SmartGuard: Advanced Motion Detection and Streaming with Raspberry Pi and AWS

Ammar Isa
Department of Electrical and Electronics Engineering
University of Bahrain

Mohab Mangoud
Department of Electrical and Electronics Engineering
University of Bahrain

***Abstract*— This project presents an innovative approach in the realm of security systems, using the capabilities of the Internet of Things (IoT) to enhance surveillance and safety measures. This report details the development and implementation of a sophisticated motion detection and streaming system using a Raspberry Pi 4 and web cameras, integrated seamlessly with Amazon Web Services (AWS) for cloud-based data handling and storage. The project aimed to create an advanced IoT-based security system capable of real-time motion detection, video streaming, and remote system management. By using technologies such as MQTT for device communication and incorporating a user-friendly HTML-based dashboard, the project provides efficient, real-time surveillance capabilities. Our findings indicate that the integration of IoT devices with cloud computing platforms significantly enhances the effectiveness and responsiveness of security systems. The project successfully overcame challenges related to hardware integration, streaming latency, and motion detection accuracy, demonstrating the feasibility and reliability of IoT in security applications. Through this project, we showcase the transformative power of IoT in revolutionizing traditional security systems, offering valuable insights and a strong foundation for future innovations in IoT-based security solutions.**

Keywords—IoT, Motion Detection, MQTT, AWS, PuTTY, RealVNC, Video Streaming, Video, Raspberry Pi, Security Systems, Security, HLS

I. Introduction

The objective of this project is to build a security camera with motion detection system using Raspberry Pi. In additional this project solves the needs of physical connection during the maintenances for the raspberry pi using VNC Cloud to control the raspberry pi. The use of advanced technology has become essential in the context of modern security systems in order to meet current security issues. Using the flexibility and power of the Raspberry Pi, this project seeks to innovate in this field by creating a smart security camera system with motion detection. The major goals are to improve the system's maintenance and accessibility features in addition to developing a dependable security apparatus. One of the notable challenges in conventional security systems is the dependency on physical connections for maintenance and monitoring. Addressing this, the project introduces the use of VNC Cloud, facilitating remote control and maintenance of the Raspberry Pi, thereby significantly reducing the need for physical interaction. This advancement not only streamlines the maintenance process but also adds a layer of flexibility in system management. The primary objective of this project is to create an HTML-based dashboard that is easy to use. Serving as the hub of the whole operation, this website offers users a thorough and user-friendly interface. Several features are integrated into the dashboard, such as a live feed from the security camera that uses High-Level broadcast (HLS) technology to broadcast smoothly. This feature is accomplished by establishing a direct connection with the AWS infrastructure and using Kinesis Video Streams to provide a steady and excellent video stream. Furthermore, the project harnesses the power of MQTT protocol, with Raspberry Pi serving as the publisher. This setup enables an efficient history of motion detection records, connecting the Raspberry Pi with the AWS platform, which acts as the broker. The website, being the subscriber in this arrangement, presents these records in an organized and accessible manner, allowing users to review and analyze security data effectively. This project seeks to transform not only the way security camera systems are maintained and used, but also the way they are designed with sophisticated motion detection. The project promises increased efficiency, flexibility, and user-friendliness in the realm of security systems by integrating technologies like Raspberry Pi, VNC Cloud, HTTP, AWS, and MQTT.

II. Paper Structure

The report commences with a comprehensive literature review (Section III), delving into prior research and developments in IoT security systems, motion detection technologies, and the role of cloud computing within the IoT paradigm. This is followed by an in-depth exploration of system design and architecture (Section IV), discussing the required hardware, software, and protocols used in the project. The implementation section (Section V) delves deeply into the technical aspects, detailing the setup of the Raspberry Pi OS, along with the configuration of PuTTY and RealVNC software, and their integration into the project. This section also covers motion detection and the detailed use of AWS Services. Next, the report presents the results and discussion (Section VI), detailing the project's successes and addressing challenges encountered during its implementation. This section assesses the outcomes, focusing on how various challenges were effectively resolved. Finally, the conclusion (Section VII) summarizes the report and discusses potential avenues for future development, reflecting on the project's implications and future prospects in the field of IoT security.

III. Literature Review

Investigating different paper is crucial part specially in fields like the developments in IoT security systems, motion detection technologies, and the role of cloud computing within the IoT paradigm. One of the papers that investigates facial recognition-based door access control system as part of a home security solution and the system is built using a Raspberry Pi, integrating IoT for remote control and communication. the core feature for identifying individuals for door access. The paper discusses various methods and technologies for effective facial recognition. IoT is used to enhance the system’s functionality, allowing remote monitoring and control, which is facilitated through a

smartphone application. the study shows Raspberry Pi for processing, control and optimizing Raspberry Pi performance for image processing which is valuable for motion detection functionality. In addition, techniques such as CNNs used in the paper for facial recognition, could potentially be adapted for more sophisticated motion detection or for adding additional features like object or person recognition to this project [1]. Also, a multifaceted home security system employing IoT technologies and a diverse array of sensors, including gas, humidity, temperature, and motion detectors, is thoroughly explored. Significantly, the paper's deployment of microcontrollers like Arduino and Raspberry Pi for sensor data processing parallels the technological approach of your project, which utilizes Raspberry Pi for motion detection and video streaming. The study's emphasis on real-time data processing and IoT-based monitoring and control systems, including the development of a user-centric Android application, aligns closely with your project's use of a web-based dashboard for interactive control and monitoring. Furthermore, the paper's discussion of challenges and future IoT trends provides valuable insights for enhancing system reliability and data accuracy, offering a comprehensive perspective that can inform and potentially guide the development and refinement of this IoT-based motion detection system project [2], [3]. Moreover, a paper that provides a comprehensive analysis of a fall detection system, utilizing RGB cameras and IoT technologies in intricate home settings, focusing particularly on aiding the elderly. This study is notably aligned with your project in its application of camera-based monitoring, combining hardware like Raspberry Pi with sophisticated software for real-time data analysis. The paper's methodology, involving the integration of IoT devices with advanced algorithms for accurate and responsive monitoring, parallels to this project use of web cameras for motion detection. It underscores the significance of real-time processing and the precision of algorithms in IoT systems. Where, the challenges addressed such as accuracy in diverse environments, and the exploration of future improvements like enhanced image processing and AI integration that offer valuable insights. These insights are particularly pertinent for optimizing is project motion detection capabilities and guiding its future development in IoT-based security systems [4]. In addition, a paper introduces a novel approach to detect small, fast-moving insects using advanced object detection methods, incorporating motion-informed enhancement (MIE) and deep learning models like YOLO and Faster R-CNN. This methodology, focusing on real-time analysis and image preprocessing in complex environments, aligns closely with the objectives of this IoT project that involves real-time motion detection and video streaming. The integration of Raspberry Pi with sophisticated image processing algorithms as explored in the paper, parallels this project utilization of similar technology for video data analysis. Crucially, the paper's strategies for enhancing detection accuracy in variable conditions and its insights into deep learning applications provide valuable guidance for this project. These methodologies could significantly refine the motion detection capabilities of this IoT-based security system, particularly in complex and dynamic environmental settings, paving the way for future advancements in IoT and camera-based monitoring systems [5]. A notable omission in the papers the lack of discussion on the maintenance and updating of IoT systems, particularly in hard-to-reach areas. These studies, while comprehensive in their technical approach, do not delve into the practical aspects of sustaining and upgrading these systems post-deployment, which is critical for long-term functionality and reliability. In this IoT project it has been implemented a strategy for remote maintenance and updates. This approach is crucial for systems deployed in areas where physical access is challenging or limited. With leveraging cloud services, SSH protocols, and other remote access tools where system allows for the updating of software, troubleshooting, and performance tuning from a distance. This capability significantly reduces the need for physical intervention. Moreover, by facilitating remote maintenance that ensures higher system uptime and reliability as issues can be addressed promptly without the need for on-site visits. This project creates a system with an ability to receive updates remotely means it can adapt to evolving environmental conditions, security threats, or technological advancements, maintaining its effectiveness over time. In addition, this project not only advances the technological aspects discussed in the papers but also contributes to the practical application of IoT systems. It provides a blueprint for how maintenance and updates can be efficiently managed in IoT deployments particularly in scenarios where accessibility is a concern. The inclusion of remote maintenance and update capabilities in this project addresses a critical gap in the existing literature, highlighting the importance of not just developing IoT solutions but also ensuring their long-term sustainability. This aspect of the project not only enhances its practicality but also serves as a valuable reference point for future IoT developments, especially in the context of security systems in challenging environments.

IV. System Design And Architecture

This project harnessed the compact power of the Raspberry Pi 4 as the core computing device, complemented by web camera for real-time visual monitoring. The software stack was built primarily using Python with OpenCV library for its proven capabilities in image processing and motion detection algorithm. Where, integration with AWS platform played a pivotal role in this project, enabling sophisticated data handling, stream capabilities, and remote system management. This section contains a detailed description of the system architecture, including the hardware part like Raspberry Pi 4 with the web camera, and the required software like AWS platform, PuTTY, and RealVNC. It provides an overview about the use of Hypertext Transfer Protocol (HTTP). Additional to the use of the HTTP Live Streaming (HLS) protocol for efficient video streaming and Massage Queuing Telemetry Transport (MQTT) protocol for reliable message queuing and delivery is elaborated upon highlighting their significance in the project. Moreover, the implementation of secure communication protocols such as Secure Socket Shell (SSH), and Remote Framebuffer (RFB) are discussed to ensure the system's integrity and security. That will give the read clear overview of the requirements to create the project from the beginning.

A. Hardware

The main hardware requirement in this project are computer, Raspberry Pi 4, and web camera to achieve a successful result. First, computer is essential to control the Raspberry Pi 4 and to impellent the necessary codes to apply the motion detection, etc. Also, computer will be used to fill a gap in the previous literate reviews. The Raspberry Pi 4 is a significant upgrade in Raspberry Pi series that offering enhanced performance and flexibility. As a compact, cost-

effective which is powerful single-board computer. Figure.1 show the structure of Raspberry Pi 4.

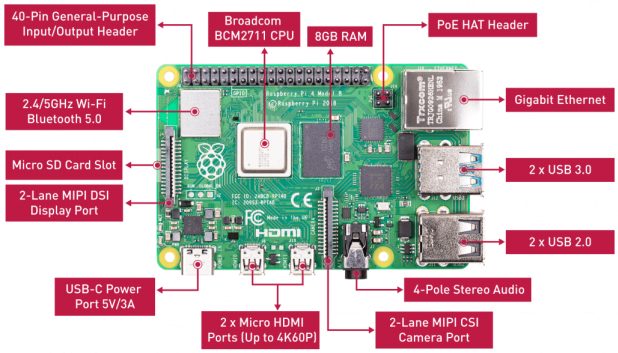

Fig. 1. Raspberry Pi 4 Structure

The Raspberry Pi 4 is powerful due to its comprehensive specification, which is equipped with a Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64bit SoC, operating at 1.5GHz that provides substantial computational power. It comes in different RAM configurations – 2GB, 4GB, or 8GB, allowing for versatile application based on project requirements. Also, it includes a Gigabit Ethernet port, dual-band 2.4/5.0 GHz wireless Lan, Bluetooth 5.0, and BLE for various connectivity options. The 40-Pin GPIO header is fully backward compatible with the previous boards, offering wide ranging interfacing capabilities. In addition, it features two USB 3.0 ports and two USB 2.0 ports, enabling the connection of multiple peripherals. The board supports two micro-HDMI ports, supporting up to 4K resolution, and 4-pole stereo audio and composite video port. For the storage, it uses a microSD card slot for loading the operating system and data storage, which is flexible and convenient [6] . Picking the Raspberry Pi 4 as the main embedded system for different reasons such as, the computational capability where the quad-core processor and the option for up to 8GB RAM make it an ideal choice for handling the computational demands of motion detection algorithms and video processing tasks. Also, its wide range of connectivity features including wireless LAN and Bluetooth are essential for projects that require remote communication and data transmission. Moreover, the support for high-definition video and compatibility with camera modules make it particularly suited for a project that revolves around video capture and streaming. In addition, its small form device and it require low energy consumption which is crucial for surveillance system intended to run continuously. Considering its performance capabilities, the raspberry pi 4 offers an economical solution keeping the project cost effective without compromising functionality.

