IoT

Article

A Smart Fire Detector IoT System with Extinguisher Class Recommendation Using Deep Learning

Tareq Khan

School of Engineering, Eastern Michigan University, Ypsilanti, MI 48197, USA; tareq.khan@emich.edu

Abstract: Fires kill and injure people, destroy residences, pollute the air, and cause economic loss. The damage of the fire can be reduced if we can detect the fire early and notify the firefighters as soon as possible. In this project, a novel Internet of Things (IoT)-based fire detector device is developed that automatically detects a fire, recognizes the object that is burning, finds out the class of fire extinguisher needed, and then sends notifications with location information to the user and the emergency responders smartphones within a second. This will help firefighters to arrive quickly with the correct fire extinguisher—thus, the spread of fire can be reduced. The device detects fire using a thermal camera and common objects using a red-green-blue (RGB) camera with a deep-learning-based algorithm. When a fire is detected, the device sends data using the Internet to a central server, and it then sends notifications to the smartphone apps. No smoke detector or fire alarm is available in the literature that can automatically suggest the class of fire extinguisher needed, and this research fills this gap. Prototypes of the fire detector device, the central server for the emergency responder's station, and smartphone apps have been developed and tested successfully.

Keywords: fire detection; deep learning; thermal camera; fire extinguisher class; Jetson Nano; smartphone app; SQL server; Bluetooth; Wi-Fi; push notification

check for updates

Citation: Khan, T. A Smart Fire Detector IoT System with Extinguisher Class Recommendation Using Deep Learning. *IoT* **2023**, *4*, 558–581. https://doi.org/10.3390/iot4040024

Academic Editor: Amiya Nayak

Received: 12 October 2023
Revised: 21 November 2023
Accepted: 23 November 2023
Published: 25 November 2023

1. Introduction

According to the National Fire Protection Association (NFPA), an estimated 358,500 home fires occur every year in the United States alone. House fires cause an average of 2620 civilian deaths and nearly 12 billion dollars in property damage each year in the USA. Among residential fires, 50% occur due to cooking, 12.5% from heating equipment, and 6.3% from electrical malfunction. Over 22% of non-residential fires are electrical fires, caused by short circuits or wiring problems [1]. One way to reduce the spread of fire is to detect the fire early and notify emergency responders with information about the fire as soon as possible. In this project, a novel fire detection device based on the Internet of Things (IoT) framework is developed. This device autonomously detects fire occurrences, identifies the burning objects, recommends the required class of fire extinguisher needed, and then transmits notifications to both end users and emergency responders via smartphones—all accomplished within a span of a second. This instrumental advancement serves to facilitate the prompt arrival of firefighters armed with the appropriate fire extinguisher, thereby effectively mitigating the propagation of fires. The novelty of this research lies in its ability to fill the gap of a smoke detector or fire alarm system in the literature that can autonomously recommend the required class of fire extinguisher. The device employs a thermal camera for fire detection, and a red-green-blue (RGB) camera with a deep-learning-based algorithm to detect common objects. Upon fire detection, the device uses the Internet to send data to a central server, and then the server disseminates push notifications to the designated smartphones. The proposed device could be used in homes, schools, factories, grocery stores, warehouses, etc. The overall operation of the proposed system is shown in Figure 1.

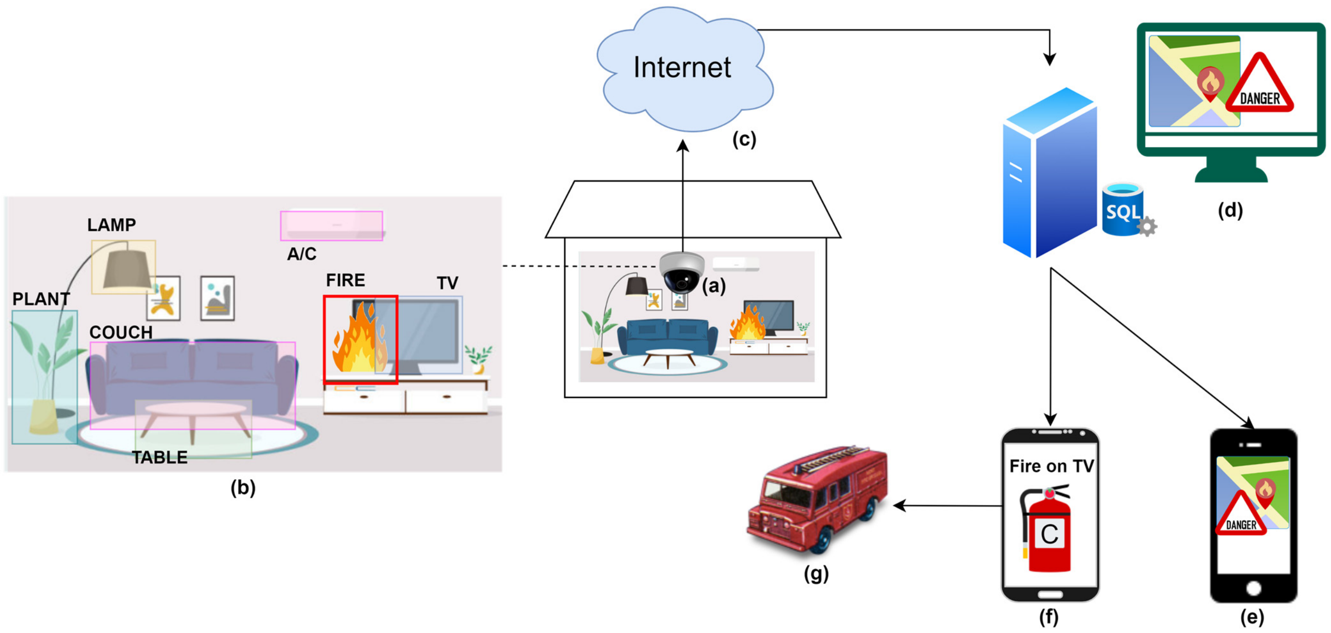


Figure 1. The proposed fire detector (**a**) captures the image of the room and detects common objects such as plants, couches, tables, lamps, TV, A/C, etc., and detects the fire using a thermal camera as shown in (**b**). When fire is detected, it sends data using the Internet (**c**) to the central server (**d**). The fire event location is marked on the map (**d**), saved in a database, and, depending upon the object on fire, it determines the type of fire extinguisher needed—for instance, class C for electrical fire on TV. The server then sends push notifications to the user's smartphone app (**e**) and the emergency responder's smartphone app (**f**). A firetruck (**g**) is dispatched.

The needs and significances of the proposed system are mentioned below:

- Traditional smoke detectors determine the presence of fire from smoke. If someone is cooking where smoke is generated, these smoke detectors produce false alarms [2]. In the proposed fire detector device, fire is detected using thermal camera images with a higher confidence level, and thus false alarms can be reduced.
- The smoke detectors have a high response time as smoke needs to travel to the detector. The proposed thermal-camera-based fire detector has a lower response time as light travels faster than smoke.
- When smoke is detected, traditional smoke detectors produce alarm sounds. If there is no one in the house and the fire starts from a leaking gas pipe or electrical short-circuits, then no one will hear the sound, and the fire will spread. In the proposed device, notifications will be sent to the users and the emergency responders using the Internet, so people will be notified even if they are away from home—thus, it will give peace of mind.
- Fire extinguishers are classified as types A, B, C, D, or K [3]. It is crucial to use the right type of extinguisher for the specific class of fire to avoid personal injury or damage to property. The wrong type of extinguisher could cause electrical shocks or explosions, or spread the fire [4]. The proposed device recognizes the object that is burning and suggests the class of fire extinguisher needed and then sends a notification with this information to emergency responders. Thus, the emergency responders know the type of fire extinguisher needed and can arrive at the site with the right fire extinguisher—thus, harm to life and property can be reduced.

The rest of the paper is organized as follows. Section 2 discusses the related works. In Section 3, materials and methods are discussed for detecting objects and fires, and recommending the extinguisher class, as is the prototype system architecture consisting of the device, central server, and smartphone apps. The simulation and prototype system's results are elaborated upon in Section 4, while Section 5 delves into the discussion and future works. Lastly, Section 6 provides a conclusion.

2. Literature Review

The recent commercial smoke detector produced by Google [5] can send a notification to smartphones; however, its detection technique is smoke-based and cannot suggest the required fire extinguisher. The proposed work uses a thermal camera for fire detection, which is quicker, as light travels faster than smoke. In [6], fire is detected in camera images using Convolutional Neural Network (CNN)-based object detection models such as a faster Region-based Convolutional Neural Network (RCNN), Region-based Fully Convolutional Network (R–FCN), Single-Shot Detector (SSD), and You Only Look Once (YOLO) v3. Pretrained models of these four CNN models have been retrained with a custom dataset using transfer learning. Among them, YOLO v3 detected fire most quickly with 83.7% accuracy for that custom dataset. The work in [7] uses two deep learning models—Faster-RCNN Inception V2 and SSD MobileNet V2—to detect indoor fires in camera images. The models were trained using a custom dataset and the accuracy of the fire detection of these models was 95% and 88%, respectively. In [8], the authors use a reduced-complexity CNN architecture, InceptionV4-OnFire, to detect fires from camera images. A custom dataset was used to train the model and an accuracy of 96% was achieved for that dataset. The work in [9] uses camera images to detect fires using image processing techniques. The proposed method has three stages: fire pixel detection using a color model developed in the CIE L*a*b* color space, moving pixel detection, and analyzing fire-colored moving pixels in consecutive frames. The method was tested on several video sequences and it had a detection accuracy of 99.88%. In [10], image processing is used to detect fires. The authors use the YCbCr color space and fuzzy logic for fire detection. The proposed method was trained using custom fire images and it achieved up to 99% accuracy. Here, works [6–10] use RGB camera images with deep learning and image processing algorithms to detect fire. Although a good detection accuracy was reported for these techniques, it should be noted that these results are for a custom dataset, and the accuracy may differ when implemented in real life with unknown environments. Generalization is challenging for deep learning models and these models perform unexpectedly if conditions such as lighting, viewing angle, orientation, etc. are different from the training set. Hence, the reliability of these models in fire detection may not be sufficiently high, which is crucial for ensuring the safety of lives and property. In the proposed work, fires are detected using a thermal camera, and it can detect object temperature and fire with an accuracy of 100% as long there is a line of sight. Its detection is not affected by lighting conditions or the orientation of the object. Moreover, the works in [6–10] neither recognize the burning object on fire nor notify the emergency responders suggesting the class of fire extinguisher needed. No hardware implementation of the IoT system is presented in those works.

The work in [11] proposed a smart fire alarm system to be used in a kitchen. Here, fire is detected using a thermal camera and the user manually draws a rectangle on the stove area as the region of interest. The proposed system also contains an RGB camera for detecting a person in the kitchen using the YOLOv3-tiny CNN model. If a person is detected and there is a fire in the stove area, then the alarm will not be triggered, as the person can take care of the fire. However, according to this method, if the person's clothes or somewhere other than the stove area catches fire, such as on the shelves, curtains, or carpet, then an alarm will not be triggered. Thus, emergency responders will not be alerted. The proposed method has been implemented in an embedded Industry Personal Computer (IPC) interfaced with an expensive bi-spectrum camera that can capture both thermal and RGB images. This work neither recognizes the burning object on fire nor notifies the emergency responders suggesting the class of fire extinguisher needed.

A comparison of the proposed system with other works is shown in Table 1. Compared with these works, the proposed smart fire detector uses a low-cost thermal camera to detect fires reliably in any room of a building, uses an RGB camera to detect the object on fire using a deep learning method, and notifies the emergency responders suggesting the class of fire extinguisher needed—which is critical information for mitigating the fire. To the author's knowledge at the time of writing this paper, no work is available in the literature that can detect the object that is on fire and automatically suggest the class of fire extinguisher needed, and this research fills this gap. This work also presents an IoT system prototype comprising the fire detector device, central monitoring server software, and smartphone apps for the users and emergency responders.

Table 1. Comparison with other works.

	P. Li et al. [6]	**J. Pincott et al. [7]**	**G. Samarth et al. [8]**	**T. Celik et al. [9]**	**H. Demirel et al. [10]**	**Y. Ma et al. [11]**	**Proposed**
Fire detection method	From images using CNN	From images using CNN	From images using CNN	Image processing	Image processing with fuzzy logic	Thermal camera	Thermal camera
Fire detection accuracy	83.7%	95%	96%	99.88%	99%	100%	100%
The object on fire detection	No	No	No	No	No	No	Yes, using ssd-inception-v2
Embedded system implementation	No	No	No	No	No	Yes	Yes
Record fire scene video with timestamp	No	No	No	No	No	Yes	Yes
Plot on map	No	No	No	No	No	No	Yes
User and device configuration	No	No	No	No	No	No	Yes
Database implementation	No	No	No	No	No	No	Yes
Smartphone notification	No	No	No	No	No	Yes	Yes
Extinguisher recommendation	No	No	No	No	No	No	Yes

3. Materials and Methods

3.1. Detection Algorithms

The experimental setup as shown in Figure 2 is used to develop the object and fire detection, and extinguisher class recommendation algorithm. An RGB camera and a thermal camera are interfaced with an NVIDIA Jetson Nano developer kit [12]. Realistic small-sized furniture for dolls [13] was placed under the cameras during experiments. Small fires were generated using a barbeque lighter in front of the furniture. A brief description of the detection of the objects in the room, the fire, the burning object, and the extinguisher class recommendation is described below.

