AI-powered Intrusion Detection System

Documentation

Sathwik Kannam
Michel Jensen

Contents

1 Hardware

The hardware consists of several parts: a 3D-printed box encompassing the camera, light sensor, and IR LED, another 3D-printed box with the alarm system consisting of an Arduino Uno, Bluetooth module, and a siren, a Raspberry Pi Camera Module 2 NoIR, an OKdo Nano C100 (computing platform), a Luminosity sensor, and an IR LED array.

Refer to Section 4 for the hardware overview and the manual for a detailed overview.

1.1 System footprint

1.1.1 IDS case

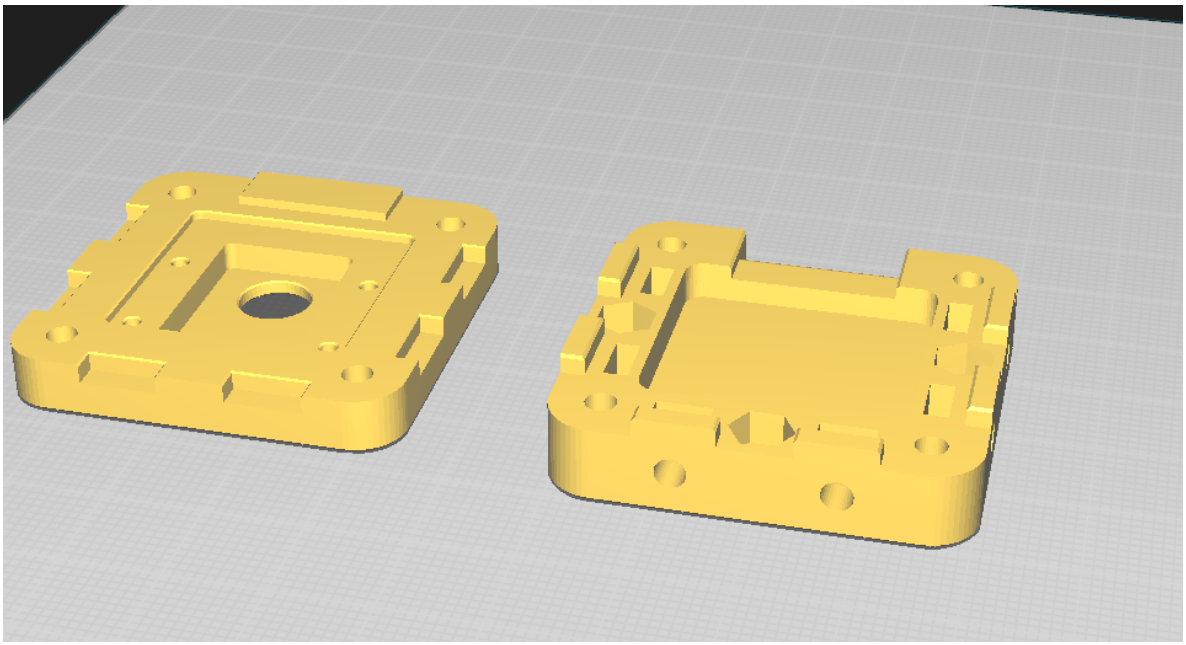

Figure 1: Camera case of Raspberry Pi camera

There are two 3D cases involved in the IDS part of the system, one for the camera and the light sensor and IR LED array. This approach allows the system to be modular as one can place the camera and IR LEDs separately, giving more freedom. The above case is strictly for the camera, where the Raspberry Pi Camera goes in between the two parts requiring no screws. However, there are holes at the edges for M.2 screws if needed. Nevertheless, the camera will be ideally housed in the case without screws.

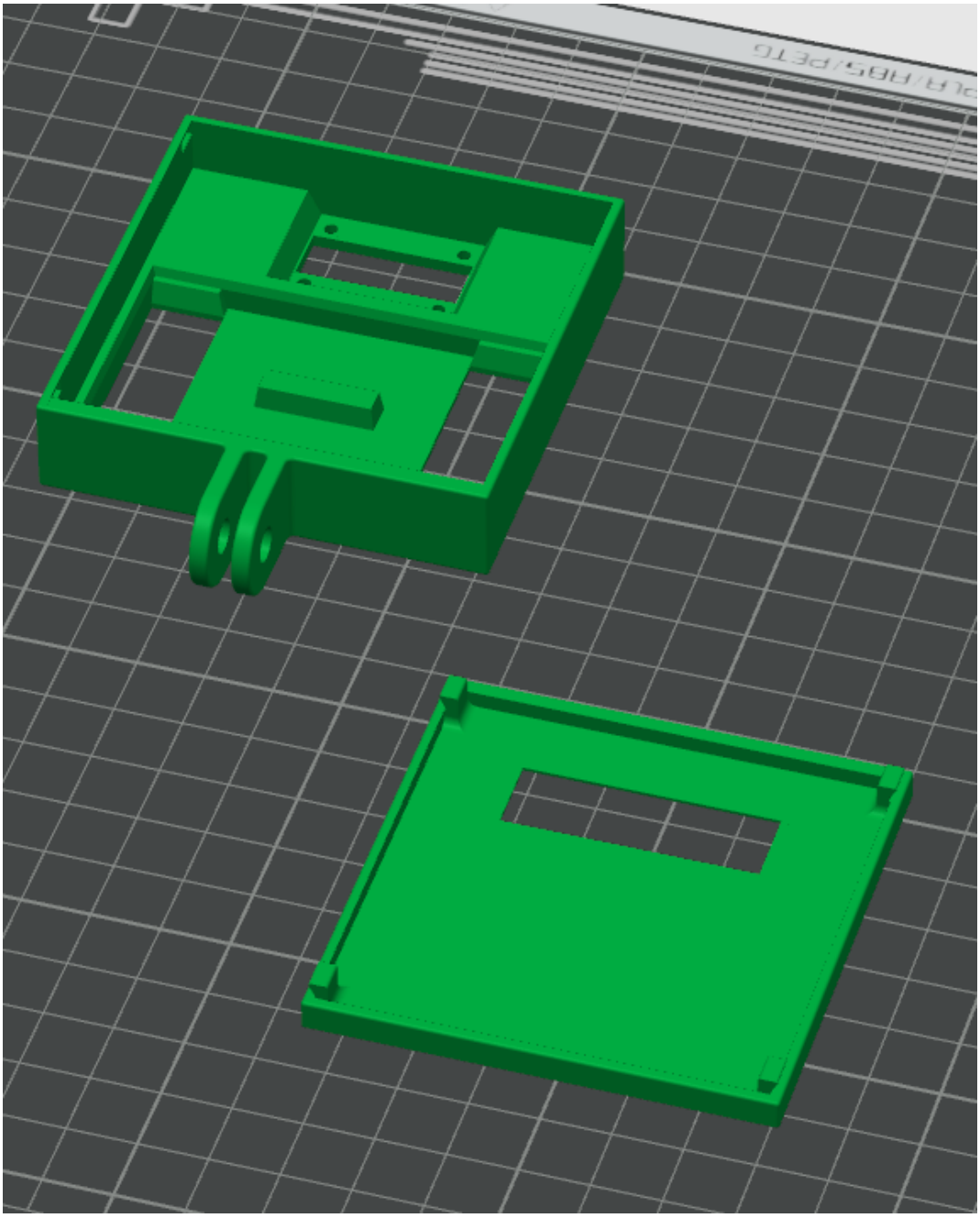

Figure 2: 3D model for Light sensor and IR LED array

The above case houses the light sensor and IR LEDs; an opening on the top for the light sensor only exposes the sensor part. Furthermore, holes are added to the case when screwing the sensor into the case. However, it is not necessary. Underneath the light sensor housing, there are two vertically large rectangular cutouts. These cutouts expose the four IR LEDs, and the cutout is big enough for the light to shine through. On the right of the above figure is a rectangular closure enclosing the case on the left; within this closure is a horizontal cutout, which allows wires and the large transistor to pass through the case to connect them to the Jetson.