Webcam is essential part of this project where the primary role of it is to continuously monitor the environment to detect motion. Its capability ensures that even subtle movements are captured, which is crucial for the effectiveness of motion detection algorithm. Also, it capturing the real-time video stream. Moreover, once the motion detected, the camera records the video feed, and store it. The webcam also serves as a data acquisition tool. The images and videos captured by the camera are essential inputs for the analysis algorithms running on the Raspberry Pi. In the broader scope of the project, the camera acts as a surveillance device, contributing to the security aspect by providing visual monitoring of the targeted area. Figure.2 shows an example of Webcam.

Fig. 2. Example of Webcam

The integration of the Raspberry Pi 4 and a high-definition web camera forms an effective core for this IoT-based motion detection project. The Raspberry Pi 4, with its robust processing power, versatile memory options, and extensive connectivity, is ideally suited for handling complex motion detection algorithms and video processing tasks. Its compact size and energy efficiency make it particularly fitting for continuous surveillance systems, offering a cost-effective yet powerful solution. The web camera, pivotal in continuously monitoring and recording motion, enhances the system's effectiveness with its high-quality video capture and crucial role in data acquisition for analysis. Together, these components create a sophisticated, reliable, and economical security system, demonstrating the potential of combining cutting-edge technology in IoT applications.

B. Software & protocols

Different types of software used to achieve a successful result. Python is chosen as primary programming language for this IoT motion detection project duo to ease of use where its renowned for its readability and simplicity that making it accessible for achieve the desired results in this project. Also, Python boasts a vast and active community that provides an abundance of recourses and support, which is invaluable for troubleshooting and enhancing project capability. Moreover, its seamlessly integrates with various hardware and software platforms, including Raspberry Pi and web cameras which allowing it for smooth development process. Python is very efficient when it comes for image processing and computer vision while it provides powerful libraries such as what used in this project OpenCV that make an optimal choice for motion detection. OpenCV offers efficient algorithm detecting motion. In this project, frame differencing techniques, where consecutive video frames are compared to identify differences caused by motion were primarily used. Also, it provides tools for image manipulation and processing, where are crucial for preparing frames for analysis and enhancing the accuracy of motion detection. OpenCV's compatibility with Python and its ability to run on various platforms, including the Raspberry Pi, makes it an ideal choice for this project.

RealVNC (Virtual Network Computing), it is a remote access software that allows users to control and interact with computers remotely over network. It utilizes the Remote Framebuffer (RFB) protocol to transmit the keyboard and mouse input from one computer which is the client to another which is the server, and relay the graphical screen updates

back to the client [7]. This technology is particularly beneficial for managing devices like Raspberry Pi in remote or inaccessible locations. RealVNC is used to remotely access and control the Raspberry Pi's desktop environment over the internet. This is useful for managing the Raspberry Pi settings, software, and files from a remote location. The Raspberry Pi acts as a VNC server and any device with a VNC client connect to it. While, the server software runs on the Raspberry Pi, enabling it to share its screen while the client software runs on another device, allowing it to control the Raspberry Pi. In addition, RealVNC offers a cloud service that simplifies the connection process by reducing the complex network configuration requirements. Moreover, this feature is useful when it comes to this project duo to its availability to be connected easily with the Raspberry Pi that makes it accessible over the internet securely through the RealVNC Cloud services without requiring direct network connection. In addition, this connection is secured through end-to-end encryption ensuring that remote access is protected against unauthorized access. The core of RealVNC's functionality is the RFB (Remote Framebuffer) protocol which is designed to work efficiently over low bandwidth and high-latency connections that makes it suitable for various network conditions. Moreover, RFB transmits the graphical interface of the Raspberry Pi in real-time while sending control inputs from the client. This allows for responsive and interactive user experience similarly as if someone is directly working on the Raspberry Pi. This will advantage the project in various ways which enables convenient setup, management, and troubleshooting of the Raspberry Pi without the need for physical access. Also, it facilitates real-time control of the Raspberry Pi which is crucial for timely modifications in the project setup or software. In additional, it increases the system's accessibility that allowing to interact with the Raspberry Pi from any location that provided with internet.

PuTTY is an open-source application that was developed in 1997 by Simon Tatham. It is widely used for remote access to servers and computing devices like Raspberry Pi over various network protocols including SSH (Secure Shell) [8]. PuTTY uses the SSH protocol to establish a secure and encrypted connection between the client which is the user's computer and the Raspberry Pi which is the server in this context. Also, SSH is a network protocol that provides a secure channel over an unsecured network that makes it ideal for remotely executing commands and managing system. Moreover, by using PuTTY as an SSH client, users can securely log in to the Raspberry Pi from a remote location. This access will allow users to run command-line operations, modify configurations, update software, and perform other tasks directly on the Raspberry Pi shell. PuTTY is useful as different, fast, and simple way to connect to the Raspberry Pi terminal without its own monitor, keyboard, or mouse. This remote access capability is invaluable for monitoring, maintaining, and troubleshooting the Raspberry Pi especially when it is deployed in locations that are not easily accessible. PuTTY uses SSH protocol that ensures that all data transmitted between the PuTTY client and the Raspberry Pi. It is encrypted, safe against eavesdropping and malicious attacks [9]. PuTTY gives great advantages to this project like it enables efficient management of the Raspberry Pi operating system and software remotely which is critical part for the maintenance and update required in this project. Also, the use of SSH ensures that all interactions with Raspberry Pi are secure which is important specially when dealing with sensitive data like a motion detection system. Moreover, it offers immediate and flexible access to the Raspberry Pi shell that allow it for quick adjustments and real-time monitor.

AWS Services is heart of this project that connect everything together. In this IoT camera motion detection project, various Amazon Web Services (AWS) components play integral roles in data handling, storage, device management, and video streaming. The services used include Amazon S3, AWS IoT Core, IAM (Identity and Access Management), DynamoDB, and Kinesis Video Streams. Each of these services contributes uniquely to the project's functionality and efficiency. Amazon S3 (Simple Storage Service) is utilized for storing captured video footage from motion detection system. It acts as a scalable and secure object storage service. It provides high durability, availability, and performance, making it ideal for storing and retrieving any amount of data in anytime from anywhere. Next, AWS IoT Core that enables secure, bidirectional communication between IoT devices like the Raspberry Pi and the AWS server. It supports the MQTT messaging protocol which is vital for this project. It ensures a secure and efficient exchange of messages between the Raspberry Pi and the web application facilitating real time alerts and commands. Then, DynamoDB which is a NoSQL database service that used for storing structured data such as motion detection event logs, date, time and other specific data efficiently with low latency. It offers fast and predictable performance with seamless scalability, making it suitable for applications that require a database with minimal response times. Also, Amazon Kinesis Video Streams is used for securely streaming video from the Raspberry Pi camera to the website for real-time. It facilitates HLS protocol to real-time streaming and analysis of video data, enabling applications like live monitoring and motion detection events. It also integrates easily with other AWS services for enhanced data processing and storage.

HTML website dashboard hosted using the HTTP protocol. The website serves as the user interface for the IoT camera motion detection system. It provides users with centralized platform to monitor, control and interact with the motion detection system implemented on the Raspberry Pi 4. The dashboard is designed to be intuitive, responsive, and accessible from various devices, including desktops, laptops, and mobile phones. It contains displays real-time video streaming from the Raspberry Pi webcam that allowing user to monitor the environment actively. In addition, notifications and alerts are displayed when motion is detected with a history log records all motion detection events, including timestamps. Also, it provides options to enable or disable motion detection feature which allows users to adjust the settings. The front-end built using HTML for structure, CSS for styling and JavaScript for interactivity and dynamic content loading. Frameworks like Bootstrap, and Node.js used for responsive design and streamlined development.

V. IMPLEMENTAION

To apply IoT motion detection in a secure camera in a proper way, practical steps that involved configuring, setting up, and programming the Raspberry Pi 4, integrating it with a webcam, and implementing various AWS services for data handling and streaming will be explained in detail. Where the key elements include the development of Python scripts using OpenCV for motion detection, employing secure communication protocols like SSH and RFB for system security, and setting up MQTT for efficient messaging upon

motion events. This section aims to provide a clear insight into the journey from concept to reality, highlighting the critical roles of each component in the creation of this advanced IoT security solution.

A. Raspberry Pi OS Setup

Preparing the Raspberry Pi 4 OS is fundamental step in preparing the IoT camera motion detection project. The choice of OS and its configuration are critical as they lay the ground for the software and applications that will run on the device. The Raspberry Pi OS for this project will be "Raspberry Pi OS (Legacy, 64-bit) Full" because its compatibility to cameras. First step is to insert the MicroSD Card into the computer and downloading the official Raspberry Pi software "Raspberry Pi Imager" [10]. Figure.3 shows the user interface of the software, where in the Raspberry Pi Device the choices will be on "Raspberry Pi 4" which is our device. For the Operating System the choice will be on "Raspberry Pi OS (Legacy, 64-bit)" which is the required OS for our project. Next, choose the inserted MicroSD Card that will be used in the Raspberry Pi device in the Storage section. Finally, click into "Next" button in follow on with the installation until it complete and it will be ready to be inserted into the Raspberry Pi 4 device.

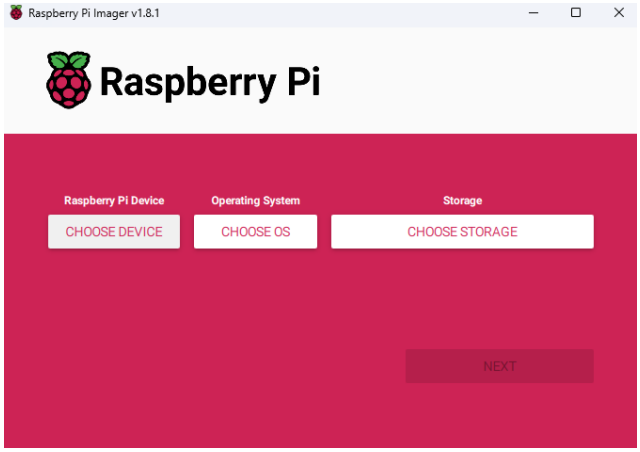

Fig. 3. Raspberry Pi Imager User Interface

B. PuTTY & RealVNC Setup

To turn on the Raspberry Pi 4 that will require to insert the prepared MicroSD into the device. Then, connect the Raspberry Pi 4 to a monitor, keyboard, and mouse. Where it will be used only for the first time to setup softwares. First, complete the step of your Raspberry Pi operation system, connect it to the network, and make sure to update the system. Using Raspberry Pi terminal, the wlan0 IPV4 can be extracted through this command "ifconfig", which is required to connect the Raspberry Pi with the PuTTY software to implement the SSH protocol and controlling the Raspberry Pi through it terminal. Now back to main computer or laptop that will be used as main control station which will gives the opportunities to control massive quantity of Raspberry Pi devices if needed. Download PuTTY through its official website [11]. Figure.4 shows the PuTTY software user interface. In the Host Name (or IP address) insert your Raspberry Pi IPV4, make sure that the Port is 22, and click to the "Open" button.