3.1.1. Object Detection

From RGB camera images, object detection within a room is performed to identify the burning object's name and subsequently recommend the appropriate class of fire extinguisher. The deep learning model, SSD-Inception-V2, which combines the Single Shot MultiBox Detector (SSD) architecture [14] with the Inception neural network [15], is used for object detection. This object detector operates by employing a multiscale feature extraction process, extracting feature maps from various convolutional layers of the Inception network. These feature maps are then used to predict bounding boxes and class scores for objects at different spatial resolutions, allowing for the detection of objects of various sizes within a single pass through the network. Additionally, SSD-Inception introduces auxiliary convolutional layers to further enhance the feature representation. The model utilizes anchor boxes to propose potential object locations and refines these predictions using a combination of localization and classification tasks. By integrating Inception's advanced feature extraction capabilities with SSD's real-time, single-pass detection approach, SSD-Inception achieves a balance between accuracy and speed, making it well suited for real-time object detection tasks. The model is trained using the Common Objects in Context (COCO) dataset [16,17], with a total of 91 classes of objects such as chair, couch, bed, dining table, window, desk, door, TV, oven, refrigerator, blender, book, clock, etc. [18].

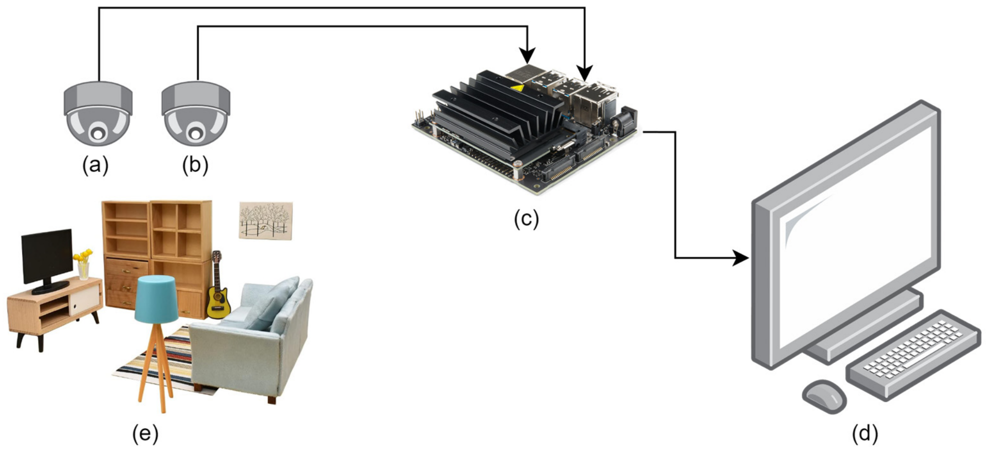


Figure 2. Experimental setup: an RGB camera (**a**) and a thermal camera (**b**) are interfaced with an NVIDIA Jetson Nano developer kit (**c**). A monitor, wireless keyboard, and wireless mouse (**d**) were connected to the Jenson Nano. Realistic small-sized furniture (**e**) was placed under the camera during experiments.

3.1.2. Fire Detection

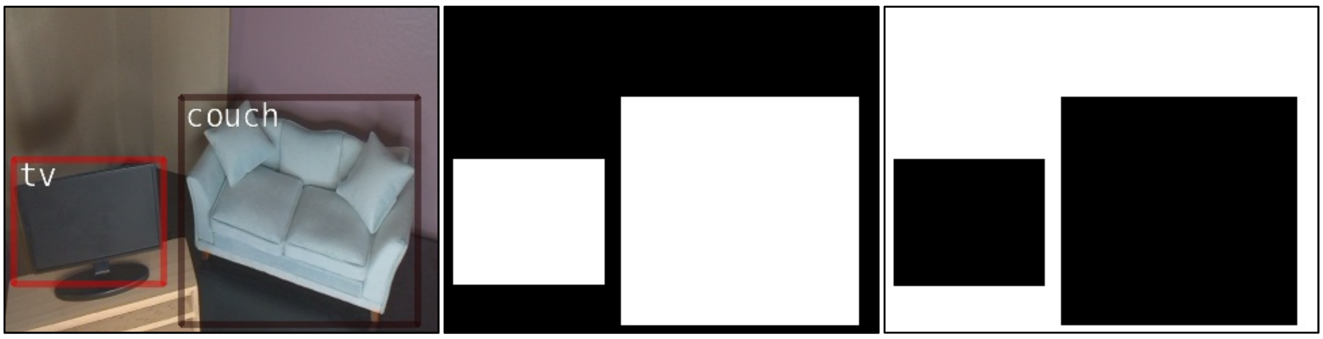
In the proposed system, fires are detected using a thermal camera. A thermal camera is designed to capture and quantify this emitted infrared energy from objects. Every object emits infrared energy, referred to as its heat signature. A thermal camera captures and then translates this infrared data into an electronic image, revealing the apparent surface temperature of the objects. The thermal camera is not affected by the lighting conditions of the environment; thus, it can measure temperature in both day and night conditions. Fires are detected from the thermal image using thresholding. If a pixel temperature is higher than the threshold, then that pixel is detected as fire.

3.1.3. Burning Object Detection

The name of the object on fire is detected by calculating the overlap between the area inside the boundary box, referred as β, of each detected object from the RGB image, and the area of each fire contour, referred as φ, from the thermal image, as shown in (1). This means that if the intersection of sets β and φ is not an empty set, then ʄ to β yields the name of the object on fire, η, where ʄ maps the boundary box to its assigned object name.

$$(\beta \cap \varphi \neq \emptyset) \Longrightarrow \eta = ʄ(\beta) \tag{1}$$

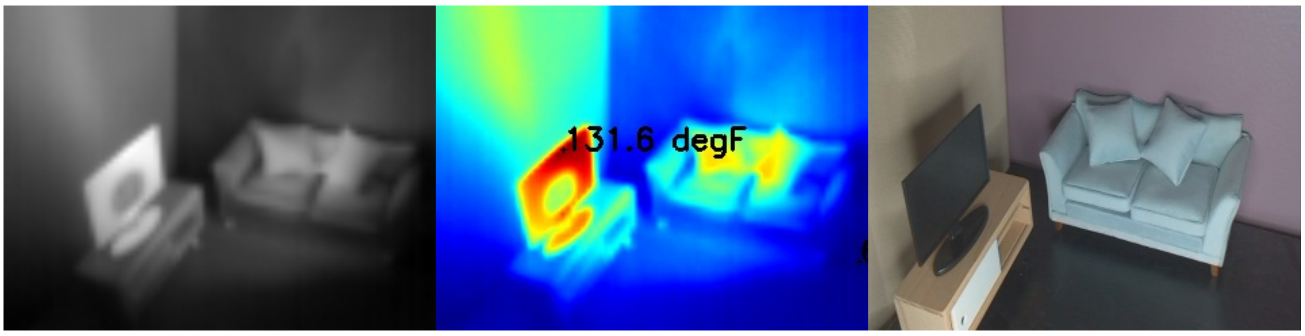
The RGB camera captures the optical image and then the boundary box of each common object is detected according to the discussion in Section 3.1.1. In Figure 3a, two detected objects, a TV and a couch, with boundary boxes are shown. Then, mask images are generated for each object where the pixels inside the boundary box are set to 1 and other pixels are set to 0. An OR-ed (i.e., union) image of all the object masks is shown in Figure 3b where the black pixel indicates 0 and white pixels indicate 1. The inverse of the mask in Figure 3b is calculated, where the white area represents the mask of the unknown objects, as shown in Figure 3c.



(**a**) (**b**) (**c**)

Figure 3. (**a**) Detected objects—tv and couch—with boundary boxes from RGB camera image; (**b**) OR-ed (i.e., union) image of all the object masks; (**c**) mask of the unknown objects.

The thermal camera, which is placed beside the RGB camera, captures the surface temperature of the objects. A grayscale thermal image and a pseudo-colored thermal image using a jet colormap are shown in Figure 4a,b, respectively. The image captured by the RGB camera of the same scene is shown in Figure 4c. If we compare the images in Figure 4a,c, we see that the corresponding objects have different sizes and aspect ratios. The reason for this mismatch is due to the difference in the physical location of the two cameras, their different focus lengths, and the different aspect ratios of the captured images (RGB camera 1.777 and thermal camera 1.333). Thus, overlapping these two images will not give accurate results.



(**a**) (**b**) (**c**)

Figure 4. (**a**) Grayscale image captured by the thermal camera; (**b**) pseudo-colored thermal image using a jet colormap with maximum temperature point labeled; (**c**) image captured by the RGB camera of the same scene.

To solve this problem, a transformation known as homography [19] is used. Homography takes at least four points in the source image and their corresponding points in the target image as inputs and then calculates a 3 × 3 transformation H matrix. Once the H matrix between the two images is known, a wrapped image can be generated as an output by multiplying the source image by the H matrix. The generated wrapped image will have similar object dimensions to the target image. In this project, the thermal image is considered the source image, and the RGB image is considered the target image, as shown in Figure 5a,b. Nine corresponding point pairs are manually selected. Then, using OpenCV [20,21], the H matrix is calculated as shown in (2), and a wrapped image is generated as shown in Figure 5c. The objects in Figure 5c, the image taken by the thermal camera, have similar sizes and aspect ratios when compared with the objects in Figure 5b, the image taken by the RGB camera. Thus, these images can be overlapped without too much error in corresponding points.

$$H = \begin{vmatrix} 1.13722850e+00 & 9.47917113e-02 & -5.05224869e+01 \\ -2.39533853e-02 & 1.41938468e+00 & -4.80018099e+01 \\ 1.88423621e-04 & 3.61817207e-04 & 1.00000000e+00 \end{vmatrix} \tag{2}$$

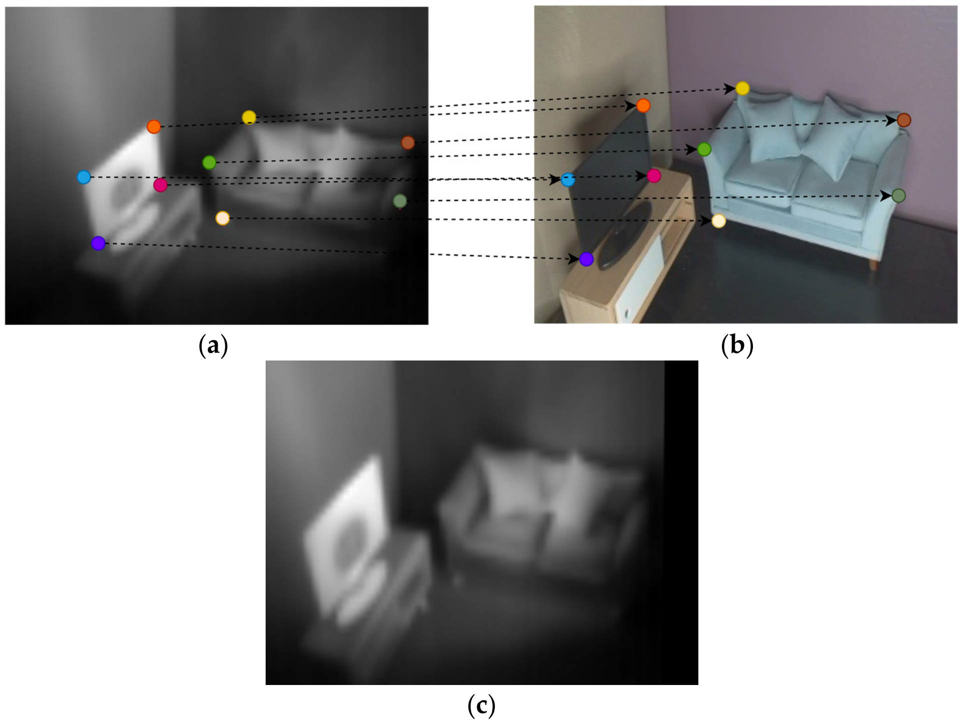


Figure 5. Thermal image (**a**) and RGB image (**b**) with corresponding points; (**c**) generated wrapped thermal image using homography having similar dimensions to the objects in (**b**).