In this case, the reason for not implementing the camera's housing in this casing is that the ribbon connector is short. Therefore, as shown previously, a separate case is designed for more user flexibility.

Lastly, the entire box can be mounted to another part, as shown in the following image; this allows

the box to be adjusted horizontally.

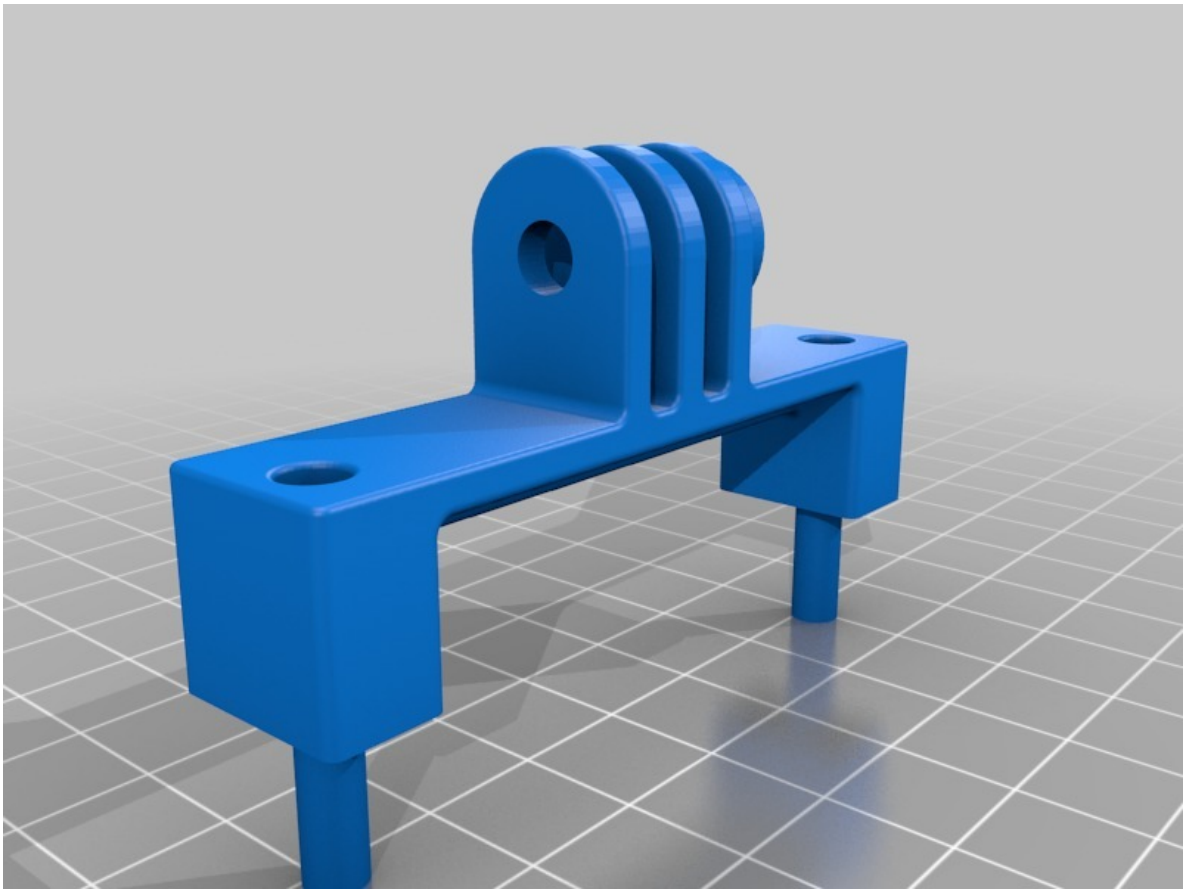

Figure 3: 3D model for mounting the IDS case

The above model is used to mount the casing in Figure 2. They are both attached using a M.3 screw, and one can rotate the Figure 3 case horizontally and lock it in place with a screw.