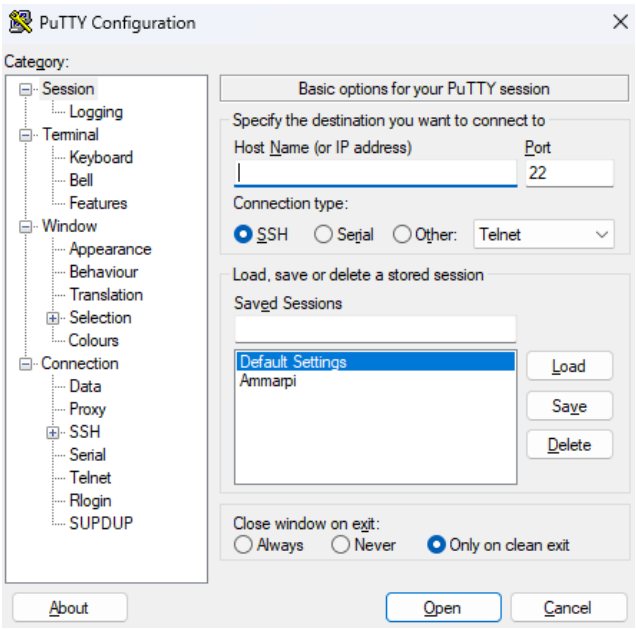

Fig. 4. PuTTY Configuration User Interface

After that, login to your Raspberry Pi through your username and password which will guarantee a full access into your Raspberry Pi terminal that gives you a freedom of controlling and installing, updating softwares etc. Then, to enter the configuration of the Raspberry Pi write this command in the terminal "sudo raspi-config" and it will take you to the configuration tool as shown in Figure.5 so you will be able to enter Interface Options to enable important settings to this project. From there make sure that Legacy Camera, SSH, and VNC is enabled so you can enhance these tools.

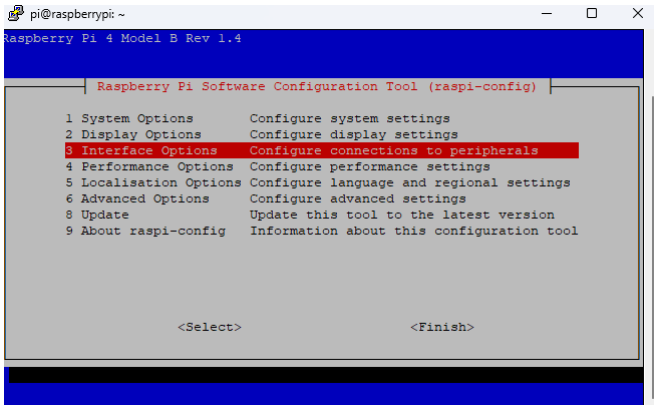

Fig. 5. Raspberry Pi Software Configuration Tool Interface

After that, providing a graphical and cloud connection could be beneficial in various ways. So RealVNC is an efficient and simple way to reach that goal. Through the Raspberry Pi terminal (can be used through PuTTY) write this command "vncserver-virtual" to create virtual VNC server as Figure.6 which will give you the opportunity to use the graphic interface of the Raspberry Pi without the requirement of it to be connected to any monitor, mouse, or keyboard. Back to your main computer, through the RealVNC viewer that can be downloaded from the official RealVNC website [12] create

an account and add your devices into your “Device Access” to achieve the Cloud control over the devices.

```
New desktop is raspberrypi:1 (192.168.100.45:1)
pi@raspberrypi:~ $ vncserver-virtual
```

Fig. 6. VNC Virtual Server Command & Required IP Address to be inserted into RealVNC Viewer

After downloading the application Figure.7 shows the RealVNC user interface, then right click into the interface and choose the “New connection...” so it gives the availability to use your Raspberry Pi address that shown in Figure.6 to get the graphical control into your Raspberry Pi 4 which will enables the RFB protocol and gives you a full access.

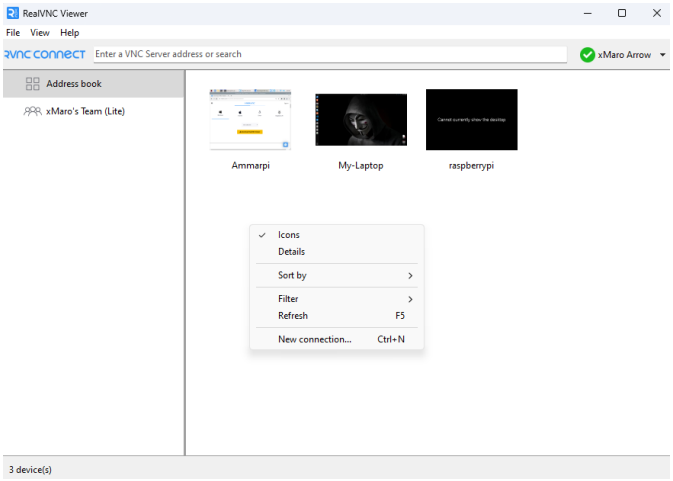

Fig. 7. RealVNC User Interface

C. *Motion Detection Setup*

Python and the OpenCV library used to develop an efficient motion detection application through the Raspberry Pi and webcam. First, in this project Python 3.9.2 version while OpenCV will be on 4.8.1.78 version. The idea of the motion detection is to separate the moving object from the background and compare the movement through the frames. To implement that using Python code along with the OpenCV library. The following explanation breaks down the key components of the Python script used for detection motion, providing insights into how each segment contributes to the overall functionality of the system. Begin with importing the “cv2” module which is OpenCV library.

```
import cv2
```

Next, initializing the camera using “VideoCaputer(0)” that will initializes video capture through the web camera while “0” denotes the default camera.

```
cap = cv2.VideoCapture(0)
```

Then, the script starts by capturing two consecutive frames (‘frame1’ and ‘frame2’) from the video feed. These frames are used to detect motion by comparing them.

```
ret, frame1 = cap.read()
ret, frame2 = cap.read()
```

After that, initializing the continues loop using the “while” function as long as the camera is operational (‘cap.isOpened()’). Inside this loop the frames are processed to detect motion. Calculates the absolute difference between two frames using “cv2.absidiff(frame1, frame2)” This difference highlights areas of movement while the image is then converted to grayscale and blurred using Gaussian blur to reduce noise and improve the accuracy of motion detection.

```
diff = cv2.absdiff(frame1, frame2)
gray = cv2.cvtColor(diff,
cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (5, 5), 0)
```

Applying thresholding that converts the image to a binary form, making it easier to identify significant movements. In addition, dilation implementation enlarges the regions of motion that facilitating the detection of contours. Then, identifies contours in the dilated image, where contours are the outlines representing areas of motion.

```
 _, thresh = cv2.threshold(blur, 20, 255,
cv2.THRESH_BINARY)
dilated = cv2.dilate(thresh, None,
iterations=3)
contours, _ = cv2.findContours(dilated,
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

The script iterates through each contour using “cv2.boundingRect(contour)’ to create a rectangular boundary around the areas of motion. Moreover, small movements are filtered out based on the contour area, focusing only on significant motion. To provide visual feedback in the video feed “cv2.rectangle()” draws rectangles around detected motion in the frame, while “cv2.imshow()” displays the current frame with highlighted motion areas.

```
for contour in contours:
  (x, y, w, h) = cv2.boundingRect(contour)

  if cv2.contourArea(contour) < 900:
     continue
  cv2.rectangle(frame1, (x, y), (x+w,
y+h), (0, 255, 0), 2)

cv2.imshow("feed", frame1)
frame1 = frame2
ret, frame2 = cap.read()
```

The frames are updated at the end of each loop iteration preparing for the next round of motion detection. This Python code effectively sets up a motion detection system using frame differencing, contour analysis, and real-time video processing as shown in Figure.8. It highlights significant motion in the video feed and provides a straightforward visual representation of detected movements. This approach forms the core of our IoT-based motion detection that enables the system to capture and react to physical movements in its environment reliably.

Fig. 8. Demonstration for the process that the motion detection code goes through to detect the motion, while its starting from the left side where that process the image to grey colors. The middle image is after applying the blur, thresh, and dilated codes. Finally, it compares the contours and draws the green rectangle on the detected motion.

D. AWS Services Setup

Amazon Web Services (AWS) plays a pivotal role in our IoT project, providing the backbone for video streaming, data storage, and device management. The integration of various AWS services was essential to achieving a seamless, efficient, and scalable IoT solution. The first step in integration AWS Kinesis Video Streams into our IoT project is to establish an account with Amazon Web Service (AWS). The first step in integrating AWS Kinesis Video Streams into our IoT project was to establish an account with Amazon Web Services (AWS). Once the account was set up, we navigated to the AWS Management Console. Here, we accessed the Kinesis Video Streams service, which is designed to securely stream video from connected devices to AWS for analytics, machine learning (ML), and other processing. carefully chose a name for the stream that aligns with our project's naming conventions and makes it easily identifiable. The service offers various configuration options. For our project, we opted to use the default configuration settings provided by AWS as shown in Figure.9. This decision was based on our requirement assessment, which indicated that the default settings were well-suited for our project's needs particularly in terms of stream retention and data encryption.

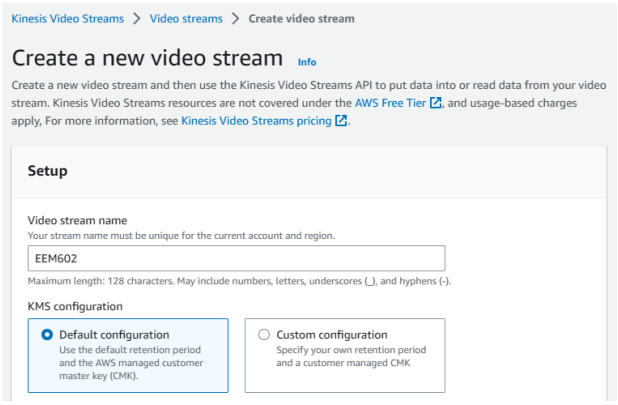

Fig. 9. Create a new video stream in Kinesis Video Streams service

After completing the stream setup, the new video stream will be activated. The next critical step in our AWS setup involved the use of AWS Identity and Access Management (IAM), a service that helps in securely controlling access to AWS resources. We started by creating a new IAM user specifically for this project where Figure.10 shows the specific location that leads to create new user in IAM dashboard. This user acts as an identity with specific permissions and is used to interact with AWS services programmatically. The creation of a dedicated user for the project ensures that we have fine-grained control over the access and permissions, enhancing the security of our system.

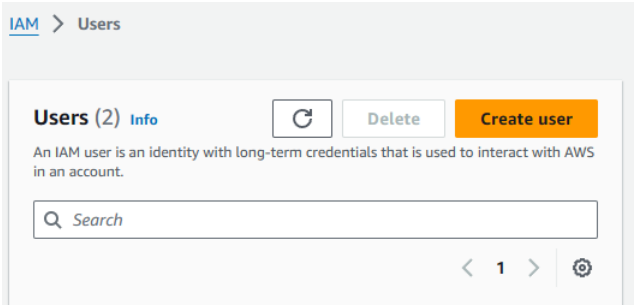

Fig. 10. Create new user in IAM dashboard

After establishing the IAM user, we proceeded to grant the necessary permissions to this user. This was achieved by attaching specific policies to the user account. Selectin "Attach policies directly" under the permissions options is important to be able to choose the required policies as shown Figure.11. We selected policies such as 'AmazonKinesisVideoStreamsFullAccess', 'AmazonDynamoDBFullAccess', and 'AmazonS3FullAccess'. Each of these policies provides comprehensive access rights to their respective services:

- AmazonKinesisVideoStreamsFullAccess: Grants full access to Kinesis Video Streams, allowing our system to manage and stream video data.
- AmazonDynamoDBFullAccess: Allows complete interaction with DynamoDB, which we use for data management and storage.
- AmazonS3FullAccess: Provides full access to Amazon S3, essential for storing and retrieving data, particularly the video files in our project.

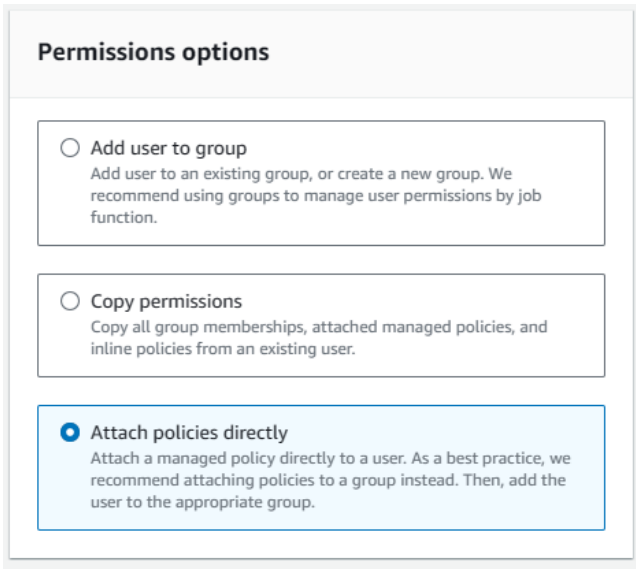

Fig. 11. Set permissions to create new user in IAM service

Finally, we generated access keys for this IAM user. Access keys in AWS are composed of an Access Key ID and a Secret Access Key, which are used to sign programmatic requests to AWS. We ensured that these keys were securely stored, as they are vital for our system's ability to interact with AWS services. The Access Key and Secret Key were used in our code to authenticate and authorize API requests to AWS services. It's important to note that these keys should be handled with utmost security and should never be hardcoded into the application or exposed in any publicly accessible areas. As part of integrating our IoT system with AWS services, a vital step involved configuring our JavaScript-based web dashboard to interact with AWS. This process was crucial for ensuring seamless communication between our web interface and the AWS backend, facilitating functionalities like video streaming, data storage, and device management. In the JavaScript code powering our dashboard that integrated the Access Key ID and Secret Access Key that obtained from AWS IAM after creating the user in the user summary dashboard as shown in Figure.12.

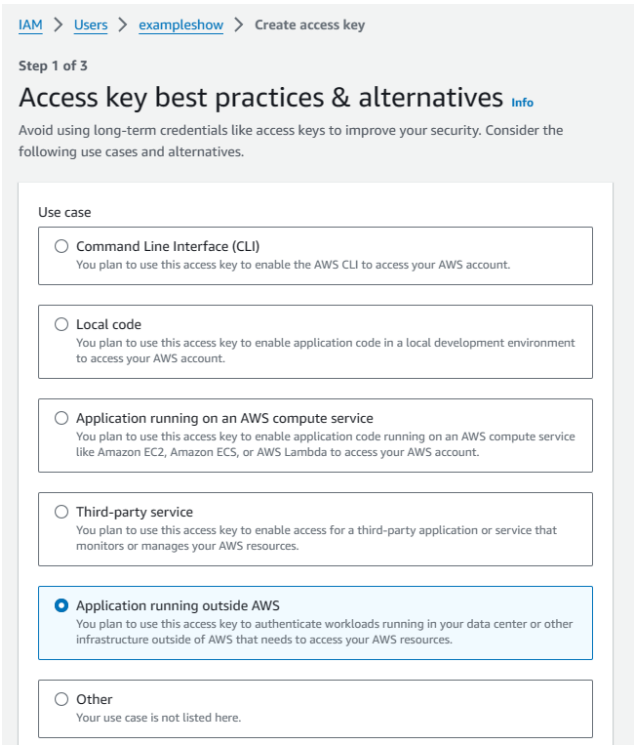

Fig. 12. Access key best practices & alternatives dashboard

These keys are essential for authenticating requests made from our web dashboard to AWS services. The full version of the code will be in the appendix of this report. It's crucial to handle these keys securely. In our implementation, we ensured that the keys were not hard-coded directly into the JavaScript files. Instead, they were securely stored and retrieved in a manner that minimizes security risks, such as using environment variables or secure server-side processes. Another essential aspect of the integration is setting the correct AWS region in our JavaScript code. The region should correspond to the location of the AWS services we are accessing. It's important because AWS services are region-scoped, meaning they operate in specific geographic areas as shown in the following code.