The thermal image is converted into a binary mask image by thresholding to detect fire—as discussed in Section 3.1.2. In this image, the pixels of fire will be 1, and the pixels without fire will be 0. Then, this image is multiplied by the H matrix to get the wrapped image that will correspond to the image taken by the RGB camera. Then, using OpenCV, the contours for each fire segment are calculated. Figure 6a shows the image from the RGB camera, where fire is placed in front of the couch. The thermal camera image with pseudo-coloring is shown in Figure 6b and the mask of the fire after applying homography is shown in Figure 6c.

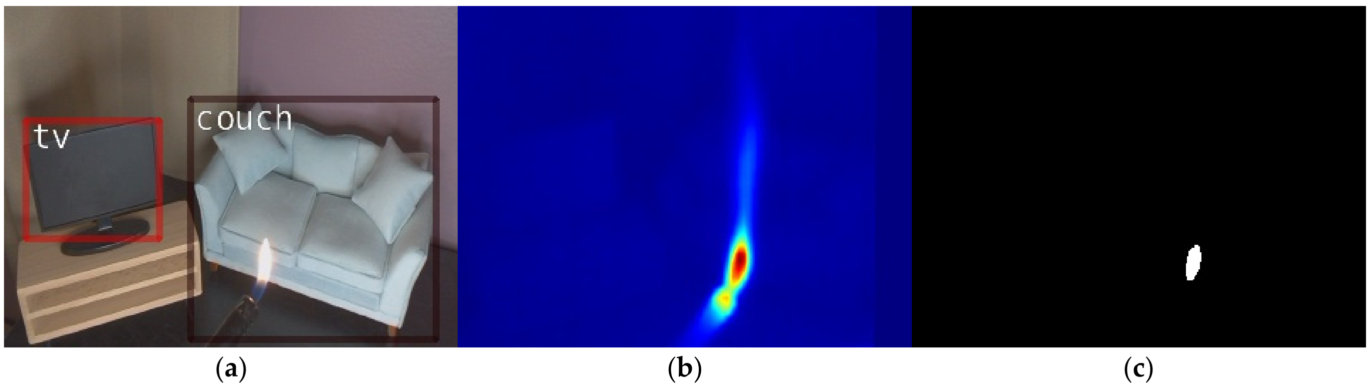


Figure 6. (**a**) RGB camera image showing detected objects' boundary boxes and fire; (**b**) pseudo-colored thermal image; (**c**) mask of the fire after applying homography.

To find the objects on fire, the intersection between the fire masks and each detected object's mask is calculated. If there is an overlap between the fire mask, as shown in Figure 6c, and the object's mask, as shown in Figure 3b, then that object is considered on fire. If the fire mask intersects with the unknown object's mask, as shown in Figure 3c, then the unknown object is considered on fire.

3.1.4. Extinguisher Recommendation

Once the name of the burning object is found, the proper extinguisher class is recommended based on the material and nature of the object. For instance, fires on TVs and refrigerators are electrical fires and will need class C. Fires on beds and couches are on wood and cloth and will need class A [3]. A lookup table is used to obtain the extinguisher class from the object name. Each of the 91 classes of objects [18] in the SSD-Inception-V2 model is assigned to an extinguisher class and they are stored in a database table.

Some objects are as hot as fire, such as an oven, and some objects are on fire for legitimate reasons, such as candles, furnaces, and fireplaces. These objects are referred to as *exception* objects, ε. Neither alerts are generated nor extinguisher classes are recommended when the exception objects are on fire. The recommended extinguisher, referred as $\mathbb{C}$, is calculated using (3). This indicates that set ε is subtracted from η, and then the function ψ is applied to the result, yielding a mapping to $\mathbb{C}$. Here, the function ψ maps the object name to its assigned fire extinguisher class.

$$\mathbb{C} = \psi(\eta - \varepsilon) \tag{3}$$

3.2. Architecture of the Prototype System

The proposed smart fire detection system, as illustrated in Figure 1, is developed, which comprises a fire detector device, a central server, and smartphone apps for users and emergency responders. Users position the device within a room for optimal camera visibility and then employ the smartphone app to configure the device's Wi-Fi settings and update relevant information on the central server. Emergency responders also utilize a dedicated smartphone app to update their details. Once configured, both users and emergency responders can receive real-time smartphone notifications from anywhere in the world, as long as they have an Internet connection, triggered by the fire detector device. Below, we provide a concise overview of the system's various modules.

3.2.1. Smart Fire Detector Device

The smart fire detector device captures both thermal and RGB images of its surroundings and effectively detects fires, including the object on fire. Upon detecting a fire, it promptly transmits the relevant data to the central server via the Internet and saves the video footage of the fire within its local SD card. Configuration of the device is facilitated using a smartphone app. Below, we provide a brief overview of the device's hardware and firmware.

Hardware

Figure 7 illustrates the hardware unit block diagram of the fire detector device, featuring the NVIDIA® Jetson Nano™ Developer Kit (NVIDIA, Santa Clara, CA, USA) [12] as its central processing unit, renowned for its compact design and energy efficiency. This single-board computer excels in executing neural network models, including tasks like image classification, object detection, and segmentation, among others. The Jetson Nano™ Developer Kit boasts a robust Quad-core ARM A57 microprocessor running at 1.43 GHz, 4 GB of RAM, a 128-core Maxwell graphics processing unit (GPU), a micro SD card slot, USB ports, GPIO, and various integrated hardware peripherals. A thermal camera [22] is connected via a smart I/O module [23] and interfaced with the Jetson Nano using a USB. According to the datasheet of the FLIR Lepton® thermal camera (Teledyne FLIR LLC, Wilsonville, OR, USA), it captures accurate, calibrated, and noncontact temperature data in every pixel of each image. An eight-megapixel RGB camera [24] is connected to the Jetson Nano via the Camera Serial Interface (CSI). For wireless connectivity, a Network Interface Card (NIC) [25], supporting both Bluetooth and Wi-Fi, is connected to the Jetson Nano's M.2 socket. An LED, serving as a program status indicator known as the heartbeat LED, is interfaced with a GPIO pin on the Jetson Nano. To power the device, a 110 V AC to 5 V 4 A DC adapter is employed, and to maintain the optimal operating temperature, a cooling fan with pulse width modulation (PWM)-based speed control is positioned above the microprocessor.

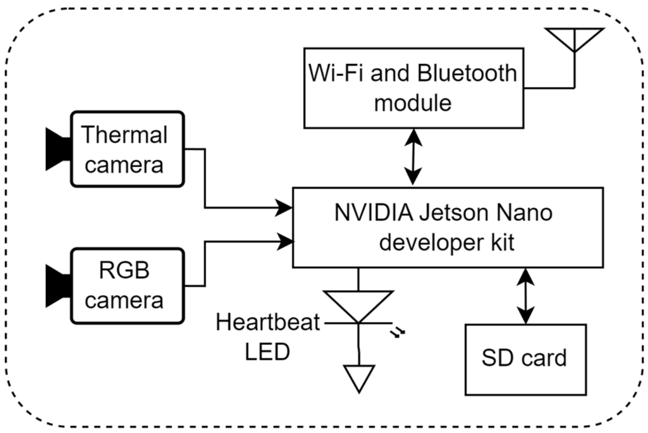


Figure 7. Hardware block diagram for the smart fire detection device.

Firmware

The Jetson Nano board contains a 64 GB SD card, running a customized version of the Ubuntu 18.04 operating system known as Bionic Beaver. The application software is developed using Python, and the system is equipped with all necessary packages, including JetPack 4.6.3. After system startup, three Python programs operate concurrently in separate threads: one for configuration, another for fire detection, and a third for accessing the recorded videos. A brief overview of these programs is provided below.

Wi-Fi Configuration: The objective of this program is to facilitate the configuration of the device's Wi-Fi connection using the user's smartphone. After booting, the program initiates Bluetooth advertisement [26] on the Jetson Nano, making the device detectable to the user's smartphone during Bluetooth scans. In this setup, the Jetson Nano serves as a Bluetooth server, while the smartphone acts as a Bluetooth client. The program subsequently waits for a Bluetooth connection request from the client through a socket [27]. If no connection request is received within 30 min of booting, then it closes the socket, disables Bluetooth advertising, and terminates the program. This timeout mechanism is implemented to reduce unauthorized access. Once the smartphone establishes a connection with the device, Bluetooth advertising is disabled, and the device awaits commands from the smartphone. The smartphone requires knowledge of nearby Wi-Fi service set identifiers (SSIDs) to proceed. When the smartphone sends a command to the device to request the list of nearby SSIDs, the device generates this list using the Linux *nmcli* tool [28] and transmits it to the smartphone. On the smartphone, the user selects the desired Wi-Fi SSID for the device to connect to and inputs the password. Subsequently, the smartphone sends a command, including the SSID and password, to the device, instructing it to connect. Upon receiving the Wi-Fi connection command, the device attempts to connect to the requested SSID and responds with the connected SSID and its local Internet Protocol (IP) address. Once the Wi-Fi configuration is completed, the smartphone sends a "done" command, prompting the device to close the socket connection, re-enable advertising, and await a new Bluetooth connection within the timeout period.

Detecting Objects and Fires: A flowchart of smart fire detection firmware is shown in Figure 8. First, it initializes the hardware and global variables. Here, the RGB camera is configured to capture 320 × 240-pixel color images [29], the thermal camera is configured to send 80 × 60 images in Y16 data format [30], the heartbeat LED pin is initialized as an output pin, and the SSD-Inception-v2 object detection model is loaded in the memory [31]. A list, *listObjectStatus*, is used to keep track of notifications for each object. The length of the list is the total number of objects the deep learning model can detect, which is 91. The list indexes correspond to the class IDs of the detected objects. The list stores the instances of a class *ObjectStatus*, having the properties *isOnFire* and *isNotificationSent*.

During initialization, these properties are set to false. The list, *listObjectExceptionClassID*, is initialized with the class IDs of the exception objects—such as oven, bowl, toaster, hair drier, etc.— where a high temperature is expected and accepted.

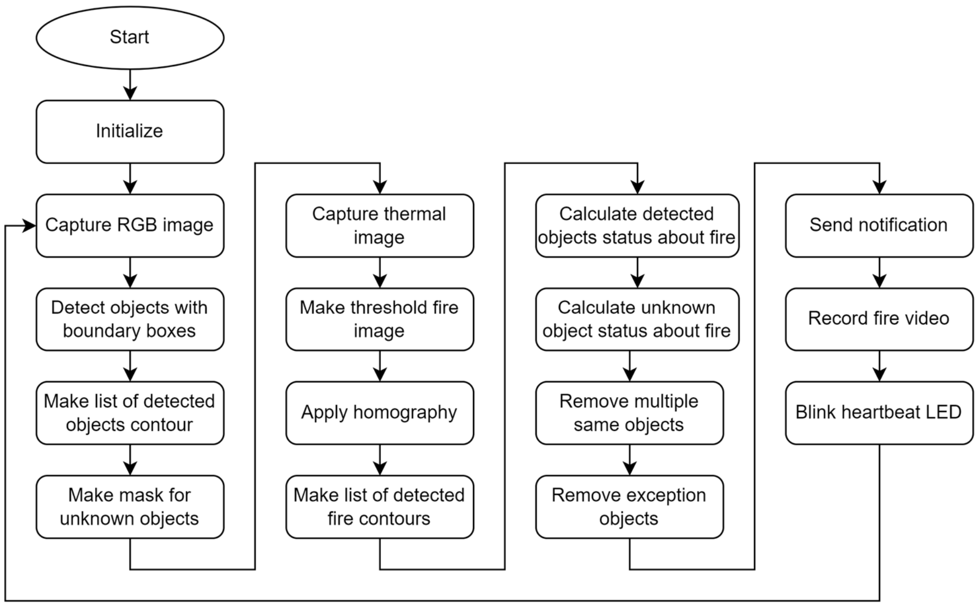

Start
Initialize
Capture RGB image
Detect objects with boundary boxes
Make list of detected objects contour
Make mask for unknown objects
Capture thermal image
Make threshold fire image
Apply homography
Make list of detected fire contours
Calculate detected objects status about fire
Calculate unknown object status about fire
Remove multiple same objects
Remove exception objects
Send notification
Record fire video
Blink heartbeat LED

Figure 8. Firmware flowchart for the smart fire detection implemented on the microcontroller.

The common objects, fire, and burning objects are detected in the firmware according to the discussion in Section 3.1. The RGB image is captured and it is stored in the GPU memory as a CUDA image [32] instead of RAM for faster instantiation of the deep learning model. The CUDA image is then passed to the object detector [33] as input, and it provides the class IDs and the bounding box coordinates as the output. Then, a list of contours [34], *listObjectContour*, is generated from the bounding box coordinates of each detected object. If no object is detected, the list will be empty. Then, a mask is generated, *maskUnknownObj*, that represents the areas of unknown objects, similar to Figure 3c. The mask is generated by first generating a canvas filled with 1 and then drawing each contour on it from the *listObjectContour* filled with 0.