1.1.2 Alarm case

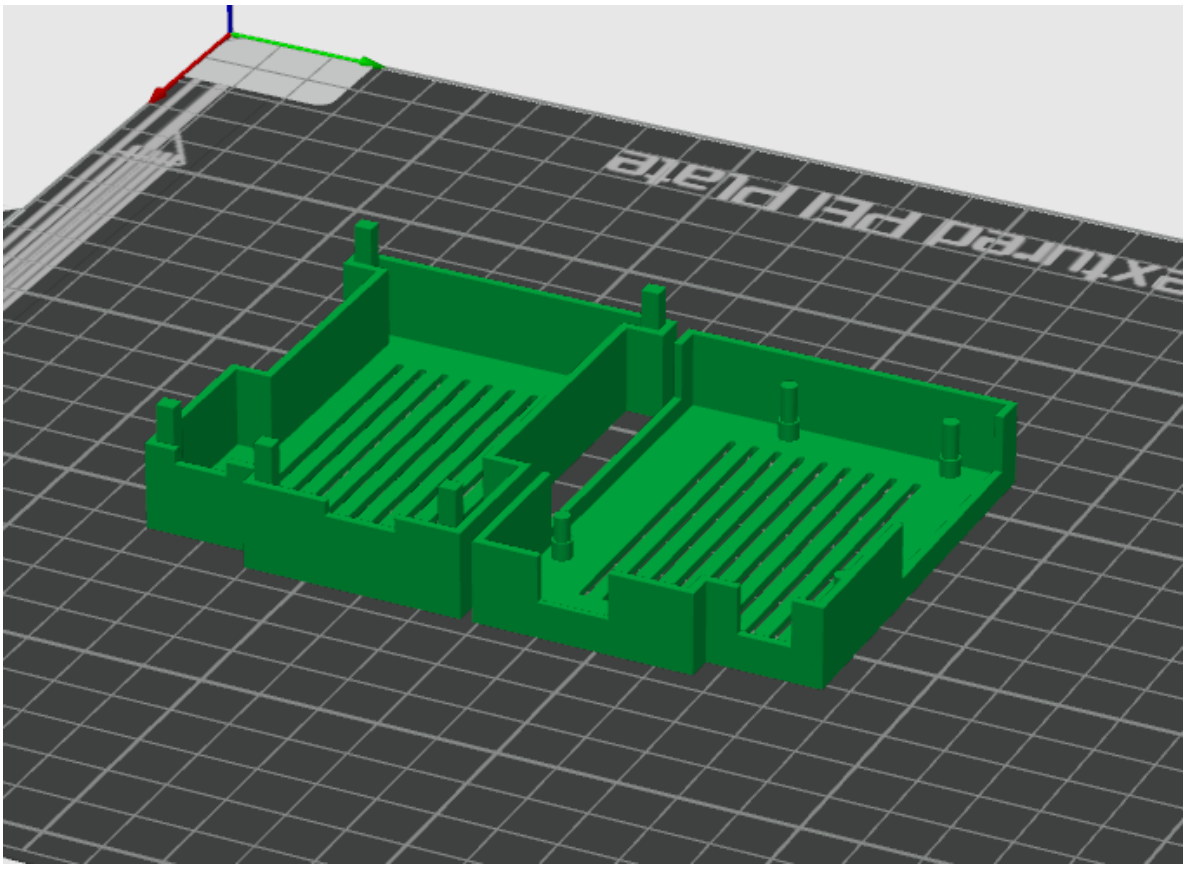

Figure 4: 3D model for alarm system

The alarm case is straightforward in design. It essentially houses an Arduino Uno, an alarm/piezo, and a Bluetooth module. No screws were necessary for this case. Furthermore, the case exposes all the pins and power to the Arduino for easy access.

1.1.3 Entire System

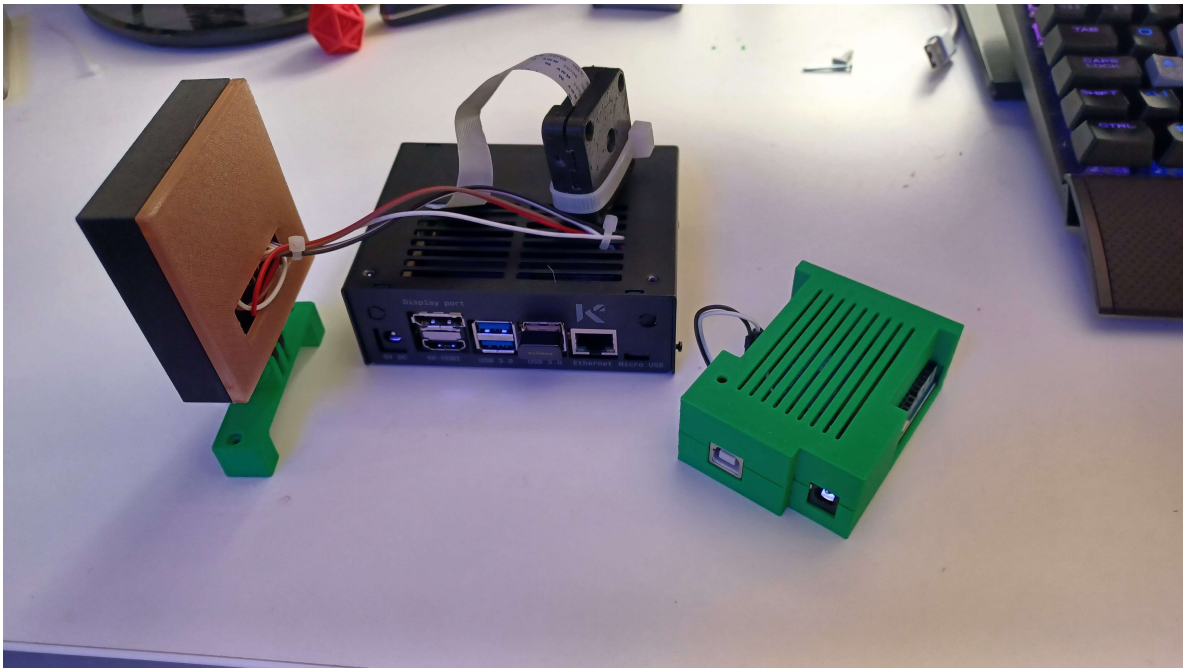

Figure 5: An example of the how the entire system looks like

The mount can be mounted to the Jetson Nano's case itself, as the mount has two holes for screws.

1.2 OKdo Nano C100

The OKdo Nano C100 was used for its integrated GPU and optimization for AI/ML tasks. This is a clone of the original NVIDIA Jetson Nano. However, it has the same components as a Jetson Nano (GPIO, GPU...). The Linux image provided by OKdo was used to flash it (refer to [1] for installing guide).

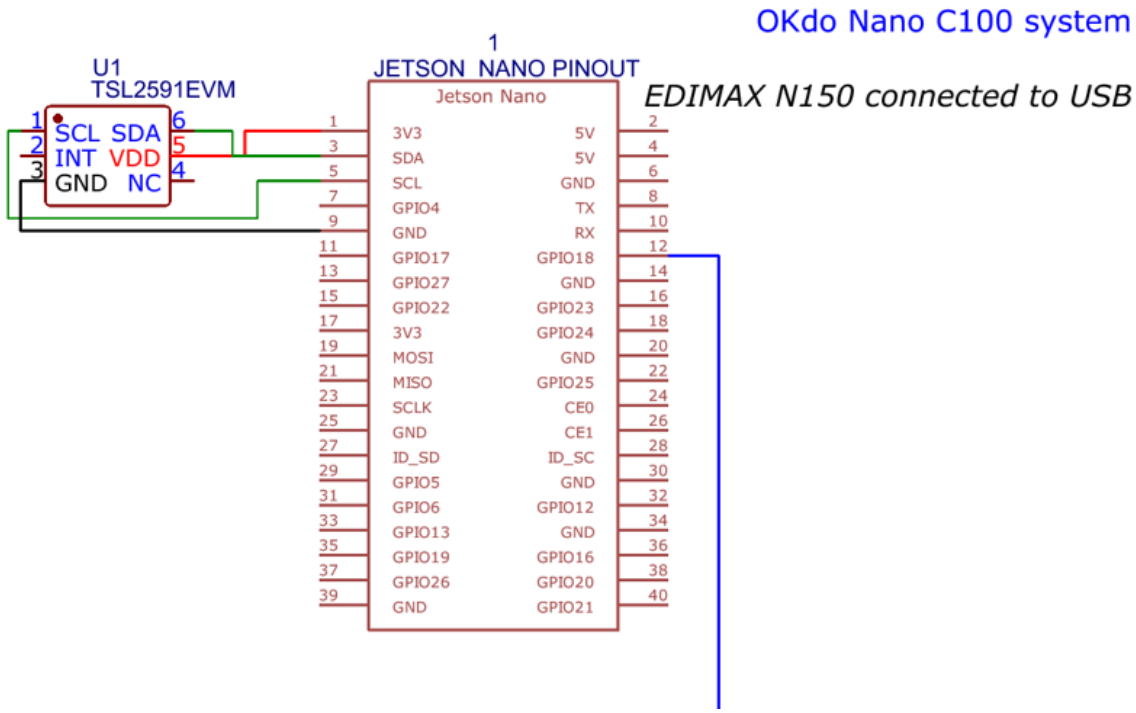

Figure 6: Connections to Jetson

The image above doesn't show the camera but is connected to the Jetson CSI port using the ribbon cable. It is essential to enable I2C communication to interface with the TSL2591 sensor. The datasheet for TSL2591 can be found here [2]. Lastly, the blue connection is for the IR LEDs.