```
var secretAccessKey =
'PLACE_secretAccessKey_HERE';
var accessKeyId =
'PLACE_accessKeyId_HERE';
var region = 'PLACE_YOUR_REGION';
```

In our JavaScript dashboard, we configured the AWS region to match the region where our Kinesis Video Stream, DynamoDB, and S3 services were set up. This ensures that our web application communicates with the correct AWS data centers, thereby reducing latency and improving efficiency. This integration was a key component of our project, as it enabled our web-based dashboard to effectively interact with AWS services. By securely configuring the Access Key, Secret Access Key, and AWS region, we established and secure connection between our frontend and AWS's cloud

infrastructure, ensuring reliable and efficient system operations.

The integration with AWS IoT Core began with the registration of our IoT devices within the AWS ecosystem. This process is crucial for enabling secure and managed communication between the devices and the AWS cloud. In AWS IoT Core, each IoT device is represented as a “Thing” This virtual representation includes the device's attributes, capabilities, and metadata. We navigated to the AWS IoT Core service and chose the 'Manage' option, then “Things” Here, we selected 'Create' to initiate the creation of a new “Thing” as shown in Figure.12.

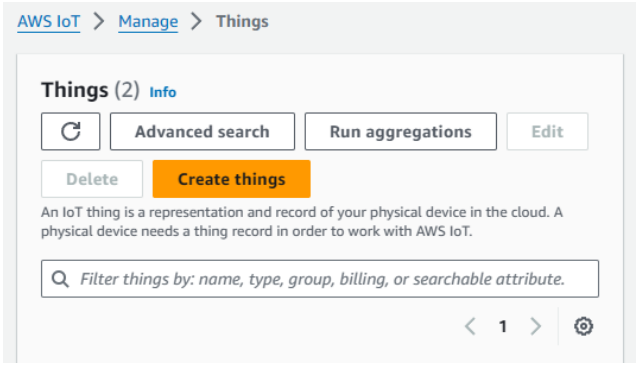

Fig. 12. Create new thing in Amazon IoT Core service

Each “Thing” was named appropriately to reflect its role in our project. We adhered to a naming convention that made it easy to identify and manage the devices. Depending on the scale of our project, we had the option to create a single “Thing” for an individual device or multiple “Things” for several devices. Our choice was based on the project's requirements and the number of devices deployed. After creating a “Thing” AWS IoT Core allows the generation and download of certificates and keys as shown in Figure.13. These are critical for establishing a secure TLS connection between the device and AWS services. We followed the prompt to download these certificates and keys immediately upon 'Thing' creation, as they cannot be retrieved later. The downloaded certificates and keys were securely stored on the devices. This step is essential to prevent unauthorized access and ensure secure device-to-cloud communication. In Addition, make sure that after downloading these certificates and key to be in the same files of the code to inherent it in our code that attached in the appendix section. Establishing policies for each 'Thing' is a key step in managing access and permissions. While, policies in AWS IoT Core define the permissions for the devices, specifying what they can and cannot do. We navigated to the 'Security' tab within IoT Core and selected 'Policies.' Here, we created a new policy by defining the actions (like 'connect', 'publish', 'subscribe', 'receive') and resources the device should have access to. We adopted a principle of least privilege, ensuring that each 'Thing' had only the necessary permissions to perform its function. After that, created policies were then attached to the certificates associated with each 'Thing.' This attachment is what enforces the defined permissions and ensures secure and controlled device operations.

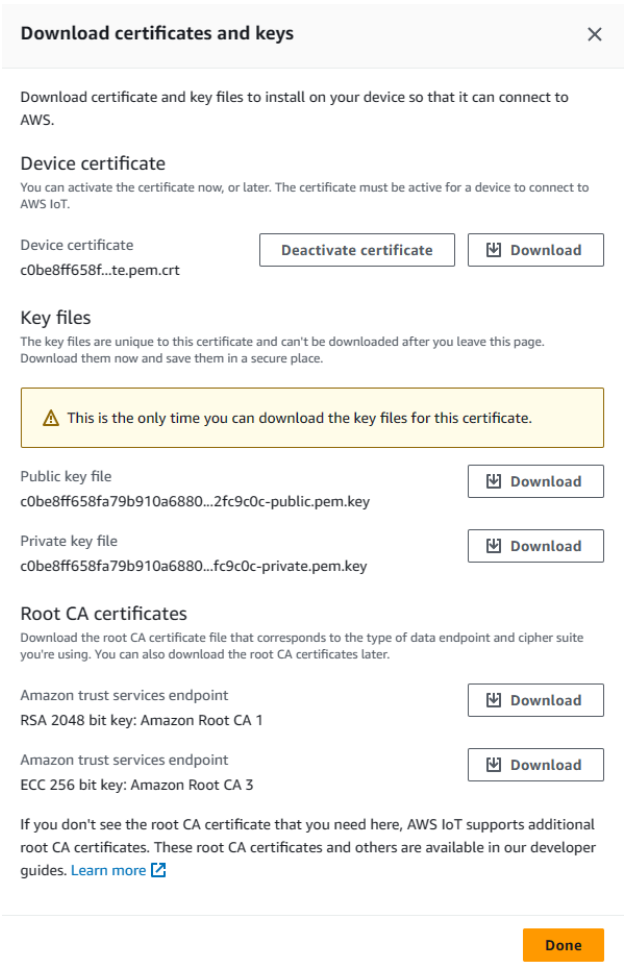

Fig. 13. Downloadable certificates and keys that will only popup for one time after Thing creation

Our project involved the efficient routing of messages from IoT devices to AWS services. To achieve this, we leveraged AWS IoT Core’s rules and roles. A role in AWS IoT Core is required to grant the IoT rule permission to interact with other AWS services, such as DynamoDB. We created a new IAM role specifically for IoT message routing. This role was designed to have the necessary permissions to write data to DynamoDB. Create new rule as shown in Figure.14.

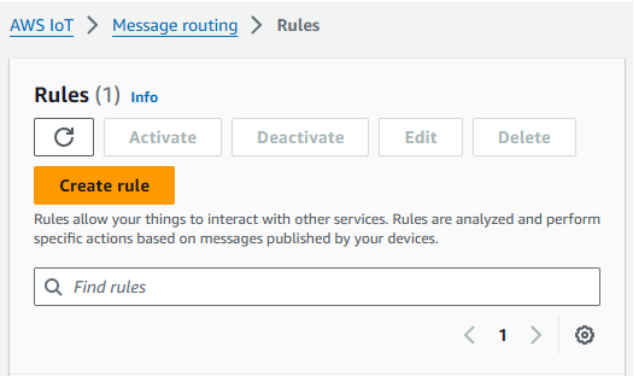

Fig. 14. Create new rule dashboard from AWS IoT Core

Within AWS IoT Core, we defined rules using SQL-like statements as shown in Figure.15. These rules dictate how the incoming IoT data is processed and routed. For instance, a rule could be set up to forward data to DynamoDB when certain conditions are met, such as a specific sensor value being recorded.

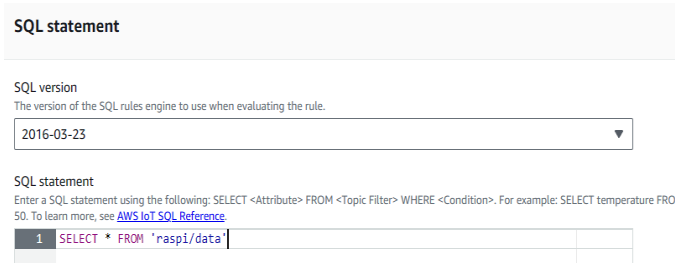

Fig. 15. SQL statement

We chose DynamoDB due to its ability to handle high-velocity data influxes which is typical in IoT applications and it will enable the power MQTT. Its fully managed nature meant that we could focus on application logic without worrying about the underlying database administration. DynamoDB's fast and predictable performance along with its ability to scale automatically, made it an ideal choice for storing and retrieving IoT data. In the AWS Management Console, under the DynamoDB service, we created a new table as shown in Figure.16. The table was named appropriately to reflect the data it would store, such as sensor readings or motion events. We defined a primary key that best suited our data structure, ensuring efficient access and querying capabilities. Additional attributes were defined based on the type of data we were storing. The IAM role created for message routing in AWS IoT Core was assigned to our DynamoDB table. This role grants the necessary permissions for AWS IoT Core to write data to the table. By controlling the access through IAM roles, we ensured that only authorized services and users could access or modify the data in DynamoDB, maintaining the integrity and security of our data.

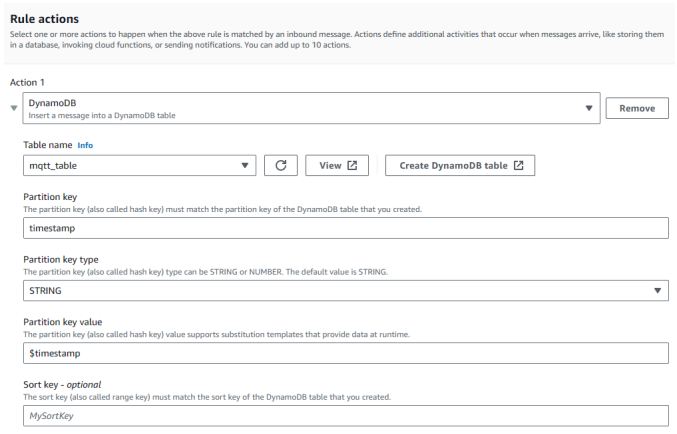

Fig. 16. Rule actions creation including DynamoDB table

Amazon S3 (Simple Storage Service) was chosen for its scalability, security, and performance for storing and retrieving any amount of data. The first step in integrating S3 into our project involved creating a new S3 bucket. We started by navigating to the Amazon S3 service within the AWS Management Console. Using the 'Create Bucket' option, we began the process of setting up a new storage container. Then, we chose a unique and descriptive name for the bucket, adhering to AWS naming conventions. This name helps in easily identifying the bucket's purpose and its association with our project. Moreover, it is crucial to select the appropriate AWS region for our bucket. The region choice affects the data latency and compliance with data sovereignty laws. We chose a region closest to our user base to optimize access speeds. Next, configuring the access and public settings of the S3 bucket was a critical step, ensuring the right balance between accessibility and security. Where, ACLs are used to manage access to bucket contents. We enabled ACLs on our bucket to have granular control over who can access the data stored in it as shown in Figure.17. By using ACLs, we could specify permissions for individual objects within the bucket, offering flexibility in controlling access to different types of data.

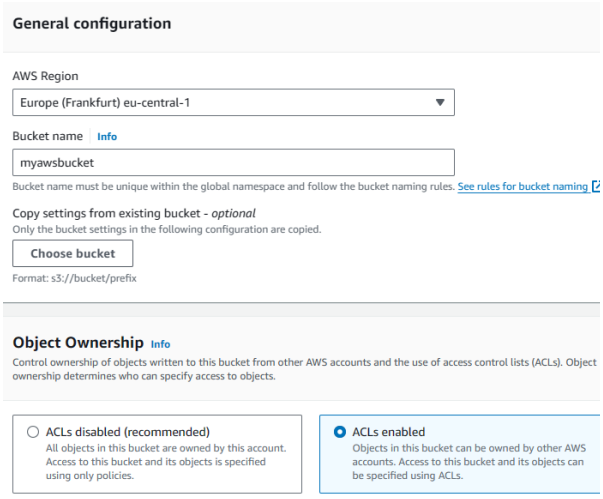

Fig. 17. Bucket Creation General Configuration

The decision to allow public access to the bucket was made based on the nature of the data and the project requirements. For instance, if the bucket contains publicly sharable data, such as public video streams, enabling public access is beneficial. In the 'Block Public Access' settings for the bucket, we carefully configured the permissions. We ensured that public access was allowed for specific use cases while maintaining the security of sensitive data as shown in Figure.18. Despite enabling public access for certain scenarios, we implemented additional security measures, such as bucket policies and IAM roles, to prevent unauthorized access or data breaches. Finally, setting up the Amazon S3 bucket was a fundamental aspect of our project, providing a secure and scalable solution for storing large volumes of data generated by our IoT system. The careful configuration of ACLs and public access settings ensured that we had the right mix of accessibility and security tailored to our project's needs.

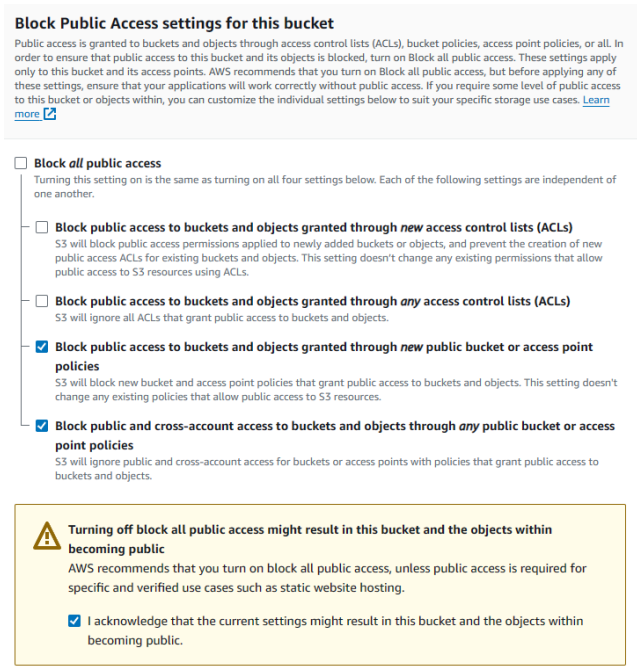

Fig. 18. Block Public Access settings for new bucket creation

Through careful configuration and integration of these AWS services, we successfully established strong infrastructure for our IoT motion detection system. This setup enabled us to stream video content, store and manage data effectively, and ensure secure and efficient communication between the various components of our system.

E. MQTT Setup

After implementing the AWS services in prober way it's essential to apply MQTT (Message Queuing Telemetry Transport) protocol. We chose the MQTT protocol for this project duo to its lightweight nature and effectiveness in IoT scenarios [13]. The integration involved configuring our Python application to communicate with AWS IoT Core using MQTT on QoS (Quality of Service) level 0. First, to securely communicate with AWS IoT Core, it was necessary to first set up and install the required certificates and keys. We used a Root CA certificate "iotRootCA1.pem", a public certificate "deviceCert.crt", and a private key "deviceCert.key", which were placed in a "certs" directory. We used the "paho-mqtt" library in Python, which is a popular MQTT client for Python applications. The client was configured to use TLS (Transport Layer Security) for secure communication. This involved setting the TLS context with the previously mentioned certificates and specifying the TLS version as "ssl.PROTOCOL_TLSv1_2". A callback function "on_connect" was defined to handle the event of successfully connecting to the MQTT broker. This function updates the "connflag" variable to indicate an active connection.