The thermal image is then captured, which is an array of size 80 $\times$ 60. Each element contains 16-bit temperature data in kelvin multiplied by 100. The array is then resized to 320 $\times$ 240 using interpolation. The temperature data, Y, are then converted into degrees Celsius, C, using (4).

$$C = (Y \div 100) - 273.15 \tag{4}$$

After that, a binary image, based on thresholding, is generated from the temperature image, where pixel values higher than the threshold are set to 255 and lower than the threshold are set to 0. The threshold temperature—anything higher than which is considered fire—is set at 65.5 degrees Celsius for this prototype. Then, homography [20,21] is applied to this image, so that the objects in the thermal image have similar coordinates to the RGB image, as discussed in Section 3.1.3. From this image, the contours of the fires are detected [34], and a list of contours, *listFireContour*, is generated from the detected fire contours. If there is more than one fire segment in the scene, then the list will contain the contours of each fire segment. If there is no fire, the list will be empty.

Once *listObjectContour* and *listFireContour* are generated, then each detected object's status, whether it is on fire or not, is determined by calculating the intersects of these contour areas. Each detected object's contour is checked for intersections for each fire contour, and the result of the intersection, *isOnFire*, and the object's class ID, *ClassID*, is appended as an instance of the class, FireStatus, in the list *listFireStatus*. To find whether the areas between two contours intersect or not, first, a canvas of size 320 $\times$ 240 is generated filled with 0. Then, mask1 is generated by drawing the first contour filled with 1 on a copy of the canvas, and mask2 is generated by drawing the second contour filled with 1 on another copy of the canvas. Logical AND is then calculated between mask1 and mask2. If there any results of 1, then the contours intersect.

Note that, the fire could be on objects that are not recognized by the object detector. In this case, the fire is considered to be on an unknown object. The class ID for the unknown object in the object detector is 0 [18]. The intersection between *maskUnknownObj* and each fire contour is calculated in a similar way by creating masks and running logical AND operations. The *ClassID* and *isOnFire* properties of the unknown object are appended to *listFireStatus.*

The object detector may detect multiple instances of objects of the same class, such as three couches in a room. If fire is detected on all three or any of the couches, then sending one notification—mentioning the couch is on fire—is sufficient instead of sending the specific fire status of each couch. However, there will be more than one entry for couch on the list *listFireStatus.* Moreover, if there is more than one fire segment detected, then there will be several entries in *listFireStatus* for the same class ID. To determine which class IDs are on fire in *listFireStatus*, the same class IDs are grouped and then a logical OR operation is used on the *isOnFire* property. Thus, the *listFireStatus* contains unique class IDs along with their fire status stored in *isOnFire*.

The class IDs in the *listFireStatus* that are on the exception objects list, *listObjectExceptionClassID*, are removed from *listFireStatus* so that a notification is not generated where fire is expected and safe.

The *isOnFire* property of the *listObjectStatus,* which is used to keep track of notifications for each 91 objects, is updated from the *listFireStatus.* If an object is no longer on fire and its notification was sent (i.e., *isNotificationSent* is true), it resets the notification status for that object. If an object is on fire and has not had a notification sent yet (i.e., *isNotificationSent* is false), it sends a notification to the server and updates the notification status (i.e., sets *isNotificationSent* to true) to indicate that a notification has been sent. The code ensures that notifications are only sent once for each object on fire to avoid continuous notifications. To send the notification, the program tries to connect with the central server using a socket [35] with a timeout of 5 s and sends a data string containing the serial number of the device, the class ID of the burning object, and the current date and time. The program reads the Bluetooth's media access control (MAC) address [26] of the Jetson Nano and it is used as the serial number of the device. As the device is connected to Wi-Fi, it can get the correct date and time information [36].

If any object in *listFireStatus* is on fire, and recording is not already in progress, it generates a filename based on the current date and time and initializes a video writer with the MJPG codec. It then sets the recording status to true and turns on the heartbeat LED to signal that the recording has started. While recording, the code writes each RGB image frame to the video file in the *rec_fire_video* folder. If no fire is detected while recording is in progress, it releases the video writer, sets the recording status to false, and turns off the LED. The LED continuously blinks when no fire is detected.

Server for Accessing Recorded Videos: For the purpose of accessing and playing the recorded fire videos for post-incident analysis, an HTTP server [37] is implemented within the device, running on port 8000, with the working directory set to the "rec_fire_video" folder where the fire videos are stored. These files can be accessed and played by the user's smartphone utilizing the local IP address of the device and the designated port number. This accessibility is applicable as long as both the smartphone and the device are connected to the same Wi-Fi network. The smartphone obtains the local IP address of the device during the Wi-Fi configuration process.

3.2.2. Software for the Central Server

The central server, created using Visual C# and Microsoft SQL Server [38], incorporates various capabilities, including mapping fire events, generating alerts, sending push notifications to smartphones, and enabling database queries using a graphical user interface (GUI). This server can be hosted on a computer, providing a platform for emergency responders to effectively monitor and respond to critical events.

Structured Query Language (SQL) Database

The software incorporates an SQL database with its table structures, field definitions, and relational connections depicted in Figure 9. In this visual representation, the primary key for each table is denoted by a key symbol positioned to the left of the field name, while lines establish the relationships, linking primary key fields on the left side to the corresponding foreign key fields on the right.

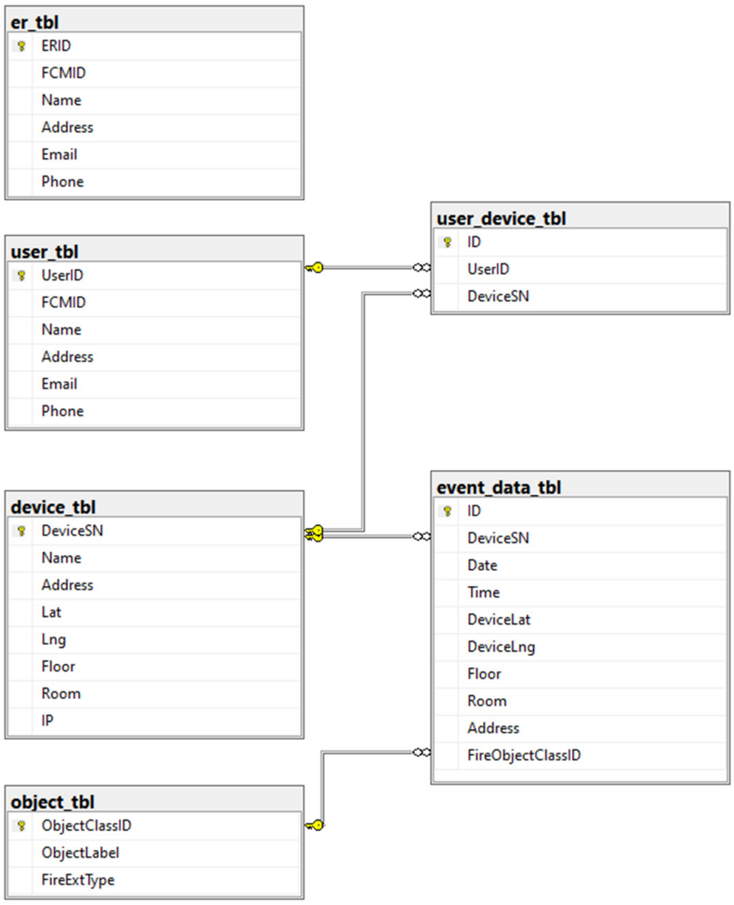


Figure 9. The database tables, their fields, and the associated relationships.

The user_tbl table houses user-related information, including name, address, email, and phone, with the user's smartphone's Android ID [39] serving as the unique UserID. Each smartphone app is treated as a distinct user, and Firebase Cloud Messaging (FCM) registration tokens [40] for sending fire notifications are stored in the "FCMID" field. A unique Android ID and FCM registration token are generated for each user upon app installation. In the device_tbl table, the device details are stored, with the Bluetooth MAC address functioning as the unique device serial number in the DeviceSN field. Location data such as latitude, longitude, address, floor, and room are also stored for a swift emergency response. Users can assign a nickname to the device, recorded in the "Name" field, while the local IP of the device is stored in the "IP" field. The user_device_tbl establishes connections between users and devices, accommodating multiple devices per user and vice versa, creating a many-to-many relationship. The er_tbl table compiles information on emergency responders, including ERID, FCMID, name, address, email, and phone, with ERID storing the Android ID and FCMID holding the FCM registration token, akin to the user table. The event_data_tbl table maintains data on each fire event, encompassing the device's serial number, date, time, location details, and the burning object's class ID, enabling comprehensive event tracking and data queries. Lastly, the object_tbl contains 91 class IDs from the deep learning model, along with their labels and assigned fire extinguisher classes.

Processing Data within a TCP Server

The central server is equipped with a Transmission Control Protocol (TCP) server [41] that actively listens on port 8050. Establishing a connection between the fire detector devices or smartphones and this server necessitates a fixed public IP and an open port. The router's public IP, assigned by the Internet service provider (ISP), is typically static and remains relatively unchanged. It serves as the fixed public IP for this purpose. To facilitate the transmission of incoming data packets from the Internet to our custom TCP server port, the local IP of the server computer is set to a static configuration, and port forwarding [42] is configured within the router. Additionally, the specified port is opened within the firewall settings [43]. The TCP server serves as the receiver for user and device configuration data, emergency responder configuration data from smartphones, and fire notification data from fire detector devices. The first byte of the data indicates whether it pertains to user and device configuration, emergency responder configuration, or fire notification, with each of these data types briefly outlined below.

Configuration data for users and devices: The user and device configuration data strings contain all field values from the user_tbl, the total number of devices, and each field value from the device_tbl for each individual device. Upon arrival at the server, the data undergo parsing, with the extracted information stored in variables and subsequently saved in the respective database tables. If the UserID already exists in the user_tbl, the user's information is edited by updating the corresponding row with the incoming data; otherwise, a new user is added by inserting a new row into the table. SQL queries [44] are employed to execute these operations, connecting to the database. For each device listed in the data, the DeviceID is verified within the device_tbl. If the DeviceID already exists, the device information is updated; otherwise, new device data are appended to the table. Subsequently, the user_device_tbl is modified to allocate the devices to the user. This involves deleting all rows containing the user's UserID and then inserting new rows for each device listed in the data, thereby ensuring the ongoing association of devices with the user, regardless of additions, edits, or removals.

Configuration data for emergency responder: The data string incorporates all field values from the er_tbl. Upon arrival at the server, the data are parsed, stored in variables, and subsequently stored in the corresponding database table. If the ERID already exists within the er_tbl, the information for the emergency responder is updated, while if the ERID is not found in the table, a new entry for the emergency responder is added.

Notification data for fire: Upon the detection of a fire event, these data are transmitted to the server from the fire detector device, comprising the DeviceSN, the class ID of the burning object, and the event date and time. Once received by the server, several actions are initiated: the location information of the device is retrieved from the device_tbl using the DeviceSN; a new entry is inserted into the event_data_tbl to preserve the event data in the database; the event is plotted on a map using a marker [45]; information about the burning object's label and the fire extinguisher class is retrieved from the object_tbl; a fire detection message is displayed; a warning sound is triggered; and Firebase Cloud Messaging (FCM) push notifications [46] are dispatched to both the smartphones of the device's assigned users and all emergency responders. To send these push notifications to each user associated with the device, FCM registration tokens for each user are gathered from the user_device_tbl and user_tbl using a multiple-table query. Each push notification includes essential details such as the DeviceSN, burning object label, fire extinguisher class, device location information, and event date and time.

Fire Event Searching

The software features a graphical user interface (GUI) allowing users to select a date and time range, define a rectangular area on the map, or apply both criteria simultaneously to search for fire events. An SQL query is generated based on the selected criteria, and the resulting data are fetched from the database. Subsequently, the identified fire events from these data are plotted on the map, and relevant location and user information are presented for viewing.

3.2.3. App for Smartphone

Two smartphone applications have been developed specifically for the Android platform: one for regular users and another for emergency responders. These applications feature a settings interface where users and emergency responders can input their information, as depicted in the user_tbl and er_tbl tables shown in Figure 9. It is worth noting that the UserID and ERID, which serve as unique Android identifiers [39] for the smartphones, along with the FCMID, a Firebase Cloud Messaging registration token [40], are all assigned automatically and require no manual input.

The primary distinction between these two applications lies in their functionality. The user application provides options for configuring their devices, while the emergency responder application lacks device configuration options since they are not end users of any devices themselves. The settings window includes a custom list view that displays the user's associated devices. Users can add new devices, edit existing ones, or remove them directly from this interface. The device properties, as illustrated in the device_tbl from Figure 9, can be updated by selecting the respective device.