1.3 EDIMAX N150

The OKdo Nano C100 doesn't come with a wireless module but supports an M.2 wireless module or a wireless USB adapter. However, due to the unofficial and unreliable driver support for any M.2 Wireless module, We used EDIMAX N150 interfacing using USB to enable both WiFi 4 and Bluetooth 4 support.

1.4 Camera module

The system uses a Raspberry Pi Camera Module 2 NoIR, which doesn't have an IR filter, so the system can see in the dark with IR LEDs. This camera was selected because Jetson supports Raspberry Pi cameras and has a vast community behind it with drivers and documentation. The same lens and sensor are used without modifications to the camera itself.

1.5 IR LEDs array

The IR LEDS array is an IR LED circuit designed using EasyEDA. It is used to add night vision capabilities to the Raspberry Pi camera. See Section 5 for overview. The array contains 4 IR LEDS (SFH-4714A) with a peak wavelength of 860nm. Further information can be found in [3].

While calculating the number of LEDs required to illuminate at least a 20 x 20 area, we realized that the series configuration would not work since the forward voltage of 7V would be too high for our power supply. The matrix configuration reduces the forward voltage to 3.5V, which is acceptable.

$$\begin{aligned} R\_\text{matrix row} &= \frac{(V_\text{supply} - 2 \cdot Vf)}{If} \\ &= \frac{(5V - 2 \cdot 1.75V)}{1A} \\ &= \frac{(5V - 3.5V)}{1A} \\ &= 1.5\Omega \end{aligned}$$

Therefore, R2 and R3 must be at least 1.5 Ohm for maximum power. R1 can be a standard 220 Ohm resistor. However, we didn't use R2 and R3 as their resistance is extremely small and usually very large for PCBs.

The circuit receives its power from an external 5V power supply (it is rated for 5V and 2A). The positive terminal of the power supply is connected to the anode of LED1 and the positive terminal of the IR LED array. The negative terminal of the power supply is connected to the ground (GND).

The circuit also includes a transistor labeled Q1 (2SA1941-0). The transistor is essential for activating the LEDs. The base of the transistor is connected to Pin D18 to do GPIO control The collector of the transistor is connected to the Jetson's 5V pin. The emitter is connected to IR LEDs.

While heat sinking is generally a good practice for high-power LEDs, the SFH-4714A series from Osram incorporates an integrated heat sink within its package design, eliminating the need for an additional heat sink in our application. Furthermore, we are not running the LEDs at their maximum rated current due to power source limitations. Hence, they don't produce that much heat.

1.6 Luminosity sensor (TSL2591)

We need to measure the luminance of the area the camera's FOV covers. Therefore, we used a TSL2591 luminosity sensor placed above the camera. It can measure to near zero lux and a maximum of 80,000 lux. See the reference manual for this part in [2]. This exact of sensor is necessary as the Jetson is limited to what it can officially support. However, this sensor works flawlessly with the Jetson using I2C.

1.7 Alarm system

An instruction detection system is only applicable when relevant individuals or authorities are informed in cases of intrusions. Therefore, we designed a simple physical alarm with a siren. An ATMega328p is required to run the logic to ring the alarm upon receiving data from the UART. The ATMega328p is connected to the HM-650 Bluetooth module and interfaces with the UART for wireless communication.

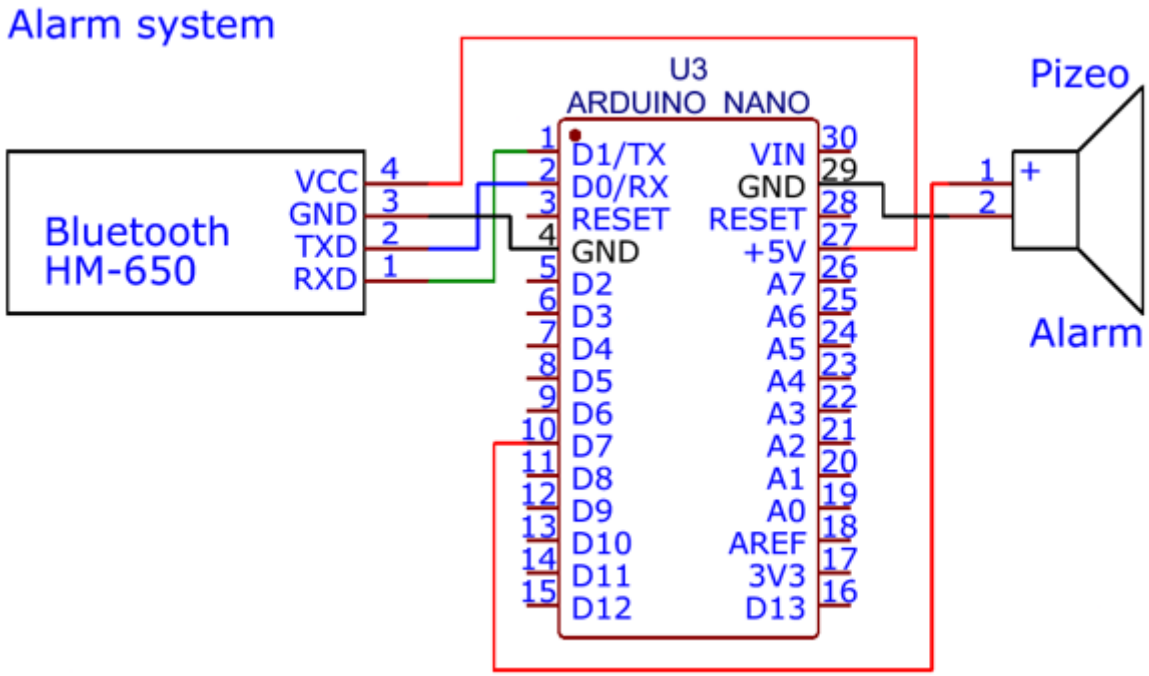

Figure 7: Alarm system connections

A Bluetooth module is required to receive data from the Jetson remotely. The Bluetooth 4.0 model requires BLE communication; the UUID and Bluetooth characteristics are found using a bleak library.

2 Software

The entire software provided in [5]

The software part essentially is a Flask server that streams the footage and the detections from the YOLOv8 model to any clients within the local network. Therefore, one can remotely access the Jetson/Flask without interfering with the hardware. The flask server also implements data collection, enabling clients to collect luminance, weather, and location. However, the user must set these variables, except the luminance, using the UI in the flask server.

