUT_PLUGIN_NAME'
    if(parameter_string != NULL && strlen(parameter_string) != 0) {
        char *arg = NULL, *saveptr = NULL, *token = NULL;

        arg = strdup(parameter_string);

        if(strchr(arg, ' ') != NULL) {
            token = strtok_r(arg, " ", &saveptr);
            if(token != NULL) {
                argv[count] = strdup(token);
                count++;
                while((token = strtok_r(NULL, " ", &saveptr)) != NULL) {
                    argv[count] = strdup(token);
                    count++;
```

```
                    if(count >= MAX_PLUGIN_ARGUMENTS) {
                        IPRINT("ERROR: too many arguments to input plugin\n");
                        return 0;
                    }
                }
            }
        }
        free(arg);
    }
    *argc = count;
    return 1;
}

/*****************************************************************************
Description.:
Input Value.:
Return Value:
*****************************************************************************/
int main(int argc, char *argv[])
{
    //char *input  = "input_uvc.so --resolution 640x480 --fps 5 --device
/dev/video0";
    char *input[MAX_INPUT_PLUGINS];
    char *output[MAX_OUTPUT_PLUGINS];
    int daemon = 0, i, j;
    size_t tmp = 0;

    output[0] = "output_http.so --port 8080";
    global.outcnt = 0;
    global.incnt = 0;

    /* parameter parsing */
    while(1) {
        int c = 0;
        static struct option long_options[] = {
            {"help", no_argument, NULL, 'h'},
            {"input", required_argument, NULL, 'i'},
            {"output", required_argument, NULL, 'o'},
            {"version", no_argument, NULL, 'v'},
            {"background", no_argument, NULL, 'b'},
            {NULL, 0, NULL, 0}
        };

        c = getopt_long(argc, argv, "hi:o:vb", long_options, NULL);
```

```
        /* no more options to parse */
        if(c == -1) break;

        switch(c) {
        case 'i':
            input[global.incnt++] = strdup(optarg);
            break;

        case 'o':
            output[global.outcnt++] = strdup(optarg);
            break;

        case 'v':
            printf("MJPG Streamer Version: %s\n",
#ifdef GIT_HASH
            GIT_HASH
#else
            SOURCE_VERSION
#endif
            );
            return 0;
            break;

        case 'b':
            daemon = 1;
            break;

        case 'h': /* fall through */
        default:
            help(argv[0]);
            exit(EXIT_FAILURE);
        }
    }

    openlog("MJPG-streamer ", LOG_PID | LOG_CONS, LOG_USER);
    //openlog("MJPG-streamer ", LOG_PID|LOG_CONS|LOG_PERROR, LOG_USER);
    syslog(LOG_INFO, "starting application");

    /* fork to the background */
    if(daemon) {
        LOG("enabling daemon mode");
        daemon_mode();
    }

    /* ignore SIGPIPE (send by OS if transmitting to closed TCP sockets) */
```

```
    signal(SIGPIPE, SIG_IGN);

    /* register signal handler for <CTRL>+C in order to clean up */
    if(signal(SIGINT, signal_handler) == SIG_ERR) {
        LOG("could not register signal handler\n");
        closelog();
        exit(EXIT_FAILURE);
    }

    /*
     * messages like the following will only be visible on your terminal
     * if not running in daemon mode
     */
#ifdef GIT_HASH
    LOG("MJPG Streamer Version: git rev: %s\n", GIT_HASH);
#else
    LOG("MJPG Streamer Version.: %s\n", SOURCE_VERSION);
#endif

    /* check if at least one output plugin was selected */
    if(global.outcnt == 0) {
        /* no? Then use the default plugin instead */
        global.outcnt = 1;
    }