To simplify the process of inputting device locations, the smartphone can be placed near the device, and the GeoLocation [47] library is used to automatically retrieve GPS coordinates and address information. Additionally, the app incorporates Wi-Fi configuration, as discussed in Section 3.2.1 via a graphical user interface (GUI). Within this interface, users can search for nearby Bluetooth devices and establish connections. Before establishing a connection, pairing between the device and smartphone is required. Once connected, the Bluetooth MAC address is assigned as the DeviceSN, the list of available Wi-Fi SSIDs is retrieved from the device and displayed in the app, and users can select their desired SSID and provide the necessary password, as described in Section 3.2.1.

Upon exiting the settings window, the smartphone establishes a connection with the central server via the Internet as a client. It utilizes a socket to transmit configuration data, which subsequently updates the server's database.

Once the device's Wi-Fi is configured, the smartphone app acquires the local IP address of the device. Utilizing this local IP and the HTTP server port of the device, users can access and play fire videos recorded on the device directly from their smartphones.

The initial screen of both applications features a customized list view displaying fire events. This list includes details such as the device name, serial number, location, burning object name, recommended fire extinguisher class, and date and time of each event. These applications are registered in the Firebase Cloud Messaging (FCM) [48] dashboard to receive push notifications. In the background, a service named FirebaseMessaging operates within the app [49]. When a push notification message is received from FCM, a callback function is triggered. The app appends the message to a list, saves the list to a file, generates a smartphone notification, and updates the list view on the screen. If a user clicks on any item within the list view, it opens Google Maps with the destination set to the location of the device. This feature allows users, as well as emergency responders, to navigate to the incident site quickly.

4. Results

4.1. Simulation Results

The most common evaluation metric for object detection accuracy is mean average precision (mAP). Table 2 shows the inference latency and the mAP for different SSD models [50] trained using the COCO dataset. In Table 2, latency for one 600 × 600 image is reported when running on a desktop computer with an NVIDIA GeForce GTX TITAN X GPU. The detector performance mAP is calculated using a subset of the COCO validation set consisting of 8000 samples. The higher the mAP, the better the accuracy is. These mAPs are bounding box accuracies rounded to the nearest integer. Here, we see that SSD-Inception-v2 has the highest mAP, except for the SSD resnet 50 fpn, which has high latency. Considering both mAP and latency, the SSD-Inception-v2 model is chosen for this project.

Table 2. Latency and mean average precision (mAP) of SSD object detection models.

Model.	Latency (ms)	mAP
SSD mobilenet_v1	30	21
SSD mobilenet v2	31	22
SSD mobilenet v2 quantized	29	22
SSD inception v2	42	24
SSD resnet 50 fpn	76	35

In Figure 10, some of the test images are shown. Here, we see that the object detector successfully detected the objects in most cases and drew the boundary boxes with the percentage confidence level.

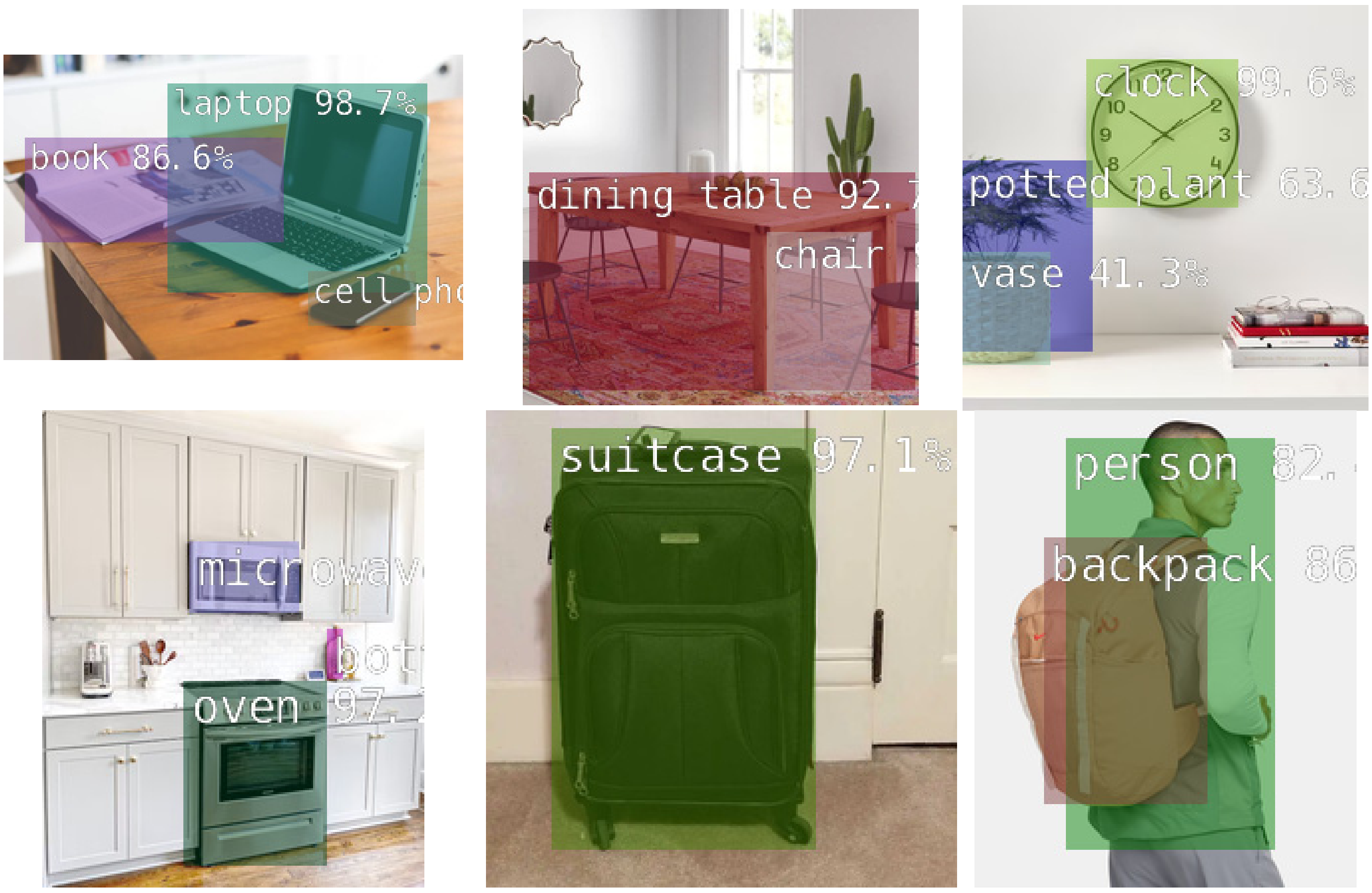


Figure 10. Several examples of object detection on test images. The object label, bounding box, and detection confidence levels are overlaid on the images.

4.2. Prototype Testing Results

A prototype of the proposed smart fire detector device, central server, and smartphone app has been developed and tested successfully. A photograph of the smart fire detector device, labeling different parts, is shown in Figure 11a. A birds-eye-view of the experimental setup with miniature toy furniture is shown in Figure 11b. The device is programmed according to the Firmware section of Section 3.2.1and is configured to run the programs automatically on boot. On the Jetson Nano device, the average time for executing the steps shown in the flowchart in Figure 8 is 65 ms. In this 65 ms, 52 ms is spent by the SSD-Inception-V2 model on detecting common objects. The power consumption of different parts and the entire device is measured using the jetson-stats library [51] and shown in Table 3.

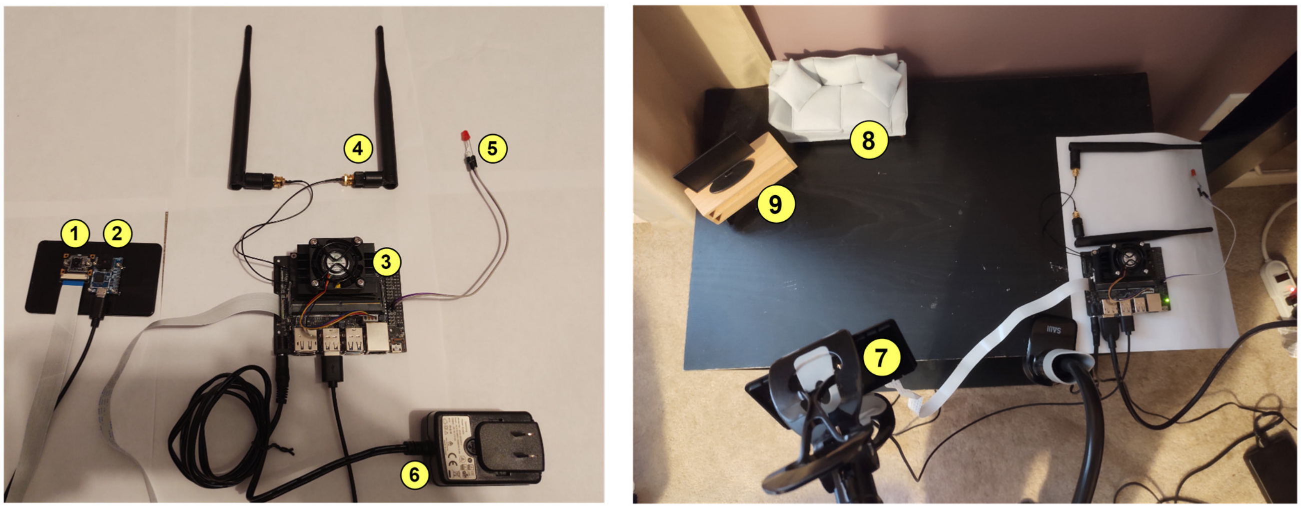
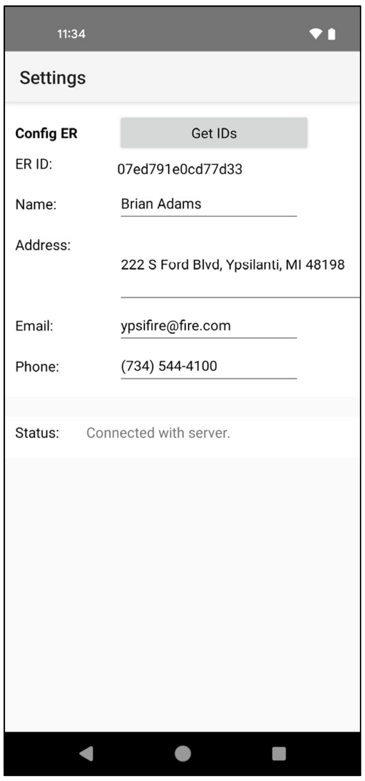


Figure 11. (**a**) Photograph of the smart fire detector device: the RGB camera (**1**) and the thermal camera (**2**) are mounted on a plastic frame side by side. They are interfaced with Jetson Nano developer kit (**3**) using CSI and USB ports namely. Wireless module with antenna (**4**), heartbeat LED (**5**), and DC power adapter (**6**) are interfaced with the Jetson Nano; (**b**) Experimental setup: the plastic frame containing the two cameras are placed above using a camera holder (**7**), so that it can capture the images of the miniature toy furniture such as couch (**8**) and TV (**9**).

Table 3. Power consumption of the smart fire detector device.

Hardware Part	**Power**
Jetson Nano's CPU	1.2 W
Jetson Nano's GPU	3 W
Entire Device	6.6 W

After the device is powered up, the heartbeat LED starts to blink, indicating the program is running and capturing both RGB and thermal images. The central server, as discussed in Section 3.2.2, was running on an Internet-connected computer. The system is then configured using the smartphone app, as discussed in Section 3.2.3. Some screenshots of the smartphone app for configuring the emergency responder, the user, and their device are shown in Figure 12. Using the app, a user and a device are added, and the Wi-Fi of the device is configured successfully. The user and device information were updated as expected in the central server. Using the smartphone app designed for emergency responders, an emergency responder was also added to the system.


11:34

Settings

Config ER Get IDs

ER ID: 07ed791e0cd77d33

Name: Brian Adams

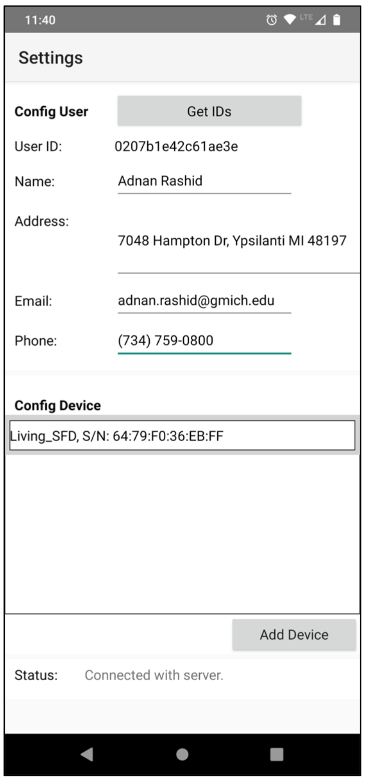
Address: 222 S Ford Blvd, Ypsilanti, MI 48198

Email: ypsifire@fire.com

Phone: (734) 544-4100

Status: Connected with server.