This documentation describes precisely what each function does in the `main.py`; however, it and the rest of the files (utility functions and so on) have comments as a docstring that outlines what each function does. Nevertheless, this documentation/section provides an overview and purpose of the whole system. Furthermore, we've implemented a logging system presented to the user, but one can follow the logs in the code to debug and understand the codebase itself.

The entire software requires a `Python` 3.8.0 for PyTorch to work and run YOLOv8 on Jetson's GPU. Therefore, it is essential to install the same dependencies as presented in Section 2.3.

2.1 OpenCV

Gstreamer must be enabled in OpenCV so Jetson Nano can access CSI ports using CUDA. However, the base OpenCV version with OKdo's Linux image and any version installed with pip doesn't allow it. Therefore, we built and installed OpenCV 4.5.0 from the source to enable Gstreamer using CMAKE. This version is installed system-wide and located in the base Python, Python 3.6.9. However, we are using 3.8.0 to run Flask and GPU-accelerated YOLOv8.

Therefore, we handle this situation by running an OpenCV with a GStreamer pipeline script using Python 3.6.9. This file streams the frames using sockets to localhost so that files in the Python 3.8.0 environment can listen to this localhost port to retrieve the frames from the camera. Therefore, the upcoming dependencies table is organized into two tables: one for Python 3.6.9 and Python 3.8.0.

2.2 File Overview

Table 1: File names, and Purpose

Filename	**Purpose**
main.py	The main driver script controls a video streaming application that detects people using a YOLO model. It has login functionality and captures video frames to detect people. It can save the images based on the data collection mode. It also supports sending data to an alarm system and logging information. It has all the flask routes
log.py	This code defines functions to create log messages and store them in a queue. The functions create messages with different styles (info, error, success, warning) and append them to the web_logs deque.

Continued on next page

Table 1 – *Continued from previous page*

Filename	**Purpose**
image.py	This Python file handles saving and managing images. It defines a directory to store the images. It provides a function save_img to save images captured using the OpenCV (cv) library, adding a timestamp to the filename. Another function count_jpgs helps count the number of existing JPG images in the directory. Finally, a function clear_images lets you clear (remove) all JPG images from the directory.
conditions.py	This code defines three categories (Location, Weather, Mode) using enumeration classes. Each category has fixed options (INDOOR, OUTDOOR, etc.).
non_flask.py	This file is the same as main.py, but it only keeps the detection-related functions (core functionality functions) without the flask implementation; this file is used just for pure data collection.
ir_led.py	This file contains functions for controlling (activating and deactivating) the IR LEDs using the GPIO file system.
csicam.py	Initialies a custom version of OpenCV 4.5.0 using Gstreamer pipelines and transmits the frames to localhost, so files such as non_flask.py and main.py in Python 3.8.0 environment can retrieve these frames. This file is retrieved from OKdo's GitHub; for documentation of this file, refer to the GitHub [4].
/alarm/jetson_ble.py	This Python file implements Bluetooth communication. It defines functions to connect to a remote Bluetooth module and send messages. It also includes a time-based logic to regulate how often messages are sent.
/alarm/main.ino	Contains the code for ATMega328p that encompasses ringing the alarm when certain messages are received via UART.
/templates/login.html	This is an HTML file for a login portal. It displays a login form with username and password fields and a submit button.
/templates/video.html	This webpage lets users control a camera and view its live stream. It has buttons to start and stop the stream. The page also has forms allowing users to collect data and the video feed. Users can choose the data collection mode (e.g., luminance, weather) and specify additional information like location and weather conditions. There is also a form to clear data and a link to download collected data in CSV format. The page has the functionality to detect people in the camera feed, but the section that displays the results is currently hidden.
/static/script.js	This code controls a camera stream and displays captured images. It allows users to start and stop the stream, view logs, and download captured images. Users can also set the capture mode (luminance, weather, location) and clear captured data. The code updates the UI based on user selections and fetches data from the server.
/static/styles.css	CSS file for all the HTML files in /templates/ directory
/static/person_clips	Contains images of detections as JPEG
/security/flask_cert.pem	Development Certificate for secure Flask server
/security/flask_key.pem	Development Private key for secure Flask server
/security/user.txt	Development username and password to access the flask server

Continued on next page

Table 1 – *Continued from previous page*

Filename	**Purpose**
/alarm/main.ino	This file contains the logic for the ATMega328p (alarm system). Written in C++ using the Arduino library.
/results/	Contains CSV files from data collection.
requirements.txt	Contains all required packages and versions necessary for running the applications.

2.3 Dependencies

Requires 10.2 version of CUDA

Table 2: Core dependencies used, their version, and purpose for Python 3.6.9 Environment

Dependency	**Version**	**Purpose**
opencv-python	4.5.0 (Compiled from source)	To read frames the camera
pyzmq	26.0.2	To stream the camera frames to localhost
numpy	1.26.0	To convert byte streams to frames/images

Table 3: Core dependencies used, their version, and purpose for Python 3.8.0 Environment

Dependency	**Version**	**Purpose**
ultralytics	8.1.15	To use YOLOv8 model and its necessary tools/metrics
flask	2.2.5	To create the locally hosted webserver
Flask_login	0.6.3	Login framework for adding authentication to the flask server
torch	1.11.0 (Custom version)	To run YOLOv8 on the Jetson's GPU
torchvision	0.12.0 (Custom version)	Required for Torch library
bleak	0.21.1	For Bluetooth (BLE) communication.
python_tsl2591	0.2.0	Library for interfacing with TSL2591 sensor.
pyzmq	26.0.2	For camera stream between Python 3.6.9 to Python 3.8.0 environments

The above is a table of all core dependencies used in our software. However, its packages have their dependencies, which should automatically install when you install them above. However, if they don't, refer to the following table.

Table 4: Core dependencies' package requirements for Python 3.8.0 Environment

Dependency	Version	Description
certifi	2024.2.2	SSL certificate authority
charset-normalizer	3.3.2	Character encoding normalization
colorama	0.4.6	Terminal text color manipulation
contourpy	1.2.0	Contour plotting library
cycler	0.12.1	Cycle control for matplotlib
filelock	3.13.1	Platform-independent file locking
fonttools	4.49.0	Manipulation of font files
fsspec	2024.2.0	File system specification
idna	3.6	Internationalized domain names handling
Jinja2	3.1.3	Template engine for Python
kiwisolver	1.4.5	Efficient solver for linear equations
MarkupSafe	2.1.5	Escaping and string handling
matplotlib	3.8.3	Data visualization library
mpmath	1.3.0	Arbitrary-precision arithmetic
networkx	3.2.1	Graph theory library
numpy	1.26.4	Numerical computing with arrays
packaging	23.2	Package management utilities
pandas	2.2.0	Data manipulation and analysis
pillow	10.2.0	Python Imaging Library (PIL Fork)
psutil	5.9.8	System monitoring and management
py-cpuinfo	9.0.0	CPU information retrieval
pyparsing	3.1.1	Parsing framework for Python
python-dateutil	2.8.2	Date and time utilities
pytz	2024.1	Time zone support
PyYAML	6.0.1	YAML parser and emitter
requests	2.31.0	HTTP library for Python
scipy	1.12.0	Scientific computing library
seaborn	0.13.2	Statistical data visualization
six	1.16.0	Python 2 and 3 compatibility utilities
sympy	1.12	Symbolic mathematics library

Continued on next page

Table 4 – *Continued from previous page*

Dependency	Version	Description
thop	0.1.1.post2209072238	FLOPs and model size estimation
tqdm	4.66.2	Progress bar for loops
typing_extensions	4.9.0	Type hints for Python 3.5+
tzdata	2024.1	Time zone database
urllib3	2.2.1	HTTP client for Python
werkzeug	2.3.8	WSGI utility library
smbus (Note: not smbus 2)	1.1.post2	For I2C communication.