```
connflag = False
def on_connect(client, userdata, flags,
response_code):
    global connflag
    connflag = True
    print("Connected with status:
{0}".format(response_code))
```

Where, MQTT client connects to the AWS IoT broker using the specified URL "mqtt_url" which can be found from AWS IoT Core in the MQTT test client in the connection details under "Endpoint", and the standard MQTT port 8883 for secure communication. The client then starts a network loop, which allows it to process incoming and outgoing messages. In a continuous while loop, the application checks for a successful connection. Once connected, it periodically publishes a JSON-formatted message to a specified topic "(raspi/data)". The message contains simulated sensor data, a timestamp, and additional information. This data is serialized into JSON format using Python's "json" module. Moreover, for our application, we used QoS level 0 for message publishing. This level, also known as "At most once" delivery, is a fire-and-forget approach that does not require acknowledgment of message receipt, suitable for our use-case where occasional message loss is acceptable and minimal overhead is desired. This implementation of MQTT protocol with AWS IoT Core using Python provided a robust and efficient way to send data from our IoT devices to the cloud. The use of secure certificates and TLS ensured that our device-to-cloud communication remained secure, while the lightweight MQTT protocol facilitated reliable data transmission even with limited bandwidth where the code is provided in the appendix section.

VI. Results and Discussion

A. Functionality Testing & Interface Efficiency

After implementing the motion detection system and integrating it with AWS services, we conducted thorough testing to evaluate its functionality and the efficiency of the website dashboard interface. The dashboard is the central control panel for users, effectively displaying real-time data and providing control options. It is divided into three primary sections: Real-Time video Stream, Inbox Section, and Control Section. At the real-time video stream area showcases the live video feed from the Raspberry Pi camera. Also, the successful implementation of the HLS (HTTP Live Streaming) protocol ensures smooth and continuous video streaming, with minimal latency and buffering issues. We evaluated the streaming quality under various network conditions to ensure consistent performance. while Figure.19 show Raspberry Pi through the process and the real-time video stream is shown in Figure.20.

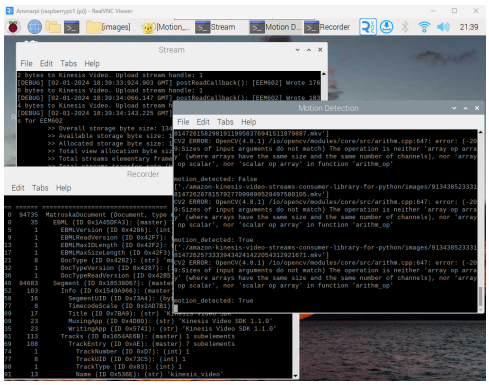

Fig. 19. Raspberry Pi Streaming and detecting motion while sending info. to AWS Service

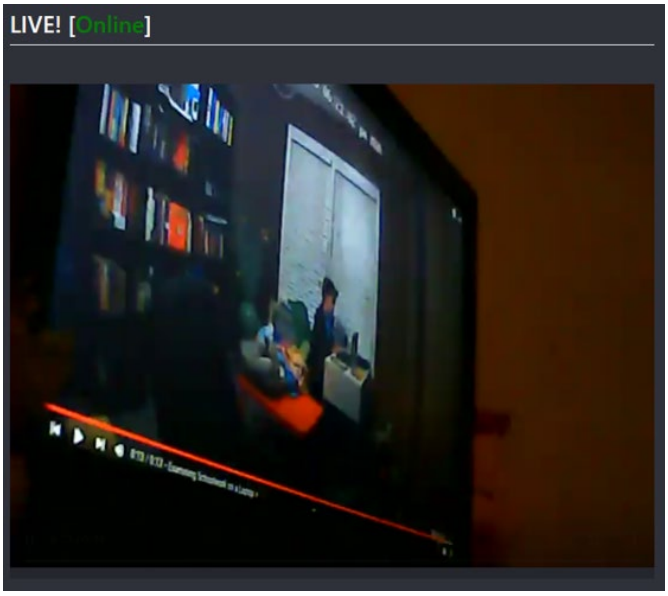

Fig. 20. Real-time video stream from the website that it connected from AWS Service us

Next, is the inbox section where this segment displays notifications and details of motion detection events. Once motion is detected, the footage is uploaded to AWS S3, and the corresponding details are displayed here. The exact time and details of the motion event are captured using MQTT protocol, enhancing the accuracy of information. We tested the reliability of motion detection alerts and the seamless uploading of video clips to S3. The focus was on the timeliness of alerts and the integrity of the data stored. Figure.21 showing this section implemented in our website dashboard.

Fig. 21. Inbox section from our website dashboard that using MQTT, and S3 using AWS & Python Code

In addition, each alert in the Inbox Section represents a specific motion detection event, complete with a timestamp and a brief description. When a user clicks on an alert, it triggers a pop-up window as shown in Figure.22. The pop-up window displays the video footage associated with the selected motion event. This footage is streamed from the AWS S3 storage where it was automatically uploaded during the motion detection process. The video can be played, paused, and viewed in full-screen mode for detailed examination. This feature provides immediate access to recorded events, enhancing the user's ability to monitor and review surveillance footage. Users can easily navigate through different recorded events, making the system more efficient for security monitoring. Moreover, Rigorous testing was conducted to ensure that the pop-up window loads quickly and the streaming from AWS S3 is smooth and uninterrupted. The user interface was tested for responsiveness and ease of use, ensuring that the feature is accessible across various devices and screen sizes. The integration with AWS S3 for storing recorded videos plays a crucial role in this feature. It ensures reliable and secure storage of surveillance footage, which is readily accessible through the user interface.

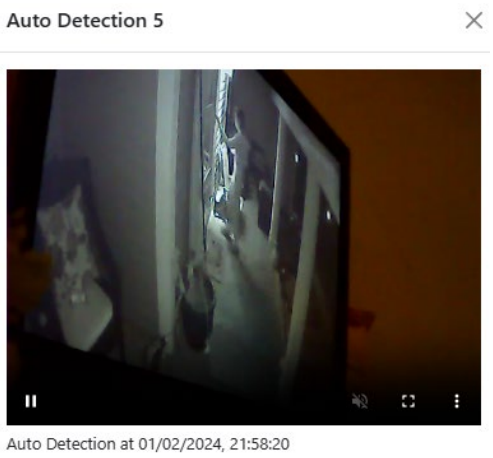

Fig. 22. Auto Detection after motion detection popup

Then, control section where users can schedule the operational hours of the camera, and toggle the streaming functionality on and off. Additionally, the delay time for the motion detection algorithm can be adjusted. The responsiveness and effectiveness of these controls were tested, ensuring that changes were applied in real-time and functioned as expected. Figure.23 showing the inbox section in our website dashboard.

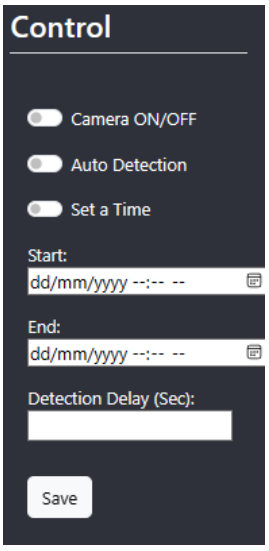

Fig. 23. Control section from website dashboard to control different settings

The dashboard design is user-friendly, with intuitive navigation and clear categorization of different functionalities. We assessed the ease of use, response time of the interface, and the clarity of the information presented. The website interface is responsive, adapting to different screen sizes and devices. Cross-device compatibility tests were conducted to

ensure consistent user experience across various platforms. Also, Figure.24 shows the full website dashboard while everything is implemented successfully.

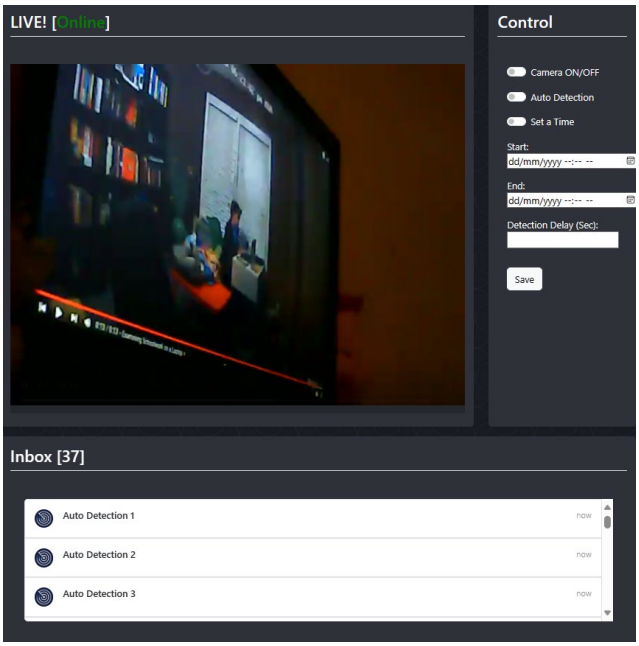

Fig. 24. Website dashboard successfully working

The functionality testing of our IoT motion detection system and its website dashboard demonstrated high efficiency and reliability. The Real-Time Video section successfully delivers live footage with HLS protocol, the Inbox Section accurately records and displays motion events with AWS integration, and the Control Section offers effective management of the camera and algorithm settings. Overall, the interface's user-centric design and responsive nature contribute significantly to its efficiency, making it a powerful tool for users to interact with and control their IoT-based security system.

B. Challenges & Solutions

In the initial stages of the project, there was a significant hurdle related to hardware integration. Despite the following standard procedures for connecting and setting up the web camera with Raspberry Pi where the system failed to recognize the camera. This issue was critical as the camera's functionality is central to our motion detection system. To diagnose the problem, we first verified the physical connection and ensured that the camera module is correctly attached to the Raspberry Pi. After confirming the hardware setup, we turned our attention to the software aspect. We hypothesized that the issue might stem from the Raspberry Pi's operating system. The Raspberry Pi was running on the default OS version which suspected that it might not fully support the camera functionalities required for our project. So, we decided to switch the operation system to a version that better suited for our needs. This involved formatting the MicroSD card to remove the existing operation system and then installing the Raspberry Pi Legacy OS version, known for its support for camera modules. After that, we ensured that the camera was enabled in the Raspberry Pi configuration settings. In addition, to verify the camera's functionality we used 'libcamera' and 'ripcam' software to test and confirm that the camera was now detectable and operational following the official Raspberry Pi camera documentation [14]. This approach successfully resolved the issue. The Raspberry Pi Legacy OS provided the necessary support for the camera, and our subsequent test confirmed that the camera was fully functional. This resolution was a pivotal moment in our project as it allowed us to proceed with the core development of the motion detection system. Moreover, this challenge underscored the importance of compatibility between hardware components and the operating system in IoT projects. Also, it highlighted the necessity of thorough testing and validation at every step of the system setup.

During the development we encountered a critical challenge with the Raspberry Pi camera module. Despite proper software configurations and multiple attempts at troubleshooting, the camera module remained undetectable by the Raspberry Pi. To isolate the cause of the problem we initially focused on software-based solutions, examining various methods and setting within the Raspberry Pi operation system. However, when these efforts did not yield results so we shifted our focus to potential hardware issues. We decided to conduct a thorough examination of the physical connections and we used a multimeter device to check the continuity and integrity of the camera module's wiring. The multimeter test revealed a crucial insight where the camera module's wire was defective, which explained the connectivity issues we were facing. With this knowledge, we decided to replace the malfunctioning camera module with considering the project timeline and resource availability. We opted for a readily accessible alternative a new webcam compatible with the Raspberry Pi. This decision was made to ensure minimal disruption to the project progress. Replacing the Raspberry Pi camera module with a new webcam proved to be a successful solution. After applying the software setting to the new camera, the new camera immediately recognized by the Raspberry Pi and we were able to proceed with the implementation of the motion detection functionality. This not only resolved the immediate issue but also reinforced the importance of having contingency plans and flexibility in hardware selection for IoT projects.

One of the significant challenges we encountered in our project was establishing a connection between the Raspberry Pi camera and AWS Kinesis Video Streaming. This step was crucial for enabling real-time video streaming capabilities in our IoT system. Where, the complexity of this challenge lay in the intricacies of configuring both the hardware (Raspberry Pi camera) and the AWS services to work seamlessly together. The initial attempts to establish this connection were met with difficulties, hindering the progress of our streaming functionality. To resolve this issue, our first step was to revisit the IAM (Identity and Access Management) configurations in AWS. We realized that the IAM roles and permissions associated with our AWS account needed adjustments to allow the Raspberry Pi camera to communicate effectively with Kinesis Video Streaming. Alongside modifying the IAM roles, we closely followed the guidelines provided by Amazon for connecting devices to Kinesis Video Streaming. These guidelines offered a structured approach and best practices for setting up the connection. So, we edited the IAM roles to grant the necessary permissions that would enable the Raspberry Pi camera to stream video data to the AWS service. This involved ensuring that the IAM roles had policies allowing access to Kinesis Video Streaming. Following the Amazon guide [15], we implemented the recommended settings and configurations on our Raspberry Pi system. This included

installing necessary libraries and configuring the network settings as per AWS requirements as shown in the following commands that required to be written in the Raspberry Pi Terminal.