    /* open input plugin */
    for(i = 0; i < global.incnt; i++) {
        /* this mutex and the conditional variable are used to synchronize access
to the global picture buffer */
        if(pthread_mutex_init(&global.in[i].db, NULL) != 0) {
            LOG("could not initialize mutex variable\n");
            closelog();
            exit(EXIT_FAILURE);
        }
        if(pthread_cond_init(&global.in[i].db_update, NULL) != 0) {
            LOG("could not initialize condition variable\n");
            closelog();
            exit(EXIT_FAILURE);
        }

        tmp = (size_t)(strchr(input[i], ' ') - input[i]);
        global.in[i].stop      = 0;
        global.in[i].context   = NULL;
        global.in[i].buf       = NULL;
        global.in[i].size      = 0;
```

```
        global.in[i].plugin = (tmp > 0) ? strndup(input[i], tmp) :
strdup(input[i]);
        global.in[i].handle = dlopen(global.in[i].plugin, RTLD_LAZY);
        if(!global.in[i].handle) {
            LOG("ERROR: could not find input plugin\n");
            LOG("       Perhaps you want to adjust the search path with:\n");
            LOG("       # export LD_LIBRARY_PATH=/path/to/plugin/folder\n");
            LOG("       dlopen: %s\n", dlerror());
            closelog();
            exit(EXIT_FAILURE);
        }
        global.in[i].init = dlsym(global.in[i].handle, "input_init");
        if(global.in[i].init == NULL) {
            LOG("%s\n", dlerror());
            exit(EXIT_FAILURE);
        }
        global.in[i].stop = dlsym(global.in[i].handle, "input_stop");
        if(global.in[i].stop == NULL) {
            LOG("%s\n", dlerror());
            exit(EXIT_FAILURE);
        }
        global.in[i].run = dlsym(global.in[i].handle, "input_run");
        if(global.in[i].run == NULL) {
            LOG("%s\n", dlerror());
            exit(EXIT_FAILURE);
        }
        /* try to find optional command */
        global.in[i].cmd = dlsym(global.in[i].handle, "input_cmd");

        global.in[i].param.parameters = strchr(input[i], ' ');

        for (j = 0; j<MAX_PLUGIN_ARGUMENTS; j++) {
            global.in[i].param.argv[j] = NULL;
        }

        split_parameters(global.in[i].param.parameters, &global.in[i].param.argc,
global.in[i].param.argv);
        global.in[i].param.global = &global;
        global.in[i].param.id = i;

        if(global.in[i].init(&global.in[i].param, i)) {
            LOG("input_init() return value signals to exit\n");
            closelog();
            exit(0);
        }
```

```
    }

    /* open output plugin */
    for(i = 0; i < global.outcnt; i++) {
        tmp = (size_t)(strchr(output[i], ' ') - output[i]);
        global.out[i].plugin = (tmp > 0) ? strndup(output[i], tmp) :
strdup(output[i]);
        global.out[i].handle = dlopen(global.out[i].plugin, RTLD_LAZY);
        if(!global.out[i].handle) {
            LOG("ERROR: could not find output plugin %s\n",
global.out[i].plugin);
            LOG("       Perhaps you want to adjust the search path with:\n");
            LOG("       # export LD_LIBRARY_PATH=/path/to/plugin/folder\n");
            LOG("       dlopen: %s\n", dlerror());
            closelog();
            exit(EXIT_FAILURE);
        }
        global.out[i].init = dlsym(global.out[i].handle, "output_init");
        if(global.out[i].init == NULL) {
            LOG("%s\n", dlerror());
            exit(EXIT_FAILURE);
        }
        global.out[i].stop = dlsym(global.out[i].handle, "output_stop");
        if(global.out[i].stop == NULL) {
            LOG("%s\n", dlerror());
            exit(EXIT_FAILURE);
        }
        global.out[i].run = dlsym(global.out[i].handle, "output_run");
        if(global.out[i].run == NULL) {
            LOG("%s\n", dlerror());
            exit(EXIT_FAILURE);
        }

        /* try to find optional command */
        global.out[i].cmd = dlsym(global.out[i].handle, "output_cmd");

        global.out[i].param.parameters = strchr(output[i], ' ');

        for (j = 0; j<MAX_PLUGIN_ARGUMENTS; j++) {
            global.out[i].param.argv[j] = NULL;
        }
        split_parameters(global.out[i].param.parameters,
&global.out[i].param.argc, global.out[i].param.argv);

        global.out[i].param.global = &global;
```

```
        global.out[i].param.id = i;
        if(global.out[i].init(&global.out[i].param, i)) {
            LOG("output_init() return value signals to exit\n");
            closelog();
            exit(EXIT_FAILURE);
        }
    }