2.4 Server-Sided

The server-side part of the system is the most important; it runs all the logic, streaming, detecting, and collecting data. The server is designed using Flask, a simple and friendly framework for creating locally hosted web servers. The server runs on the Jetson, and anyone within the local network (the same network the Jetson is connected to) can access the server remotely. One can quickly run the `main.py` to run the server; no need to customize the code configuration as we designed the code with this in mind. However, setting up the hardware as presented in the manual is essential.

2.4.1 Configuration

I. `MODEL_PATH`: This variable defines the path to the YOLO model used for object detection. By default, it's set to "models/yolov8n.pt," and the script will download it automatically if not found. You can specify a different model path here.

II. `MINUTES`: This variable determines the duration (in minutes) for which the application will run in location and weather data collection modes.

III. `RUNTIME`: This variable is a conversion of MINUTES to seconds for easier timekeeping within the script.

IV. `location`, `weather`: These variables hold the initial location and weather conditions. They are set to "Indoor" and "Sunny" by default but can be overridden by user selection in the Flask server.

V. `mode`: This variable stores the data collection mode the user selects (Location, Weather, Luminance, or None).

VI. `clear`: This flag indicates whether to clear existing CSV/results files before data collection.

VII. `initial_images_in_person_clips`: This variable keeps track of the number of JPEG images in the output directory at the beginning of a session.

VIII. `INITIALLY_CLEAR_IMAGES`: This flag controls whether to clear images in the output directory at the start of each video streaming session.

IX. `running`: This global flag controls the main loop of the video processing and detection functionality. When True, the application captures video frames, detects people, and saves frames containing people. When False, the loop stops.

X. `app`: This variable defines a Flask application instance named "app".

XI. `MINIMUM_LUMINANCE`: An integer value specifying the minimum luminance required to activate the IR LEDS.

XII. `socket`: Localhost video stream socket (tcp://*:5556) for the CSI camera stream.

XIII. `camera_process`: A Subprocess object of the Python 3.6.9 `csicam.py`.

2.4.2 Flask related Functions/Routes

I. `update_value(value_name, enum_type)`: This function updates a global variable with the given name and type using data from a JSON request. It performs validation by attempting to convert the selected value to the specified enum type

II. `set_location()`, `set_weather()`, `set_mode()`, `set_clear()`: These functions are Flask routes that handle updating the corresponding global variables (location, weather, mode, and clear) based on user selections in the web interface. They perform data validation and return success or error messages.

III. `load_user(username)`: This function, required by Flask-Login, is used to load the user object by username.

IV. `login()`: This function handles login requests. It retrieves username and password from the login form submission, validates user credentials, and redirects the user to the main page (video) upon successful login. Otherwise, it returns an error message.

V. `logout()`: This function logs out the user by calling the logout_user function from Flask-Login and redirects the user to the login page.

VI. `video()`: This function renders the video.html template, which displays the video stream and user interface for selecting a location, weather, mode, and control data collection.

VII. `video_feed()`: This function generates a video feed by streaming video frames from the camera, detecting people using the YOLO model, and potentially saving detections based on the selected mode. It utilizes the `stream_detect_people` function to achieve this.

VIII. `stop()`: This function sets the running flag to False, effectively stopping the video processing loop. It also logs the number of images saved during the session.

IX. `get_images()`: This function retrieves the image paths from the output directory and returns them as a JSON response.

X. `start()`: This function sets the running flag to True, initiating the video processing loop.

XI. `get_logs()`: This function retrieves the weblogs as a string, joins them using newline characters, creates a response object with the content type set to `text/html`, clears the weblogs, and returns the response.

XII. `download_csv()`: This function retrieves the CSV file corresponding to the current data collection mode (`mode`) from the "results" directory and allows the user to download it.

2.4.3 Core Functionality Functions

I. `stream_detect_people()`: This function is the core of the video processing and detection loop. Here's a breakdown of its functionality:

- Calls `init_light_sensor()`
- Measures the initial luminance using `measure_luminance` to see if the lux is 0. If it is, activate the IR LEDs to activate.

- Initializes global variables and lists for detection results, confidence scores, luminance data (for luminance mode), and image count.
- Clears images based on the `INITIALLY_CLEAR_IMAGES` flag at the beginning (unless otherwise specified).
- Loads the `YOLO` model and checks if `CUDA` is available for GPU acceleration.
- Retrieves the current frame transmitted from `csicam.py`.
- Runs a loop that continues until stopped by the user or when the timer expires in location and weather modes:
  - Reads a frame from the camera.
  - Runs object detection on the frame using the YOLO model.
  - Updates a list named `detected_people` with information about detected people (confidence score and bounding box coordinates).
  - If people are detected:
    - Sends a "Ring" message to the alarm system.
    - In luminance mode:
      - Saves the frame for each detection.
      - Measures the luminance using the light sensor and stores it with the corresponding confidence score in `luminance_data`.
    - In location or weather mode:
      - Saves the frame containing the detected people.
  - Draws bounding boxes and confidence scores on the frame (optional).
  - Encodes the frame as a `JPEG` image for streaming.
  - Yields the encoded frame for the video stream.
  - Clears the `detected_people` list for the next frame.
  - After the loop exits (due to user stop or timer):
    - In luminance mode:
      - Saves the collected luminance and confidence data to a CSV file.
    - In location or weather mode:
      - Calculates the average accuracy for detections based on confidence scores.
      - Saves the average accuracy and data type (`location` or `weather`) to a CSV file.
  - Releases the video capture object and closes `OpenCV` windows and terminates the `csicam.py` sub process.

II. `measure_luminance()`: This function (used in luminance mode) measures the current luminance using the TSL2591 light sensor and returns the value in lux. Run `init_light_sensor()` before running this function.

III. `init_light_sensor()`: This function initializes the I2C communication with the TSL2591 light sensor, and returns the sensor object.

IV. `draw_detections_and_info(img, detected_people, display_detections=False)`: This function draws bounding boxes and confidence information for detected people on the provided image. It allows the number of detected people to be optionally displayed.

V. `save_results(data_type, data, accuracy)`: This function saves the specified data type (e.g., location, weather, luminance), data (e.g., location information, weather condition, luminance value), and accuracy (average confidence score) to a CSV file. It handles creating the CSV file, clearing existing data (except header/column names) based on a flag, and appending the new data entry.