```
$ export GST_PLUGIN_PATH=Directory Where You Cloned the SDK/amazon-kinesis-video-streams-producer-sdk-cpp/build
$ export AWS_DEFAULT_REGION=AWS Region i.e. us-east-1
$ export AWS_ACCESS_KEY_ID=Access Key ID
$ export AWS_SECRET_ACCESS_KEY=Secret Access Key
$ ./kvs_gstreamer_sample Your Stream Name
```

The combination of adjusting IAM roles and adhering to Amazon's guidelines proved to be successful. We were able to establish a stable and efficient connection between the Raspberry Pi camera and AWS Kinesis Video Streaming. This resolution was a pivotal moment in our project, as it not only solved the immediate problem but also provided us with valuable insights into the integration of IoT devices with cloud services. The successful implementation of this connection was key to enabling the real-time video streaming functionality of our system. Moreover, this resolution was a pivotal moment in our project, as it not only solved the immediate problem but also provided us with valuable insights into the integration of IoT devices with cloud services. The successful implementation of this connection was key to enabling the real-time video streaming functionality of our system. This challenge highlighted the importance of understanding the nuances of cloud services and their integration with IoT devices. It also underscored the value of thorough documentation and following best practices in technology integration.

In an advanced phase of our project, we faced a complex challenge when integration real-time streaming with motion detection functionality. The Raspberry Pi camera was already being used for live streaming to AWS services specifically AWS Kinesis Video Stream. We encountered a limitation where the Raspberry Pi camera could not be opened simultaneously by two different applications which impeded our ability to apply the motion detection code to the live stream and record motion triggered events. Understanding the criticality of simultaneous streaming and motion detection for our project's success, we embarked on a solution exploration. Our aim was to find a way to enable motion detection on the live stream without disrupting the ongoing streaming service. After extensive research, we discovered a GitHub repository [16] with code that appeared promising for our needs. We adapted the code from GitHub to enable continuous recording of short clips from the active stream on AWS Kinesis Video Stream. Additional This solution allowed us to bypass the limitation of accessing the camera by two applications simultaneously. We then developed a Python script tailored to our requirements which it will be fully provided in the appendix. This script would process each recorded clip and applying our motion detection algorithm. On detecting motion in a clip, the script was programmed to upload the relevant footage to AWS S3 service that served our storage solution. Conversely, if no motion was detected the script would automatically delete the clip to conserve storage space and system resources. The upcoming is an insight through the code implementation. The script is using the exactly same motion detection code that it already explained in the Implementation section of this report. In addition, it integrates with AWS S3 which is a cloud storage service that uploads videos that contain motion events. The script runs in a continuous loop, constantly checking for new video clips to analyze. Moreover, it regularly checks a specified directory for new video files (specifically '.mkv' files). In addition, if any motion detection found it will rename the video file with a timestamp, it will call the 'upload_video' function to upload the video to AWS S3 service, and it will delete the video from the local storage if it has been successfully uploaded. On the other hand, the if no motion detected the video will be removed from the local storage to save space. This innovative approach successfully resolved our problem. We were able to maintain a continuous live stream while effectively implementing motion detection on the stream. The system's ability to selectively store only motion-triggered events on AWS S3 significantly optimized our storage usage and made the system more efficient. Moreover, this challenge was a testament to the complexities inherent in integrating multiple advanced technologies such as real-time streaming, motion detection, and cloud services. It highlighted the necessity of creative problem-solving in IoT projects and the value of open-source resources like GitHub. Additionally, this experience reinforced the importance of flexible thinking and adaptability in the face of technical constraints. Our success in overcoming this obstacle has not only improved our system's functionality but also enriched the technical expertise and problem-solving skills.

VII. Conclusion

The project has effectively showcased the potential of integrating Internet of Things (IoT) technology with advanced security systems. Utilizing a Raspberry Pi 4 and web cameras, the project successfully developed a sophisticated motion detection system, demonstrating the feasibility and efficiency of IoT in real-world security applications. The integration of AWS cloud services further enhanced the system's capabilities, providing a robust, scalable, and secure backend for data management and device control. This project achieved an implementation of an advanced motion detection system using IoT devices, seamless integration of Raspberry Pi with cloud services for data handling and storage, real-time system monitoring and management enabled through a user-friendly dashboard, and overcoming technical challenges related to hardware integration and streaming protocols. The project represents a significant step forward in the field of IoT security systems. By using the power of IoT and cloud computing, it demonstrates how traditional security systems can be transformed into more intelligent, efficient, and responsive solutions. This project serves as a blueprint for future IoT security systems, highlighting the potential for enhanced safety, real-time monitoring, and remote management capabilities. For future enhancements, the project could explore different aspects like the integrating machine learning for smarter motion detection and predictive analytics, optimizing the system for lower power consumption, ensuring the system can work with various IoT devices and cameras in the same time, incorporating stronger encryption and privacy measures, and developing a more adaptable and feature-rich user dashboard. The success of the project is a testament to the innovative application of IoT in security. It has not only achieved its primary goals but also provided valuable lessons in IoT system design, cloud integration, and problem-solving under real-world constraints. The project underscores the importance of adaptability, continuous learning, and the effective use of emerging

technologies in addressing modern security challenges. As what is popular in IoT-based security systems, it sets a high standard for future developments in this field while highlighting the limitless possibilities of IoT technology in enhancing safety and security in our increasingly connected world.

ACKNOWLEDGMENT

Special thanks go to University of Bahrain for providing essential technical support, and to my family and friends for their unwavering encouragement and support throughout this endeavor. This project is a testament to the collaborative efforts and steadfast support of each individual involved.

REFERENCES

[1] A. R. Syafeeza, M. K. Mohd Fitri Alif, Y. Nursyifaa Athirah, A. S. Jaafar, A. H. Norihan, and M. S. Saleha, "IoT based facial recognition door access control home security system using raspberry pi," *International Journal of Power Electronics and Drive Systems*, vol. 11, no. 1, pp. 417–424, Mar. 2020, doi: 10.11591/ijpeds.v11.i1.pp417-424.

[2] H. H. Qasim, A. E. Hamza, L. Audah, H. H. Ibrahim, H. A. Saeed, and M. I. Hamzah, "Design and implementation home security system and monitoring by using wireless sensor networks WSN/internet of things IoT," *International Journal of Electrical and Computer Engineering*, vol. 10, no. 3, pp. 2617–2624, Aug. 2020, doi: 10.11591/ijece.v10i3.pp2617-2624.

[3] C. Sisavath and L. Yu, "Design and implementation of security system for smart home based on IOT technology," in *Procedia Computer Science*, Elsevier B.V., 2021, pp. 4–13. doi: 10.1016/j.procs.2021.02.023.

[4] Z. Tian, L. Zhang, G. Wang, and X. Wang, "An RGB camera-based fall detection algorithm in complex home environments," *Interdisciplinary Nursing Research*, vol. 1, no. 1, pp. 14–26, Nov. 2022, doi: 10.1097/nr9.0000000000000007.

[5] K. Bjerge, C. E. Frigaard, and H. Karstoft, "Object Detection of Small Insects in Time-Lapse Camera Recordings," *Sensors*, vol. 23, no. 16, Aug. 2023, doi: 10.3390/s23167242.

[6] E. Wu, "Meet The Brand New Raspberry Pi 4 8GB - Latest Open Tech From Seeed." Accessed: Dec. 29, 2023. [Online]. Available: https://www.seeedstudio.com/blog/2020/05/28/meet-the-brand-new-raspberry-pi-4-8gb-ram/

[7] Lucy, "How does VNC technology work? – RealVNC Help Center." Accessed: Dec. 30, 2023. [Online]. Available: https://help.realvnc.com/hc/en-us/articles/360002320638-How-does-VNC-technology-work-

[8] O. Jimmy, "What is PuTTY and some useful tips to use it easily." Accessed: Dec. 30, 2023. [Online]. Available: https://pandorafms.com/blog/putty/

[9] Sharon Shea, "How to prevent network eavesdropping attacks | TechTarget." Accessed: Dec. 30, 2023. [Online]. Available: https://www.techtarget.com/searchsecurity/answer/How-to-prevent-network-sniffing-and-eavesdropping

[10] "Raspberry Pi OS – Raspberry Pi." Accessed: Dec. 31, 2023. [Online]. Available: https://www.raspberrypi.com/software/

[11] "Download PuTTY - a free SSH and telnet client for Windows." Accessed: Jan. 01, 2024. [Online]. Available: https://putty.org/

[12] "Download VNC Viewer for Raspberry Pi | VNC® Connect." Accessed: Jan. 01, 2024. [Online]. Available: https://www.realvnc.com/en/connect/download/viewer/raspberrypi/

[13] B. Mishra and A. Kertesz, "The use of MQTT in M2M and IoT systems: A survey," *IEEE Access*, vol. 8, pp. 201071–201086, 2020, doi: 10.1109/ACCESS.2020.3035849.

[14] "Raspberry Pi Documentation - Camera software." Accessed: Jan. 01, 2024. [Online]. Available: https://www.raspberrypi.com/documentation/computers/camera_software.html

[15] "Stream video to your Kinesis video stream and view the live stream - Amazon Kinesis Video Streams." Accessed: Jan. 02, 2024. [Online]. Available: https://docs.aws.amazon.com/kinesisvideostreams/latest/dg/producersdk-cpp-rpi-run.html

[16] "aws-samples/amazon-kinesis-video-streams-consumer-library-for-python." Accessed: Jan. 01, 2024. [Online]. Available: https://github.com/aws-samples/amazon-kinesis-video-streams-consumer-library-for-python/tree/main

VIII. Appendix

A. AWS Service (MQTT & S3) & Motion Detection Code