    /* start to read the input, push pictures into global buffer */
    DBG("starting %d input plugin\n", global.incnt);
    for(i = 0; i < global.incnt; i++) {
        syslog(LOG_INFO, "starting input plugin %s", global.in[i].plugin);
        if(global.in[i].run(i)) {
            LOG("can not run input plugin %d: %s\n", i, global.in[i].plugin);
            closelog();
            return 1;
        }
    }

    DBG("starting %d output plugin(s)\n", global.outcnt);
    for(i = 0; i < global.outcnt; i++) {
        syslog(LOG_INFO, "starting output plugin: %s (ID: %02d)",
global.out[i].plugin, global.out[i].param.id);
        global.out[i].run(global.out[i].param.id);
    }

    /* wait for signals */
    pause();

    return 0;
}
```

Explanation: The mjpg-streamer is the API used to start video feed from the webcam using WebSocket protocol. The input plugin sets the device which is being used as the camera for the recording, the device driver on the raspberry pi is checked to ensure that we are using correct dev/device number in the code. The output plugin is responsible for run the live feed on the localhost server on a particular port address 8080. Eventually, the feed is hosted web using web-hosting servers.

Code for DHT11 Sensor: DHT11.py

```
import Adafruit_DHT
```

```
import time
import csv
import BlynkLib

sensor = Adafruit_DHT.DHT11
pin = 4  # Use the GPIO pin where you connected the DATA pin
log_file = "sensor_data.csv"  # Name of the log file

BLYNK_AUTH = 'G_sEt8nDdEew6nbsZphllcJ3mxi56osy'  # Replace with your Blynk
Authentication Token

blynk = BlynkLib.Blynk(BLYNK_AUTH)

while True:
    humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)

    if humidity is not None and temperature is not None:
        current_time = time.strftime('%Y-%m-%d %H:%M:%S')
        with open(log_file, 'a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow([current_time, temperature, humidity])
        print(f'Recorded: {current_time}, Temperature: {temperature:.1f}°C,
Humidity: {humidity:.1f}%')
        blynk.virtual_write(1, temperature)  # Send temperature data to V1
        blynk.virtual_write(0, humidity)     # Send humidity data to V0
        if humidity > 45:
            # Send notification when temperature exceeds the threshold
                blynk.log_event('humi_notify')
        elif temperature < 20 or temperature > 30:
                blynk.log_event('temp_notify')
    time.sleep(5)  # Log data every 5 seconds
    blynk.run()
```

Explanation:
To implement the DHT11 sensor temperature and humidity values, the library Adafruit_DHT is used. Adafruit_DHT.read_retry(DHT11, pin_number) function is responsible for reading the temperature and humidity values from the sensor. The values are then transmitted to blynk server by using BlynkLib library. Blynk.virtual_write() is the function to write the data on the virtual pins of the gauge used in the blynk server. The notification are triggered when the values goes beyond the threshold value using the blynk.log_event() function.

Code for Proteus Simulation

```
#!/usr/bin/python
import spidev
import time
import os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

# Open SPI bus
spi = spidev.SpiDev()
spi.open(0,0)

# Define GPIO to LCD mapping
LCD_RS = 15
LCD_E  = 16
LCD_D4 = 7
LCD_D5 = 11
LCD_D6 = 12
LCD_D7 = 13

# Define sensor channels
temp_channel  = 0
'''
define pin for lcd
'''
# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005
delay = 1



GPIO.setup(LCD_E, GPIO.OUT)  # E
GPIO.setup(LCD_RS, GPIO.OUT) # RS
GPIO.setup(LCD_D4, GPIO.OUT) # DB4
GPIO.setup(LCD_D5, GPIO.OUT) # DB5
GPIO.setup(LCD_D6, GPIO.OUT) # DB6
GPIO.setup(LCD_D7, GPIO.OUT) # DB7