VI. `update_detected_people(results, detected_people, confidences)`: This function updates the detected_people list with unique detections and their confidence scores. Here's a breakdown of the process:

- Iterates through the detection results from the YOLO model.
- For each bounding box:
  - Calculates the confidence score using `_calculate_confidence` and adds the confidence score and bounding box coordinates to the `detected_people` list.
  - Appends the confidence score to the confidence list.

VII. `coordinates_in_int(box)`: This function converts the coordinates of a bounding box from float to integer values.

VIII. `_calculate_confidence(box)`: This function calculates and rounds the confidence value (average precision) from the YOLO model's detection output. It multiplies the model's confidence with the Intersection-over-Union (IoU) value and ensures the result is between 0 and 1.

IX. `get_frame_from_stream()`: The function retrieves the camera frames from the CSI port using the socket created in `csicam.py`

The rest of the files have minimum code and are straightforward. Refer to the logs and the comments provided for an overview of these files.

2.5 Client-Sided

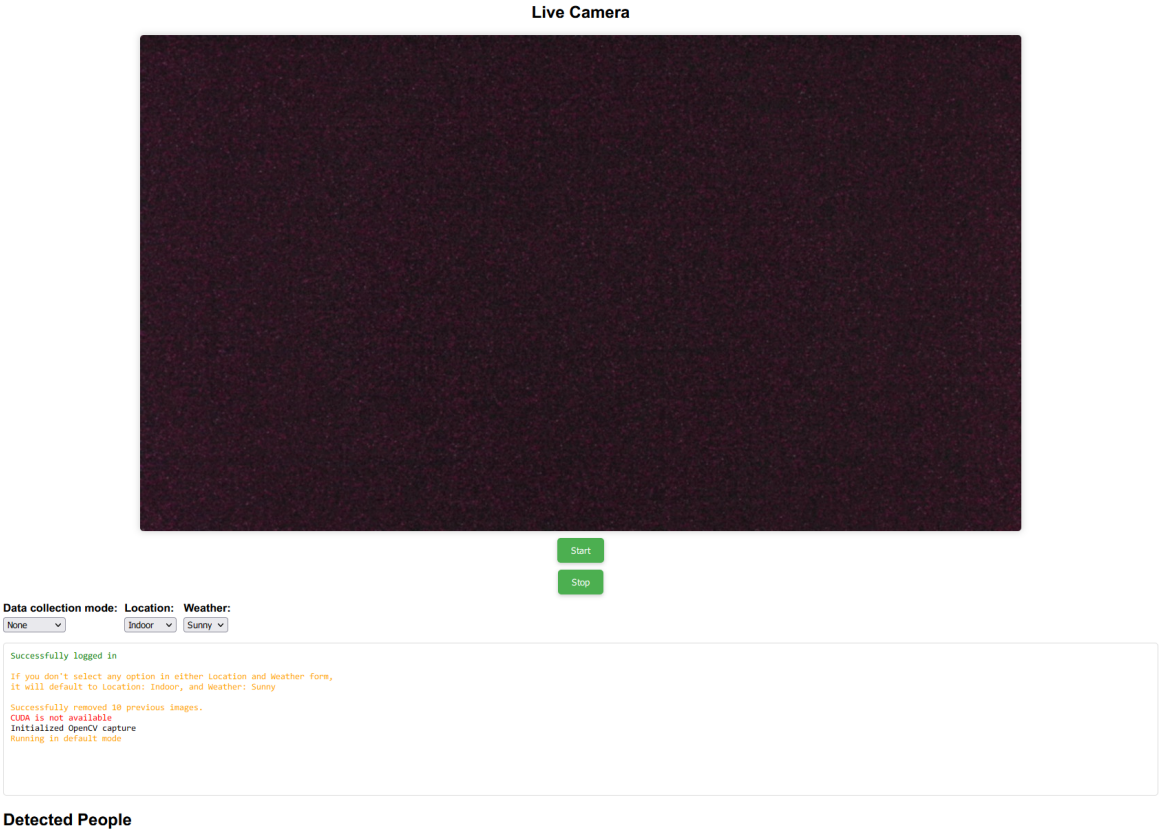

Figure 8: Flask server UI

The client-side essentially displays the HTML of the Flask server. The client is requested for login credentials for authentication and security purposes. When successfully logged in, it shows the camera stream with the camera module connected to the Jetson and the detecting and bounding boxes outputted from the YOLOv8 model. There are two buttons, START and STOP, which start and stop the camera stream and the data collection. Underneath these buttons, there are four dropdown menus:

I. Data collection mode: This tells the Jetson or main.py what data types to collect. In luminance mode, every time a detection is detected, it measures the lux and stores it in a CSV file. In weather or Location mode: For this mode to work, the client can set the weather and location parameters remotely using the respective dropdown menus. When these values are set, the camera stream and data collection will run for 10 minutes, calculate the average accuracy of all detections, and store it in a CSV file.

II. The location and weather dropdown menus allow the client to set parameters.

III. Clear dropdown: this dropdown essentially clears the necessary CSV files before adding new values from a new session.

IV. WARNING: There is an issue when the mode is set to 'None,' it has problems storing the data in the CSV files.

Underneath the dropdowns is a container containing all the logs the core files provide, such as `main.py`. This container allows remote debugging.

Lastly, underneath the log container, another container shows the images collected of all detections for verification purposes.

2.6 Alarm System

The alarm system controls a remotely activated alarm through a serial connection. It defines a pin for the alarm and communicates with a computer. It listens for the message "ring" on the serial port. If received and the alarm isn't already on, it activates the alarm pin with a specific brightness using pulse width modulation for 5 seconds. Then, it turns off the alarm and remembers the activation time. The code also prevents continuous ringing by requiring a minimum wait time (20 seconds) before allowing another activation through the "ring" message.

2.6.1 Configuration

I. `alarmActive`: Boolean flag to track if the alarm is currently activated.

II. `lastAlarmTime`: Stores the timestamp (in milliseconds) of the last alarm activation.

III. `reactivation_delay`: Constant integer representing the delay (in milliseconds) before the alarm can be reactivated (1 minute).

IV. `activation_time`: Constant integer representing the duration (in milliseconds) the alarm will sound for (5 seconds).

2.6.2 Functions

I. `setup()`: Configures the alarm and UART receive pins as output and input, respectively, and Initializes serial communication at a baud rate of 9600.

II. `loop()`:

- Checks if there's any data available on the UART.
- Reads the incoming message until a newline character is received and removes any leading or trailing spaces.
- Compares the received message with `ring`. If it matches, it triggers the `activateAlarm` function.
- Checks the `alarmActive` flag. If true, calculate the elapsed time since the last alarm activation.
- Resets the `alarmActive` flag if the reactivation delay has passed (allowing another alarm activation).

III. `activateAlarm()`:

- Checks if the alarm isn't already active.
- Activates the alarm for the set activation time.
- Turns off the alarm after the activation time.
- Sets the `alarmActive` flag to true and updates the `lastAlarmTime` timestamp.