```
import cv2
import os
from datetime import datetime
import boto3
from botocore.client import Config
import paho.mqtt.client as mqtt
import ssl
from time import sleep
import json

# AWS IoT Core credentials
mqtt_url = "a1tw66q2ooe9x7-ats.iot.eu-central-1.amazonaws.com"
root_ca = './certs/iotRootCA1.pem'
public_crt = './certs/deviceCert.crt'
private_key = './certs/deviceCert.key'

ACCESS_KEY_ID = "WRITE_YOUR_ACCESS_KEY_HERE"
ACCESS_SECRET_KEY = "WRITE_YOUR_ACCESS_SECRET_KEY_HERE"
BUCKET_NAME = "WRITE_YOUR_BUCKET_NAME_HERE"

connflag = False

motion_detected = False

videos_dir = r"./amazon-kinesis-video-streams-consumer-library-for-python/records"


def search_for_videos(videos_dir):
    files = os.listdir(videos_dir)
    videos = []

    # filter videos in dir
    for file in files:
        if file.endswith(".mkv") and str(file).count("Record of") <= 0:
            videos.append(os.path.join(videos_dir,file))
    print(videos)

    return videos


def on_connect(client, userdata, flags, response_code):
    global connflag
    connflag = True
    print("Connected with status: {0}".format(response_code))


def upload_video(video_path):
    video_name = str(video_path).split('/')[-1]
```

```
    data = open(video_path, 'rb')
    s3 = boto3.resource(
        's3',
        aws_access_key_id=ACCESS_KEY_ID,
        aws_secret_access_key=ACCESS_SECRET_KEY,
        config=Config(signature_version='s3v4')
    )
    s3.Bucket(BUCKET_NAME).put_object(Key=video_name, Body=data,
ContentType='video/mkv', ACL='public-read')
    return True

client = mqtt.Client()
client.tls_set(root_ca,
               certfile = public_crt,
               keyfile = private_key,
               cert_reqs = ssl.CERT_REQUIRED,
               tls_version = ssl.PROTOCOL_TLSv1_2,
               ciphers = None)

client.on_connect = on_connect

print ("Connecting to AWS IoT Broker...")
client.connect(mqtt_url, port = 8883, keepalive=60)
client.loop_start()

while True:
    videos = search_for_videos(videos_dir)
    for video in videos:
        motion_detected = False
        try:
            cap = cv2.VideoCapture(video)

            # Read the first frame
            ret, frame1 = cap.read()
            ret, frame2 = cap.read()

            while cap.isOpened():

                diff = cv2.absdiff(frame1, frame2)  # Find the absolute difference
between frames
                gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
                blur = cv2.GaussianBlur(gray, (5, 5), 0)  # Apply Gaussian blur
                _, thresh = cv2.threshold(blur, 20, 255, cv2.THRESH_BINARY)  # Apply
thresholding
                dilated = cv2.dilate(thresh, None, iterations=3)  # Dilate the
thresholded image
                contours, _ = cv2.findContours(dilated, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)  # Find contours

                for contour in contours:
```

```
                    (x, y, w, h) = cv2.boundingRect(contour)

                    if cv2.contourArea(contour) < 900:
                        continue  # Ignore small movements

                    cv2.rectangle(frame1, (x, y), (x+w, y+h), (0, 255, 0), 2)  # Draw
rectangle around motion area
                    motion_detected = True

                # cv2.imshow("feed", frame1)
                frame1 = frame2
                ret, frame2 = cap.read()

                if cv2.waitKey(40) == 27:  # Press 'Esc' key to exit
                    break
            except cv2.error as e:
                print("CV2 ERROR:", e)
            except Exception as ge:
                print(ge)
            finally:
                try:
                    cv2.destroyAllWindows()
                    cap.release()

                    print("motion_detected:", motion_detected)
                    # only upload the video when motion has been detected
                    if(motion_detected):

                        # change the name in it is not changed already
                        if str(video).count("Record of") <= 0:
                            now = datetime.now()
                            new_name = os.path.join(videos_dir, "Record of " +
now.strftime("%Y-%m-%d %H-%M-%S") + ".mkv")
                            os.rename(video, new_name)
                            video = new_name

                        # upload the video
                        uploaded = upload_video(video)
                        if connflag == True:
                            timestamp = datetime.now().strftime("%m/%d/%Y, %H:%M:%S")
                            data = json.dumps({"timestamp":timestamp, "disc":45,
"status":"GREEN"})
                            client.publish("raspi/data", payload=data, qos=0)
                        else:
                            print ("waiting for connection...")


                    # remove the video if there is no motion or it has been uploaded
                    if uploaded or not motion_detected:
                        os.remove(video)
```

```
            except Exception as e:
                print(e)
    sleep(5)
```

B. Website dashboard JavaScript Code

```
const express = require('express');
const app = express();
const server = require('http').Server(app);
const io = require('socket.io')(server);
const path = require('path');
const bodyParser = require('body-parser');
const aws = require('aws-sdk')
const { device: deviceModule } = require('aws-iot-device-sdk');
const multer = require('multer')
const multerS3 = require('multer-s3');

const port = 3000;

var HLS_URL = null;
var errorCode = null;
var DaynamoDB;

var secretAccessKey = 'WRITE_YOUR_ACCESS_SECRET_KEY_HERE';
var accessKeyId = 'WRITE_YOUR_ACCESS_KEY_HERE';
var region = 'WRITE_YOUR_REGION_HERE';
const BUCKET = 'WRITE_YOUR_BUCKET_NAME_HERE';
var tableName = "WRITE_YOUR_DYNAMODB_TABLE_NAME_HERE";

app.use(express.static(path.join(__dirname, 'public')));
app.use(bodyParser.urlencoded({ extended: true }));

app.set('view engine', 'ejs');

aws.config.update({
    secretAccessKey: secretAccessKey,
    accessKeyId: accessKeyId,
    region: region,

});

const s3 = new aws.S3();
var docClient = new aws.DynamoDB.DocumentClient();


const upload = multer({
    storage: multerS3({
        s3: s3,
        acl: "public-read",
        bucket: BUCKET,
```

```
        key: function (req, file, cb) {
            console.log(file);
            cb(null, file.originalname)
        }
    })
})
app.get('/', function(req, res){
    res.sendFile(__dirname + '/views/index.html');
})


app.get("/list", async (req, res) => {

    var dd;

    let r = await s3.listObjectsV2({ Bucket: BUCKET }).promise();
    let x = r.Contents.map(item => item.Key);

    var params = {
        TableName: tableName,
    };

    docClient.scan(params).eachPage((err, data, done) =>{
        if (err) {
            console.error("Unable to read item. Error JSON:", JSON.stringify(err, null,
2));
        } else {
            allData = JSON.stringify(data);
            items = JSON.parse(allData).Items;
            DaynamoDB = items.map(item => item.payload);
            aws_data = {
                s3: x,
                dynamodb: DaynamoDB
            }
            res.send(aws_data)

        }
    })



})


var CameraInput;
var DetectionInput;
var SetTimeInput;
var startTimeInput;
var endTimeInput;
var delayDetectionInput;
```