# Define some device constants
LCD_WIDTH = 16    # Maximum characters per line
LCD_CHR = True
```

```
LCD_CMD = False
LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line

'''
Function Name :lcd_init()
Function Description : this function is used to initialized lcd by sending the different commands
'''
def lcd_init():
  # Initialise display
  lcd_byte(0x33,LCD_CMD) # 110011 Initialise
  lcd_byte(0x32,LCD_CMD) # 110010 Initialise
  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  time.sleep(E_DELAY)
'''
Function Name :lcd_byte(bits ,mode)
Fuction Name :the main purpose of this function to convert the byte data into bit and send to lcd port
'''
def lcd_byte(bits, mode):
  # Send byte to data pins
  # bits = data
  # mode = True  for character
  #        False for command

  GPIO.output(LCD_RS, mode) # RS

  # High bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x10==0x10:
    GPIO.output(LCD_D4, True)
  if bits&0x20==0x20:
    GPIO.output(LCD_D5, True)
  if bits&0x40==0x40:
    GPIO.output(LCD_D6, True)
  if bits&0x80==0x80:
    GPIO.output(LCD_D7, True)

  # Toggle 'Enable' pin
```

```
  lcd_toggle_enable()

  # Low bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x01==0x01:
    GPIO.output(LCD_D4, True)
  if bits&0x02==0x02:
    GPIO.output(LCD_D5, True)
  if bits&0x04==0x04:
    GPIO.output(LCD_D6, True)
  if bits&0x08==0x08:
    GPIO.output(LCD_D7, True)

  # Toggle 'Enable' pin
  lcd_toggle_enable()
'''
Function Name : lcd_toggle_enable()
Function Description:basically this is used to toggle Enable pin
'''
def lcd_toggle_enable():
  # Toggle enable
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)
'''
Function Name :lcd_string(message,line)
Function  Description :print the data on lcd
'''
def lcd_string(message,line):
  # Send string to display

  message = message.ljust(LCD_WIDTH," ")

  lcd_byte(line, LCD_CMD)

  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)
```

```
# Function to read SPI data from MCP3008 chip
# Channel must be an integer 0-7
def ReadChannel(channel):
  adc = spi.xfer2([1,(8+channel)<<4,0])
  data = ((adc[1]&3) << 8) + adc[2]
  return data


# Function to calculate temperature from
# TMP36 data, rounded to specified
# number of decimal places.
def ConvertTemp(data,places):

  # ADC Value
  # (approx)  Temp  Volts
  #    0      -50    0.00
  #   78      -25    0.25
  #  155        0    0.50
  #  233       25    0.75
  #  310       50    1.00
  #  465      100    1.50
  #  775      200    2.50
  # 1023      280    3.30

  temp = ((data * 330)/float(1023))
  temp = round(temp,places)
  return temp



# Define delay between readings
delay = 5
lcd_init()
lcd_string("welcome ",LCD_LINE_1)
time.sleep(2)
while 1:
  temp_level = ReadChannel(temp_channel)
  temp       = ConvertTemp(temp_level,2)

  # Print out results
  lcd_string("Temperature  ",LCD_LINE_1)
  lcd_string(str(temp),LCD_LINE_2)
  time.sleep(1)
```

The above code is a Python script for a Raspberry Pi that reads temperature data from an MCP3008 analog-to-digital converter (ADC) using the SPI interface and displays the temperature on an LCD.

This is a small snippet of the code of the simulation. The script is programmed to turn a Raspberry Pi into a communication bridge linking a temperature sensor and an LCD screen. Initially, it prepares the Raspberry Pi's GPIO pins to handle the LCD screen’s operations. The script then sets up a pathway for data to flow from the MCP3008—a device that converts the sensor's analog signals into digital form—through the SPI bus. Once the system boots up, a welcome note pops up on the LCD. From there on, the script falls into a perpetual cycle: it reads the temperature sensor's output, converts those readings into understandable temperature figures, and updates the LCD screen with the latest temperature. This process repeats every second, providing a dynamic and continuous temperature display.