(**a**)


11:40

Settings

Config User Get IDs

User ID: 0207b1e42c61ae3e

Name: Adnan Rashid

Address: 7048 Hampton Dr, Ypsilanti MI 48197

Email: adnan.rashid@gmich.edu

Phone: (734) 759-0800

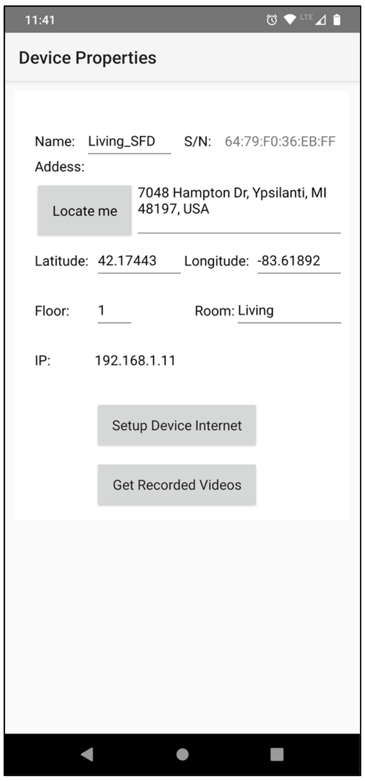
Config Device

Living_SFD, S/N: 64:79:F0:36:EB:FF

Add Device

Status: Connected with server.

(**b**)


11:41

Device Properties

Name: Living_SFD S/N: 64:79:F0:36:EB:FF

Addess:

Locate me 7048 Hampton Dr, Ypsilanti, MI 48197, USA

Latitude: 42.17443 Longitude: -83.61892

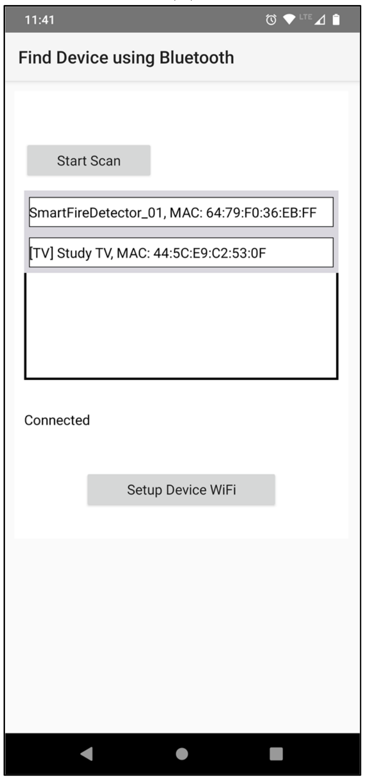
Floor: 1 Room: Living

IP: 192.168.1.11

Setup Device Internet

Get Recorded Videos

(**c**)


11:41

Find Device using Bluetooth

Start Scan

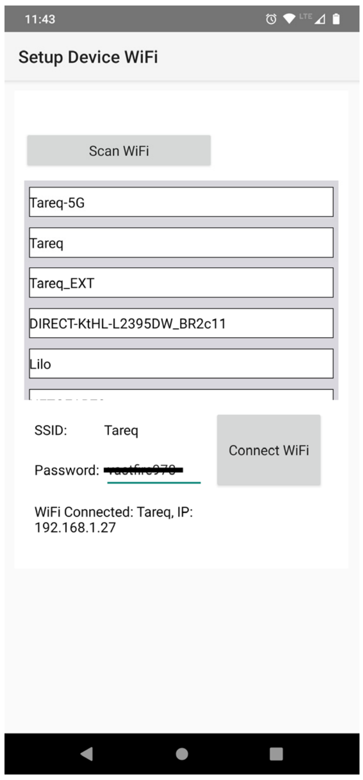
SmartFireDetector_01, MAC: 64:79:F0:36:EB:FF

[TV] Study TV, MAC: 44:5C:E9:C2:53:0F

Connected

Setup Device WiFi

(**d**)


11:43

Setup Device WiFi

Scan WiFi

Tareq-5G

Tareq

Tareq_EXT

DIRECT-KtHL-L2395DW_BR2c11

Lilo

SSID: Tareq

Password: [redacted]

Connect WiFi

WiFi Connected: Tareq, IP: 192.168.1.27

(**e**)

Figure 12. Screenshots from the smartphone apps: (**a**) emergency responder app displaying configuration options; (**b**) user app presenting user configuration and a list of connected devices; (**c**) device property configuration and buttons for Wi-Fi Internet setup and playing recorded fire videos; (**d**) searching for and connecting with devices using Bluetooth; and (**e**) configuring the Wi-Fi SSID for the device.

The smart fire detector system was tested for different cases inside a lab environment by setting small fires in front of the miniature furniture using a barbeque lighter. The objects were not burnt due to safety reasons. The different testing cases are briefly described below:

- Testing fire on a known object: During testing, fires were set in front of the objects that are known to the object detector as listed in [18]. Testing was carried out for the couch and TV as shown in Figure 13a,b, respectively. The device successfully detected the fire and the class of the burning object and notified the central server within a second. Upon receiving the notification data from the device, the central server successfully marked the location of the fire event on the map, displayed the assigned user and device information in the event log, saved the event data in the database, generated warning sounds, and sent smartphone notifications to the assigned user and all the emergency responders. Some screenshots of the central server software and smartphone app after a fire event are shown in Figures 14 and 15. Here, we see that the proposed system correctly identified the objects on fire, such as the couch and TV, and also successfully suggested the fire extinguisher class, such as A for the couch and C for the TV.
- Testing fire on an unknown object: When the fire was on an unknown object as shown in Figure 13c, such as the wooden TV stand, it detected fire but did not recommend fire extinguisher class. The notification was successfully sent to the smartphones about the fire, without recommending a fire extinguisher.
- Testing fire on an exception object: When the fire was on an *exception* object, as shown in Figure 13d, such as on an oven where a high temperature is expected, the system neither considered it fire nor sent any notification.
- Testing with multiple users and devices: The system was also tested with multiple emergency responders, multiple users, and devices having many-to-many relationships, and notifications were sent successfully as expected to several devices.

In the central server, fire events can be successfully searched for using a range of dates and times, a rectangular area on the map, or both. A screenshot of the searching fire event is shown in Figure 16.

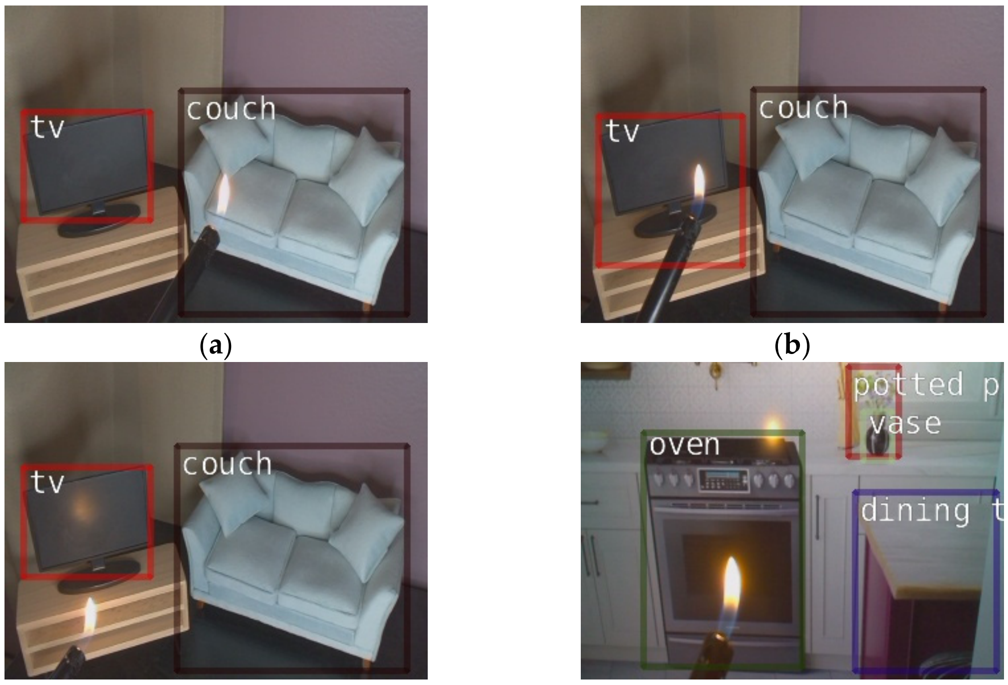
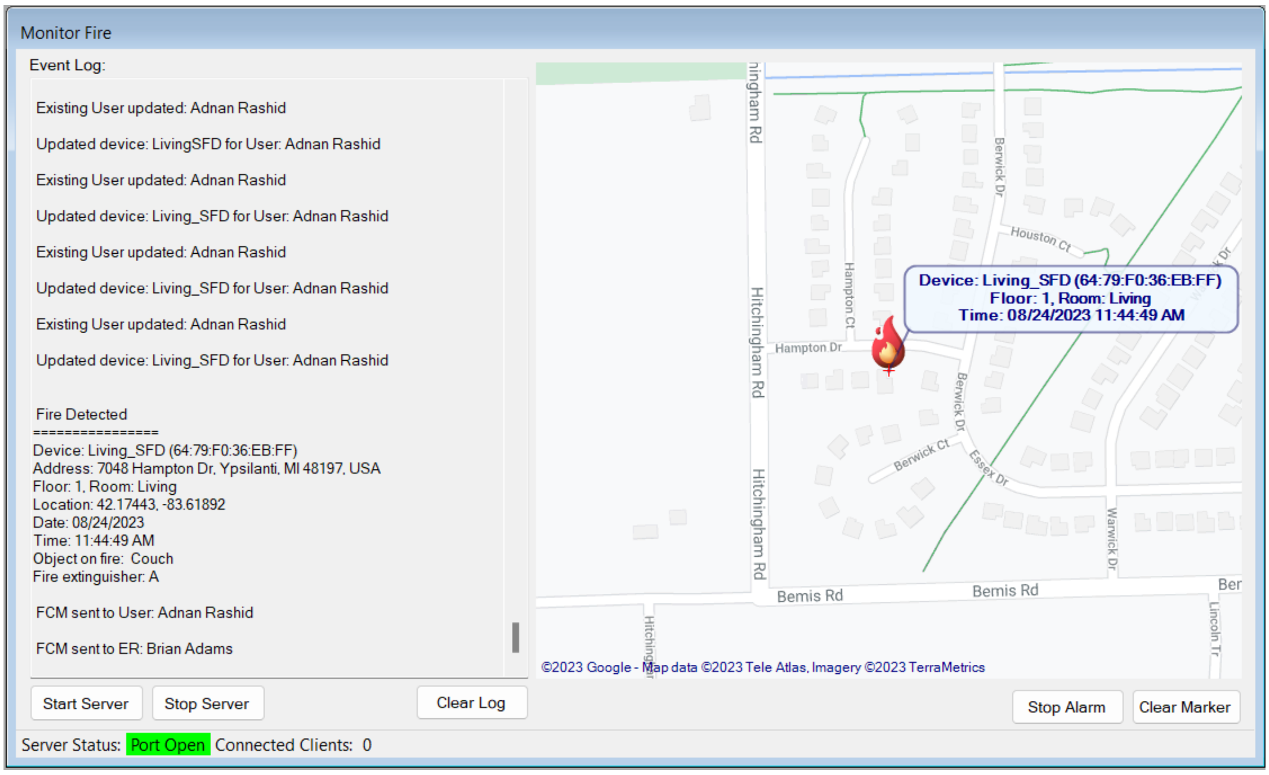


Figure 13. Images captured by the RGB camera when testing with fire: (**a**) fire on couch; (**b**) fire on TV; (**c**) fire on unknown object; (**d**) fire on exception object such as oven.