3 Data Related

The main.py or the flask server will generate CSV files from data collection, and they can be in the form of the following tables:

Luminance (lux)	Accuracy
660	80.1
510	45.1
.	.
.	.

Figure 9: Sample Luminance data

Location	Average Accuracy
Indoor	81.5
Outdoor	91.2
.	.
.	.

Figure 10: Sample Location data

Weather	Average Accuracy
Sunny	99.1
Rainy	67.1
.	.
.	.

Figure 11: Sample Weather data

These CSV files are found in the results/ directory, or you can remotely download these files using the flask server when you click the "STOP" button.

Refer to the save_results() function in the main.py for more information.

4 Appendix 1

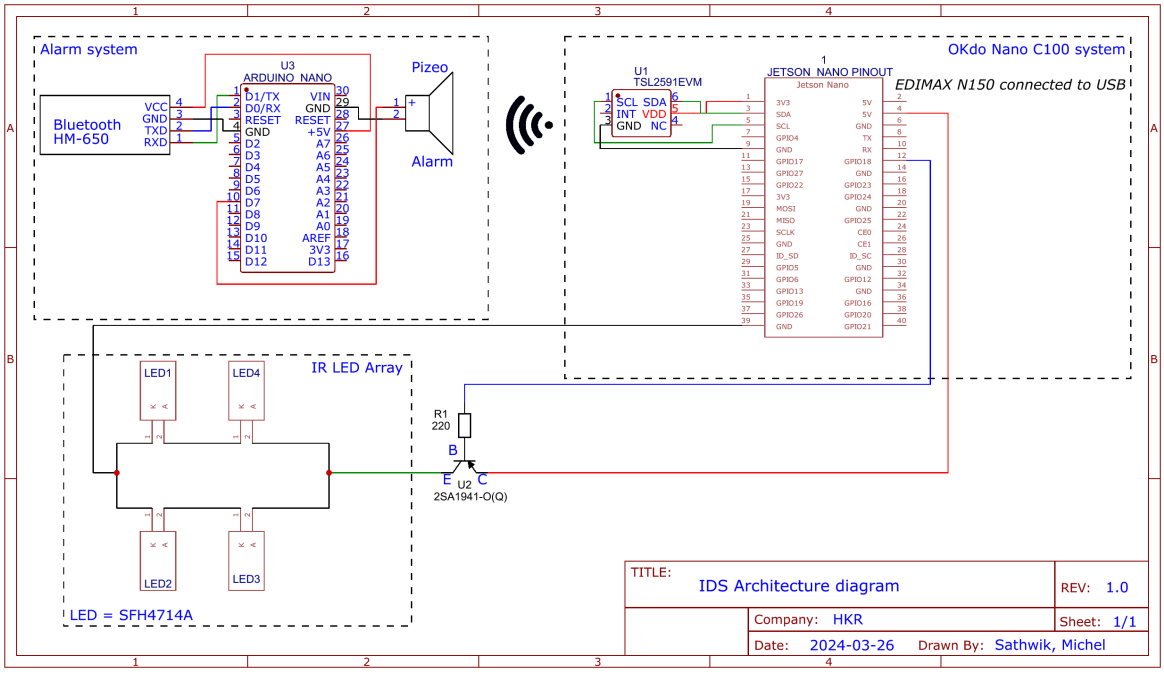

Figure 12: Hardware Overview

5 Appendix 2

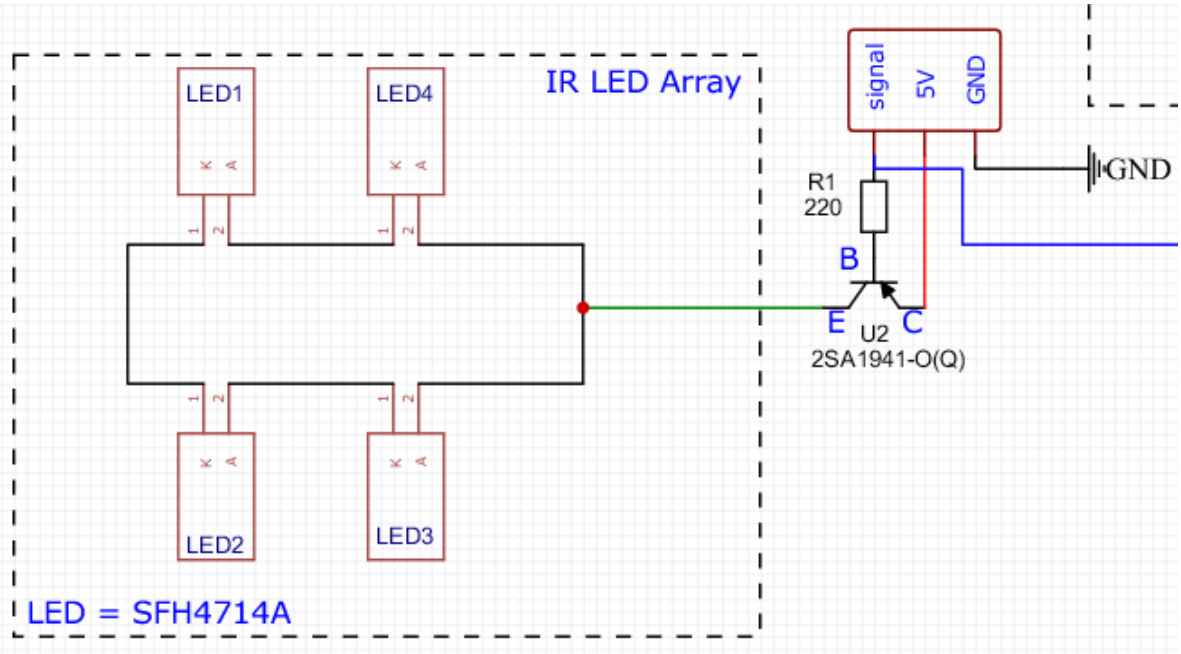

Figure 13: IR LEDs array

References

[1] OKdo, *OKdo Nano C100 Developer Kit powered by NVIDIA® Jetson Nano Module*, Available: https://www.okdo.com/us/getting-started/get-started-with-the-c100-nano-csi-cameras/, OKdo, 2024

[2] Adafruit, *Adafruit TSL2591 High Dynamic Range Digital Light Sensor - STEMMA QT*, Available: https://www.adafruit.com/product/1980, Adafruit, 2023

[3] ams-osram, *OSRAM OSLON® Black, SFH 4714A*, Available: https://ams-osram.com/products/leds/ir-leds/osram-oslon-black-sfh-4714a, ams-osram, 2024

[4] LetsOKdo, *csi-camera*, Available: https://github.com/LetsOKdo/csi-camera/blob/master/simple_camera.py, csi-camera, 2024

[5] eggheadtwins, *YOLOTHESIS*, Available: https://github.com/eggheadtwins/YOLOTHESIS/tree/bluetooth, GitHub, 2024