```
var isTrue = false;

app.post('/submit', function(req, res){
    res.sendFile(__dirname + '/views/index.html');
    CameraInput = req.body['Camera'];
    DetectionInput = req.body['Detection'];
    SetTimeInput = req.body['SetTime'];
    startTimeInput = req.body['startTime'];
    endTimeInput = req.body['endTime'];
    delayDetectionInput = req.body['delayDetection'];
    isTrue = true;
});


server.listen(port, async function(){
    console.log('Listening on port ' + port);

    io.on('connection', async function(socket){


        console.log('User Connected!!!');
        socket.emit("Camera", CameraInput);
        socket.emit("Detection", DetectionInput);
        socket.emit("SetTime", SetTimeInput);
        socket.emit("startTime", startTimeInput);
        socket.emit("endTime", endTimeInput);
        socket.emit("delayDetection", delayDetectionInput);

        setInterval(async () => {
            // AWS S3 Get Files Name
            let r = await s3.listObjectsV2({ Bucket: BUCKET }).promise();
            let x = r.Contents.map(item => item.Key);

            // AWS DynamoDB Get Data
            var params = {
                TableName: tableName,
            };

            docClient.scan(params).eachPage((err, data, done) =>{
                if (err) {
                    console.error("Unable to read item. Error JSON:",
JSON.stringify(err, null, 2));
                } else {
                    allData = JSON.stringify(data);
                    items = JSON.parse(allData).Items;
                    DaynamoDB = items.map(item => item.payload);
                }
            })

            aws_data = {
```

```
                s3: x,
                dynamodb: DaynamoDB
            }
            socket.emit("listLenth", aws_data);
        }, 1000);


        // send the stream HLS URL or 404
        KV_AWS();
        if(errorCode == 404){
            socket.emit("stream", errorCode);
        }
        else{
            socket.emit("stream", HLS_URL);
        }


    });

});
async function KV_AWS(){


    var streamName = 'STREAM_NAME';

    // Step 1: Configure SDK Clients
    var options = {
        accessKeyId: ' WRITE_YOUR_ACCESS_KEY_HERE ',
        secretAccessKey: 'WRITE_YOUR_ACCESS_SECRET_KEY_HERE',
        region: 'WRITE_YOUR_REGION_HERE'
    }

    var kinesisVideo = new aws.KinesisVideo(options);
    var kinesisVideoArchivedContent = new aws.KinesisVideoArchivedMedia(options);


    try{
      // Step 2: Get a data endpoint for the stream
    const kv_response = await kinesisVideo.getDataEndpoint({
      StreamName: streamName,
      APIName: "GET_HLS_STREAMING_SESSION_URL"
    }, function(err, response) {
      if (err) { return; } // console.error(err)
      kinesisVideoArchivedContent.endpoint = new aws.Endpoint(response.DataEndpoint);
    }).promise();

    // Step 3: Get an HLS Streaming Session URL
    var playbackMode = 'LIVE'; // 'LIVE' or 'ON_DEMAND'
    //var startTimestamp = new Date('START_TIMESTAMP'); // For ON_DEMAND only
    //var endTimestamp = new Date('END_TIMESTAMP'); // For ON_DEMAND only
```

```
    var fragmentSelectorType = 'SERVER_TIMESTAMP'; // 'SERVER_TIMESTAMP' or
'PRODUCER_TIMESTAMP'
    const SESSION_EXPIRATION_SECONDS = 60*60

    const hls_response = await kinesisVideoArchivedContent.getHLSStreamingSessionURL({
      StreamName: streamName,
      PlaybackMode: playbackMode,
      HLSFragmentSelector: {
          FragmentSelectorType: fragmentSelectorType,
          TimestampRange: playbackMode === 'LIVE' ? undefined : {
    //            StartTimestamp: startTimestamp,
    //            EndTimestamp: endTimestamp
          }
      },
      Expires: parseInt(SESSION_EXPIRATION_SECONDS)
    }, function(err, response) {
      if (err) {

        errorCode = err.statusCode;

        if(errorCode == 400){
            return hls_response;
        }

    }

    if(errorCode == null){
        HLS_URL = response.HLSStreamingSessionURL, response;
    }

    }
    ).promise();

    }
    catch(error){
    }

  }
```

C. Website Inbox JavaScript Code

```
var list = [];

$(function(){
    let socket = io();

        socket.on("listLenth", (data) => {
            var idNum = 0;
            var id;
```

```
            const container = document.getElementById('inboxMSG');
            var ChildLenght = $('#inboxMSG').children().length;

            if(data.s3.length > ChildLenght){
                var link = getLink('BUCKET_NAME', 'REIGON_NAME');
                s3Data = data.s3;
                DynamoDBData = data.dynamodb;

                var time = data.dynamodb.map(item => item.timestamp);

                $('.inboxLen').text('Inbox [' + s3Data.length + ']');

                for (let i = s3Data.length-1; i < s3Data.length; i++) {
                    id = 'msg' + i;
                    addRow(id, msg(i+1 ,id, link+s3Data[i], time[i]));

                }
            }


        });



});

getList('/list', (data)=>{
    var id;

    const container = document.getElementById('inboxMSG');
    removeAllChildNodes(container);

    var link = getLink('BUCKET_NAME', 'YOUR_REIGON');
    s3Data = data.s3;
    DynamoDBData = data.dynamodb;

    var time = data.dynamodb.map(item => item.timestamp);

    $('.inboxLen').text('Inbox [' + s3Data.length + ']');

    for (let i = 0; i < s3Data.length; i++) {
        id = 'msg' + i;
        addRow(id, msg(i+1 ,id, link+s3Data[i], time[i]));

    }
})

function addAllChildNodes(data){
    var id;
```

```
    const container = document.getElementById('inboxMSG');

    var link = getLink('BUCKET_NAME', 'REGION_NAME');
    rList = data;

    $('.inboxLen').text('Inbox [' + data.length + ']');

    for (let i = 0; i < data.length; i++) {
        id = 'msg' + i;
        addRow(id, msg(i+1 ,id, link+rList[i], rList[i]));

    }
}

function addRow(targetID, msg){
    const div = document.createElement('div');
    div.className = targetID + 'C';
    div.innerHTML = msg;
    list.push(div);
    document.getElementById('inboxMSG').appendChild(div);
}


function msg(num, targetID, link, time){

    return      `
        <a href="#" class="list-group-item list-group-item-action d-flex gap-3 py-3"
aria-current="true" data-bs-toggle="modal" data-bs-target="#${targetID}">
            <img src="/images/detection.jpg" alt="twbs" width="32" height="32"
class="rounded-circle flex-shrink-0">
            <div class="d-flex gap-2 w-100 justify-content-between">
                <div>
                    <h6 class="mb-0">Auto Detection ${num}</h6>
                </div>
                <small class="opacity-50 text-nowrap">now</small>
            </div>
        </a>

        <div class="modal fade" id="${targetID}" tabindex="-1" aria-
labelledby="exampleModalLabel" aria-hidden="true">
            <div class="modal-dialog">
                <div class="modal-content">
                    <div class="modal-header">
                        <h1 class="modal-title fs-5" id="exampleModalLabel">Auto
Detection ${num}</h1>
                        <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
                    </div>
                    <div class="modal-body">
```

```
                    <!--<img src="${link}" alt="Workplace" usemap="#workmap"
width="400" height="379">-->
                        <video class="inboxVideo" controls autoplay="true"
muted="true">
                            <source src="${link}" type="video/mp4">
                        </video>
                        <p>Auto Detection at ${time}</p>
                    </div>
                </div>
            </div>
        </div>
        `;
}


function getList(url, cb){
    const xhr = new XMLHttpRequest();
    xhr.onreadystatechange = status => {
        if(xhr.readyState == XMLHttpRequest.DONE){
            cb(JSON.parse(xhr.responseText));
        }
    }

    xhr.timeout = 10000;
    xhr.open('GET', url);
    xhr.send();
}


function getLink(IoTName, region){
    return `https://${IoTName}.s3.${region}.amazonaws.com/`;
}
```

D. Website Stream JavaScript Code

```
$(function(){
    let socket = io();

    socket.on("stream", (data) => {

        console.log('code: ' + data);

        var playerElement = $('#video-player');

        if(data == 404){
            playerElement.hide();
            $('#novideo').html('⚠Live Stream is Not Available⚠');
            $('#cameraStatus').html('Offline');
            $('#cameraStatus').css('color', 'red');
```

```
        }
        else{
            const playbackUrl = data;
            playerElement.show();
            var player = new Hls();
            console.log('Created HLS.js Player');
            player.loadSource(playbackUrl);
            player.attachMedia(playerElement[0]);
            console.log('Set player source');
            document.getElementById('video-player').autoplay = true;
            $('#cameraStatus').html('Online');
            $('#cameraStatus').css('color', 'green');
            player.on(Hls.Events.MANIFEST_PARSED, function() {
                player.play();
                console.log('Starting playback');
            });

        }


    });
});
```

E. Website HTML Main Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"
crossorigin="anonymous">
    <link rel="stylesheet" href="/css/style.css">
</head>
<body>

    <!-- text-center -->
    <div class="fullPageContainer">
        <div class="row">


          <!-- Live -->
          <div class="col-12 col-md-9 allContainer liveContainer">
            <div class=" my-3 p-3 contentContainer rounded shadow-sm content2">

                <h3 id="txtHint" class="border-bottom pb-2 mb-0 text">LIVE! [<span
id="cameraStatus">Offline</span>]</h3>
```

```
                <!-- Video -->
                <br>
                <div class="text-center">
                <h1 id="novideo"></h1>
                <!-- width="100%" height="500"  controls autoplay playsinline-->
                <video id="video-player"  class="videoContainer" controls
autoplay="true" muted="true">
                </video>




              </div>

            </div>
          </div>


          <!-- Controll -->

          <div class="col-12 col-md-3 allContainer">
            <div class="my-3 p-3 contentContainer rounded shadow-sm content2">
              <h3 class="border-bottom pb-2 mb-0 text">Control</h3>
              <div class="p-3">
                <!-- text-center -->
                <br>


                <form action="/submit" method="POST">

                  <div class="form-check form-switch py-2">
                    <input class="form-check-input" type="checkbox" id="Camera"
name="Camera">
                    <label class="text" for="Camera" name="Camera">Camera
ON/OFF</label>
                  </div>

                  <div class="form-check form-switch py-2">
                    <input class="form-check-input" type="checkbox" id="Detection"
name="Detection">
                    <label class="text" for="Detection" name="Detection">Auto
Detection</label>
                  </div>

                  <div class="form-check form-switch py-2">
                    <input class="form-check-input" type="checkbox" id="SetTime"
name="SetTime">
                    <label class="text" for="SetTime" name="SetTime">Set a Time</label>
```

```
                </div>

                <div class="py-2">
                  <label class="text" for="startTime">Start:</label>
                  <br>
                  <input type="datetime-local" id="startTime" name="startTime">
                </div>

                <div class="py-2">
                  <label class="text" for="endTime">End:</label>
                  <br>
                  <input type="datetime-local" id="endTime" name="endTime">
                </div>

                <div class="py-2">
                  <label class="text" for="delayDetection">Detection Delay
(Sec):</label>
                  <br>
                  <input type="number" id="delayDetection" name="delayDetection">
                </div>

                <br>
                <button type="submit" value="OK" class="btn btn-light">Save</button>

              </form>


                <!-- <button id="MSGbutton" type="button" class="btn btn-light"
name="btn11">add</button> -->



            </div>
          </div>
        </div>


        <!-- Alert -->
        <div class="col-12 allContainer ">
          <div class="my-3 p-3 contentContainer rounded shadow-sm">
            <h3 class="border-bottom pb-2 mb-0 text inboxLen">Inbox</h3>
            <div class="d-flex flex-column flex-md-row p-4 gap-4 py-md-5">
              <div class="list-group w-100" id="inboxMSG">

                <!-- alerts will added here -->
              </div>
            </div>
            <!-- End -->
```

```
            </div>
          </div>




        </div>
      </div>

      <script src="https://cdn.jsdelivr.net/npm/hls.js@latest"></script>
    <script src="https://player.live-video.net/1.22.0/amazon-ivs-
player.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/howler/2.1.2/howler.core.min.js"></script>

    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL"
crossorigin="anonymous"></script>
    <script src="/socket.io/socket.io.js"></script>
    <script src="/JS/stream.js"></script>
    <script src="/JS/inbox.js"></script>
    <script src="/JS/control.js"></script>




</body>
</html>
```