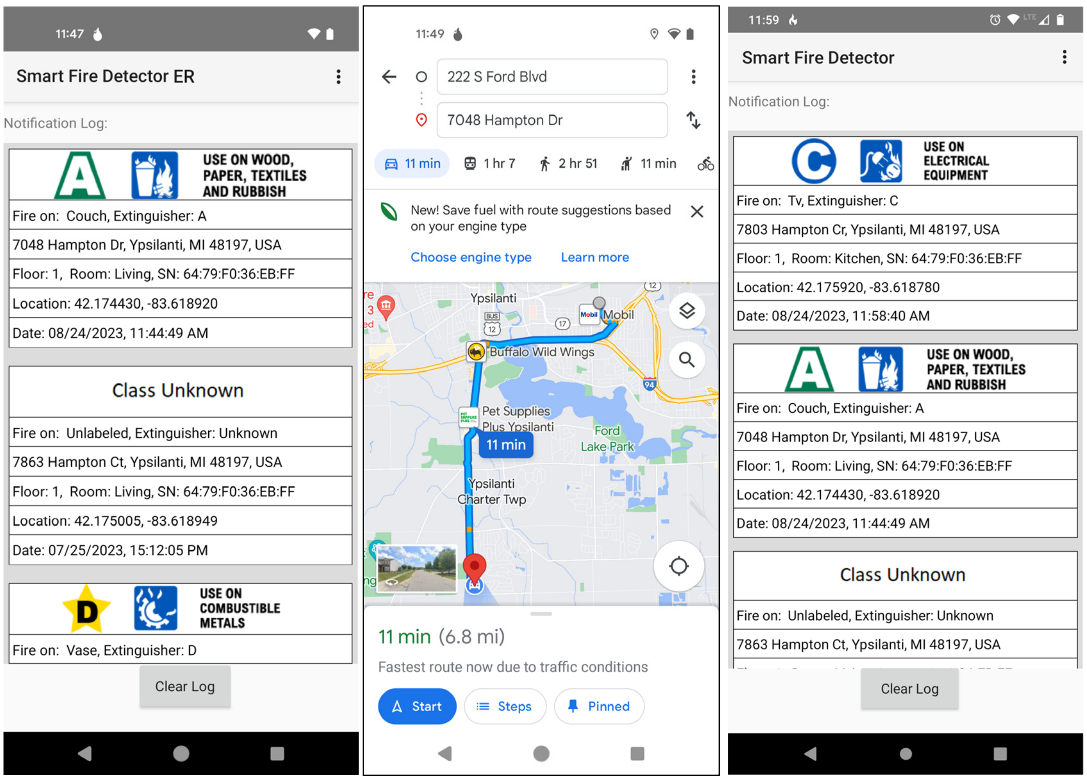


```
Existing User updated: Adnan Rashid
Updated device: LivingSFD for User: Adnan Rashid
Existing User updated: Adnan Rashid
Updated device: Living_SFD for User: Adnan Rashid
Existing User updated: Adnan Rashid
Updated device: Living_SFD for User: Adnan Rashid
Existing User updated: Adnan Rashid
Updated device: Living_SFD for User: Adnan Rashid

Fire Detected
================
Device: Living_SFD (64:79:F0:36:EB:FF)
Address: 7048 Hampton Dr, Ypsilanti, MI 48197, USA
Floor: 1, Room: Living
Location: 42.17443, -83.61892
Date: 08/24/2023
Time: 11:44:49 AM
Object on fire: Couch
Fire extinguisher: A

FCM sent to User: Adnan Rashid

FCM sent to ER: Brian Adams
```

Device: Living_SFD (64:79:F0:36:EB:FF)
Floor: 1, Room: Living
Time: 08/24/2023 11:44:49 AM

Server Status: Port Open Connected Clients: 0

Figure 14. Screenshot of the central server software demonstrating the mapping of a fire event (on the right) and displaying pertinent information, such as the object on fire (in this case, a couch), recommended fire extinguisher, within the event log on the left.



(a)

Notification Log:

USE ON WOOD, PAPER, TEXTILES AND RUBBISH
Fire on: Couch, Extinguisher: A
7048 Hampton Dr, Ypsilanti, MI 48197, USA
Floor: 1, Room: Living, SN: 64:79:F0:36:EB:FF
Location: 42.174430, -83.618920
Date: 08/24/2023, 11:44:49 AM

Class Unknown
Fire on: Unlabeled, Extinguisher: Unknown
7863 Hampton Ct, Ypsilanti, MI 48197, USA
Floor: 1, Room: Living, SN: 64:79:F0:36:EB:FF
Location: 42.175005, -83.618949
Date: 07/25/2023, 15:12:05 PM

USE ON COMBUSTIBLE METALS
Fire on: Vase, Extinguisher: D

(c)

Notification Log:

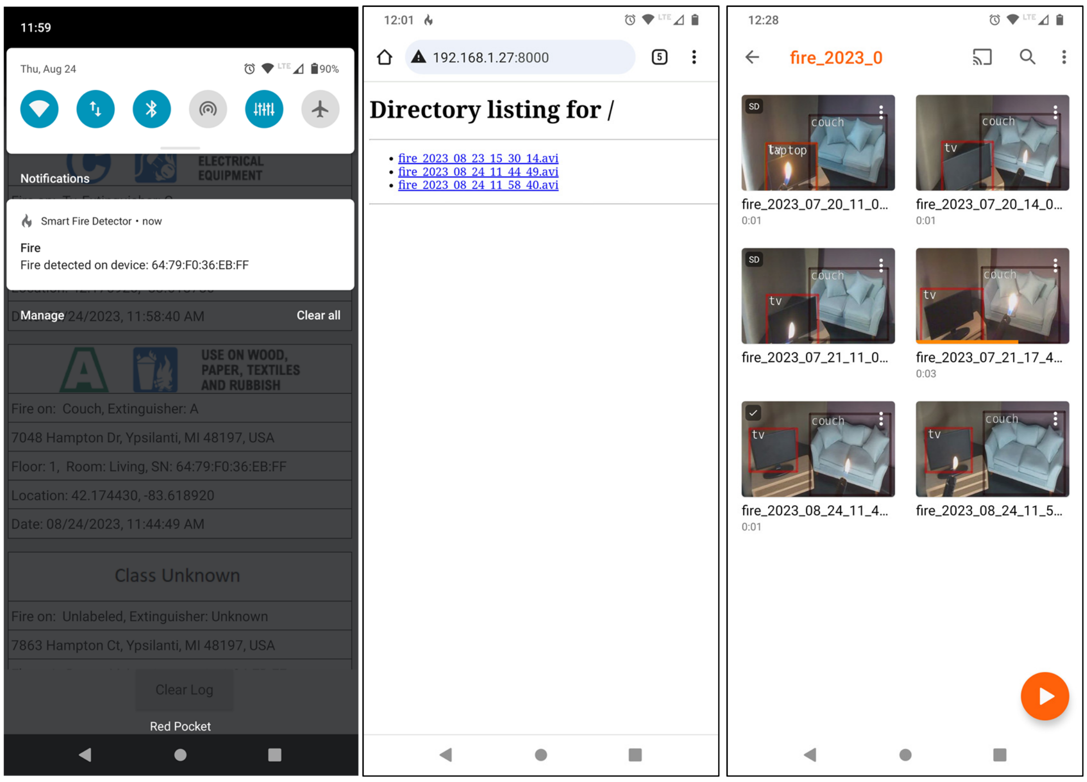
USE ON ELECTRICAL EQUIPMENT
Fire on: Tv, Extinguisher: C
7803 Hampton Cr, Ypsilanti, MI 48197, USA
Floor: 1, Room: Kitchen, SN: 64:79:F0:36:EB:FF
Location: 42.175920, -83.618780
Date: 08/24/2023, 11:58:40 AM

USE ON WOOD, PAPER, TEXTILES AND RUBBISH
Fire on: Couch, Extinguisher: A
7048 Hampton Dr, Ypsilanti, MI 48197, USA
Floor: 1, Room: Living, SN: 64:79:F0:36:EB:FF
Location: 42.174430, -83.618920
Date: 08/24/2023, 11:44:49 AM

Class Unknown
Fire on: Unlabeled, Extinguisher: Unknown
7863 Hampton Ct, Ypsilanti, MI 48197, USA

(a) (b) (c)

Figure 15. *Cont.*


11:59

Thu, Aug 24

Notifications

Smart Fire Detector • now

Fire

Fire detected on device: 64:79:F0:36:EB:FF

Manage 24/2023, 11:58:40 AM Clear all

Fire on: Couch, Extinguisher: A

7048 Hampton Dr, Ypsilanti, MI 48197, USA

Floor: 1, Room: Living, SN: 64:79:F0:36:EB:FF

Location: 42.174430, -83.618920

Date: 08/24/2023, 11:44:49 AM

Class Unknown

Fire on: Unlabeled, Extinguisher: Unknown

7863 Hampton Ct, Ypsilanti, MI 48197, USA

Clear Log

Red Pocket

12:01

192.168.1.27:8000

Directory listing for /

- fire_2023_08_23_15_30_14.avi
- fire_2023_08_24_11_44_49.avi
- fire_2023_08_24_11_58_40.avi

12:28

fire_2023_0

fire_2023_07_20_11_0... 0:01

fire_2023_07_20_14_0... 0:01

fire_2023_07_21_11_0...

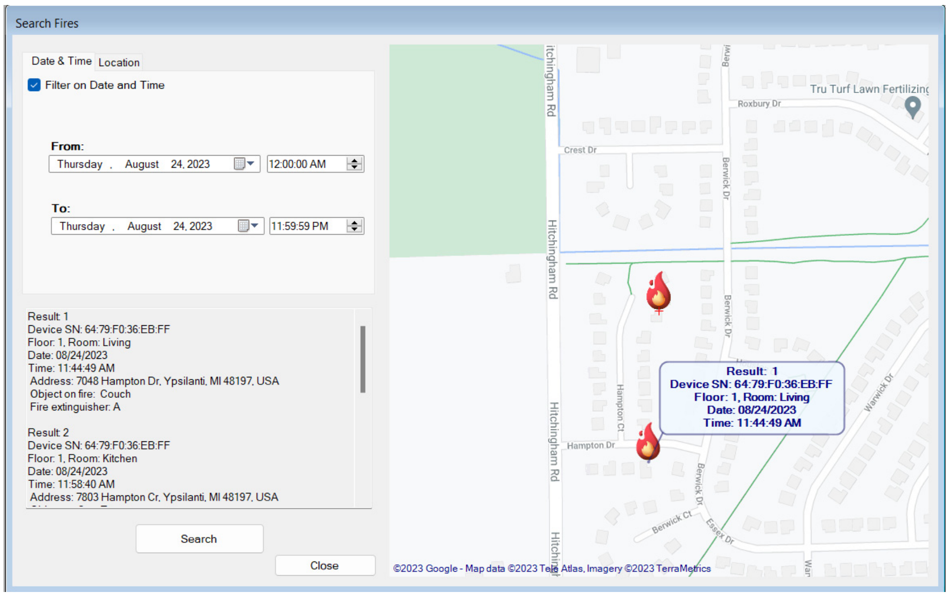
fire_2023_07_21_17_4... 0:03

fire_2023_08_24_11_4... 0:01

fire_2023_08_24_11_5...

(**d**) (**e**) (**f**)

Figure 15. Screenshots of the smartphone apps: (**a**) emergency responder app's list view showing a list of fire events with the name of the object on fire, the recommended extinguisher class, location, date, and time; (**b**) clicking on the list item shows the direction to the fire event location on the map; (**c**) user app's list view showing a list of fire events with the name of the object on fire, the recommended extinguisher class, location, date, and time; (**d**) smartphone notification when fire is detected; (**e**) accessing recorded fire event videos on the device from user's smartphone app; (**f**) thumbnails of the recorded videos on the smartphone app using VLC media player.


Search Fires

Date & Time | Location

☑ Filter on Date and Time

From: Thursday , August 24, 2023 12:00:00 AM

To: Thursday , August 24, 2023 11:59:59 PM

Result 1
Device SN: 64:79:F0:36:EB:FF
Floor: 1, Room: Living
Date: 08/24/2023
Time: 11:44:49 AM
Address: 7048 Hampton Dr, Ypsilanti, MI 48197, USA
Object on fire: Couch
Fire extinguisher: A

Result 2
Device SN: 64:79:F0:36:EB:FF
Floor: 1, Room: Kitchen
Date: 08/24/2023
Time: 11:58:40 AM
Address: 7803 Hampton Cr, Ypsilanti, MI 48197, USA

Result: 1
Device SN: 64:79:F0:36:EB:FF
Floor: 1, Room: Living
Date: 08/24/2023
Time: 11:44:49 AM

Figure 16. Central server software screenshot for fire event searching. Fire events can be searched for using a range of dates and times, selecting a rectangular location on the map, or both. The left side shows the detailed log and the right side shows the plot on the map based on the search result.

5. Discussion and Future Work

It was found during experiments that fire is reliably detected all the time as long as there is a line of sight between the fire and the thermal camera. Fire detection is not affected by the ambient lighting conditions. However, sometimes, object detections by the RGB camera are missed in a few frames. It was found that the performance of object detection was affected by the lighting conditions or the orientation of the object. Fluctuations in detection are a common occurrence in the majority of object detection models, and more basic research on machine learning is needed to solve this.

In this work, the object detection model is trained using 91 common objects [18]. We plan to add more objects to the model by retraining it using transfer learning in the future. Instead of finding the bounding box coordinates of the objects, semantic segmentation could be used, where each pixel is classified. This will improve the accuracy of detecting the object on fire; however, the training and inferencing models for semantic segmentation will require more data and computing power. We plan to investigate this approach in the future.

One of the biggest challenges is to position the cameras for fire detection in houses because this technique requires a direct view of the fire. To cover all objects and to avoid occlusion in the room, multiple cameras may be interfaced in the Jetson Nano device and placed in different corners of the room. Along with the RGB cameras, night vision cameras can be interfaced so that the device can detect objects on fire even in low light conditions. Note that the thermal camera can detect heat and fire irrespective of the lighting condition. In this work, homography was applied to the thermal image, as discussed in Section 3.1.3, by manually selecting corresponding point pairs between the thermal image and RGB image. This manual process can be automated using the Scale-Invariant Feature Transform (SIFT) algorithm [52]. This will calculate the H matrix dynamically, and we plan to implement it in the future.

The highest temperature ever recorded in North America was 56.6 degrees Celsius, and this scorching record was set at Greenland Ranch in Death Valley, California, on 10 July 1913. No other location in the United States has even come close to encountering such extreme heat. The threshold temperature – anything higher than which is considered as fire—is set at 65.5 degrees Celsius for this prototype. If any object experiences long sun exposure and becomes hot, then it will not be considered on fire as long as the temperature of the object is below the 65.5 degrees Celsius threshold level.

One scenario could be that fire destroys the electric supply network before the proposed device detects fire. To avoid such situations, multiple cameras can be interfaced with the device to increase coverage. Moreover, we plan to reduce the power consumption of the proposed device, as shown in Table 3, so that it can run with backup batteries in case of main power failures.

The object and fire detection process continue all the time—even during a fire—as shown in the firmware flowchart in Figure 8. In the firmware, the object detection is inside an infinite loop and it continuously repeats. If an object falls, then it will be detected as long as it is included in the object detection model. Fire generally grows gradually from a small to large size. When the fire on the object is less extensive, the object detector will be able to detect both the object and the fire and notify the server. However, when the fire grows on and engulfs the object, the object may become distorted or fully covered by the fire. In that case, the object will not be recognized. In such a scenario, a notification is already sent when the fire on the object is less extensive.

Instead of using Bluetooth for the Wi-Fi configuration, an alternate method could be that the IoT device creates its own Wi-Fi network (i.e., temporary access point) when it is powered on for the first time. The smartphone of the user can then connect to this access point and transfer Wi-Fi credentials to the device. This approach will eliminate the need for Bluetooth in the IoT device and the smartphone. We plan to implement this in the future.

The fire scenes detected and stored as video files within the device are not transmitted to the central server. These files can exclusively be accessed using the user's smartphone when it is connected to the same Wi-Fi network as the device, ensuring there are no privacy concerns. We plan to automatically blur the human faces in the fire videos to further address privacy concerns in the future. To ensure better security when connecting with the central server and the server in the device for accessing videos, we plan to implement a REST API that can authenticate by providing a username and password within an HTTP header. Though this approach will have additional header data rather than sending fewer raw TCP/IP data, it will help to make the system more secure.

6. Conclusions

This project has produced an innovative IoT-based fire detection device that not only identifies fires but also determines the burning object and the appropriate class of fire extinguisher required, sending location-specific notifications to users and emergency responders on their smartphones within a second. The device utilizes a thermal camera for fire detection and an RGB camera with a deep learning algorithm for object recognition. Notably, it fills a crucial gap in the existing literature by offering an automated system that suggests the class of fire extinguisher needed. The project encompasses a fully functional prototype of the fire detection device, a central server for emergency responders, and successfully tested smartphone apps.

Funding: This research was funded by the Summer Research/Creative Activity (SRA) award of Eastern Michigan University.

Data Availability Statement: The data presented in this study are available only for non-commercial research purposes on request from the corresponding author.

Conflicts of Interest: The author declares no conflict of interest.

References

1. House Fire Statistics. Available online: https://www.thezebra.com/resources/research/house-fire-statistics/ (accessed on 30 August 2023).
2. The Reasons for Smoke Detector False Alarms. Available online: https://www.x-sense.com/blogs/tips/the-common-reasons-for-smoke-detectors-false-alarms (accessed on 30 August 2023).
3. Choosing and Using Fire Extinguishers. Available online: https://www.usfa.fema.gov/prevention/home-fires/prepare-for-fire/fire-extinguishers/ (accessed on 30 August 2023).
4. Different Types of Fire Extinguishers for Each Kind of Fire. Available online: https://weeklysafety.com/blog/fire-extinguisher-types/ (accessed on 30 August 2023).
5. Nest Protect smoke and CO Alarm. Available online: https://store.google.com/product/nest_protect_2nd_gen?hl=en-US (accessed on 1 September 2023).
6. Li, P.; Zhao, W. Image fire detection algorithms based on convolutional neural networks. *Case Stud. Therm. Eng.* **2020**, *19*, 100625. [CrossRef]
7. Pincott, J.; Tien, P.W.; Wei, S.; Calautit, J.K. Indoor fire detection utilizing computer vision-based strategies. *J. Build. Eng.* **2022**, *61*, 105154. [CrossRef]
8. Samarth, G.; Bhowmik, C.A.N.; Breckon, T.P. Experimental Exploration of Compact Convolutional Neural Network Architectures for Non-Temporal Real-Time Fire Detection. In Proceedings of the 2019 18th IEEE International Conference on Machine Learning and Applications (ICMLA), Boca Raton, FL, USA, 16–19 December 2019; pp. 653–658.
9. Celik, T. Fast and Efficient Method for Fire Detection Using Image Processing. *ETRI J.* **2010**, *32*, 881–890. [CrossRef]
10. Çelik, T.; Özkaramanlı, H.; Demirel, H. Fire and smoke detection without sensors: Image processing based approach. In Proceedings of the 2007 15th European Signal Processing Conference, Poznan, Poland, 3–7 September 2007; pp. 1794–1798.
11. Ma, Y.; Feng, X.; Jiao, J.; Peng, Z.; Qian, S.; Xue, H.; Li, H. Smart Fire Alarm System with Person Detection and Thermal Camera. In *ICCS 2020. Lecture Notes in Computer Science*; Springer: Cham, Switzerland, 2020; Volume 12143, pp. 353–366.
12. Jetson Nano Developer Kit. Available online: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (accessed on 6 September 2023).
13. iLAND Dollhouse Furniture and Accessories. Available online: https://www.amazon.com/Dollhouse-Furniture-Accessories-Bookshelves-Decorations/dp/B09QM4WMDP?th=1 (accessed on 6 September 2023).
14. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. SSD: Single Shot MultiBox Detector. In *European Conference on Computer Vision (ECCV), Lecture Notes in Computer Science*; Springer: Cham, Switzerland, 2016; Volume 9905, pp. 21–37.

15. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the Inception Architecture for Computer Vision. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 2818–2826.
16. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision (ECCV), Lecture Notes in Computer Science*; Springer: Cham, Switzerland, 2014; Volume 8693, pp. 740–755.
17. COCO Dataset. Available online: https://cocodataset.org/ (accessed on 5 September 2023).
18. SSD COCO Class Labels. Available online: https://github.com/dusty-nv/jetson-inference/blob/master/data/networks/ssd_coco_labels.txt (accessed on 5 September 2023).
19. Babbar, G.; Bajaj, R. Homography Theories Used for Image Mapping: A Review. In Proceedings of the 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 13–14 October 2022; pp. 1–5.
20. Feature Matching + Homography to Find Objects. Available online: https://docs.opencv.org/3.4/d1/de0/tutorial_py_feature_homography.html (accessed on 7 September 2023).
21. Homography Examples Using OpenCV (Python/C ++). Available online: https://learnopencv.com/homography-examples-using-opencv-python-c/ (accessed on 7 September 2023).
22. FLIR Lepton 2.5—Thermal Imaging Module. Available online: https://www.sparkfun.com/products/16465 (accessed on 13 September 2023).
23. PureThermal 2 FLIR Lepton Smart I/O Module. Available online: https://www.digikey.com/en/products/detail/groupgets-llc/PURETHERMAL-2/9866290 (accessed on 13 September 2023).
24. Waveshare 8MP IMX219-77 Camera Compatible with NVIDIA Jetson Nano Developer Kit. Available online: https://www.amazon.com/IMX219-77-Camera-Developer-Resolution-Megapixels/dp/B07S2QDT4V (accessed on 13 September 2023).
25. Wireless NIC Module for Jetson Nano. Available online: https://www.amazon.com/Wireless-AC8265-Wireless-Developer-Support-Bluetooth/dp/B07V9B5C6M/ (accessed on 13 September 2023).
26. Bluetooth Device Configure. Available online: https://manpages.ubuntu.com/manpages/trusty/man8/hciconfig.8.html (accessed on 13 September 2023).
27. PyBluez. Available online: https://pybluez.readthedocs.io/en/latest/ (accessed on 13 September 2023).
28. Wi-Fi Wrapper Library. Available online: https://pypi.org/project/wifi-wrapper/ (accessed on 13 September 2023).
29. C++/CUDA/Python Multimedia Utilities for NVIDIA Jetson. Available online: https://github.com/dusty-nv/jetson-utils (accessed on 14 September 2023).
30. Boson Video and Image Capture Using OpenCV 16-Bit Y16. Available online: https://flir.custhelp.com/app/answers/detail/a_id/3387/~/boson-video-and-image-capture-using-opencv-16-bit-y16 (accessed on 14 September 2023).
31. Locating Objects with DetectNet. Available online: https://github.com/dusty-nv/jetson-inference/blob/master/docs/detectnet-console-2.md#pre-trained-detection-models-available (accessed on 14 September 2023).
32. Image Manipulation with CUDA. Available online: https://github.com/dusty-nv/jetson-inference/blob/master/docs/aux-image.md (accessed on 14 September 2023).
33. Jetson Inference Library Documentation. Available online: https://rawgit.com/dusty-nv/jetson-inference/master/docs/html/python/jetson.inference.html#detectNet, (accessed on 14 September 2023).
34. OpenCV Contours. Available online: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html (accessed on 14 September 2023).
35. Socket—Low-Level Networking Interface. Available online: https://docs.python.org/3/library/socket.html (accessed on 14 September 2023).
36. Date and Time Library. Available online: https://docs.python.org/3/library/datetime.html (accessed on 14 September 2023).
37. HTTP Server. Available online: https://docs.python.org/3/library/http.server.html (accessed on 14 September 2023).
38. SQL Server 2022 Express. Available online: https://www.microsoft.com/en-us/sql-server/sql-server-downloads (accessed on 18 September 2023).
39. Android Identifiers. Available online: https://developer.android.com/training/articles/user-data-ids (accessed on 18 September 2023).
40. FCM Registration Token. Available online: https://firebase.google.com/docs/cloud-messaging/manage-tokens#ensuring-registration-token-freshness (accessed on 18 September 2023).
41. TCP Server. Available online: https://www.codeproject.com/articles/488668/csharp-tcp-server (accessed on 18 September 2023).
42. How to Port Forward. Available online: https://www.noip.com/support/knowledgebase/general-port-forwarding-guide/ (accessed on 18 September 2023).
43. How Do I Open a Port on Windows Firewall? Available online: https://www.howtogeek.com/394735/how-do-i-open-a-port-on-windows-firewall/ (accessed on 18 September 2023).
44. Thompson, B. C# Database Connection: How to Connect SQL Server. Available online: https://www.guru99.com/c-sharp-access-database.html (accessed on 18 September 2023).
45. GMap.NET—Maps for Windows. Available online: https://github.com/judero01col/GMap.NET (accessed on 18 September 2023).

46. FcmSharp. Available online: https://github.com/bytefish/FcmSharp (accessed on 18 September 2023).
47. GeoLocation. Available online: https://www.b4x.com/android/forum/threads/geolocation.99710/#content (accessed on 18 September 2023).
48. Firebase Cloud Messaging. Available online: https://firebase.google.com/docs/cloud-messaging (accessed on 18 September 2023).
49. FirebaseNotifications—Push messages/Firebase Cloud Messaging (FCM). Available online: https://www.b4x.com/android/forum/threads/b4x-firebase-push-notifications-2023.148715/ (accessed on 18 September 2023).
50. TensorFlow 1 Detection Model Zoo. Available online: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md#coco-trained-models (accessed on 15 November 2023).
51. Jetson-Stats. Available online: https://rnext.it/jetson_stats/ (accessed on 5 October 2023).
52. The SIFT Keypoint Detector. Available online: https://www.cs.ubc.ca/~lowe/keypoints/ (accessed on 11 October 2023).