A Low-Cost, High-Performance AI-based Night Vision Device for Small Arms: Designed for the Armed Forces

Md ushama shafoyat[1], Sheyeam husnain[2], injamamul haque niyaz[3], ahnaf jubaer saad[4], Mohummad Shariful Islam[5]

[1,5] *Department of Biomedical Engineering, Military Institute of Science and Technology (MIST), Dhaka 1216, Bangladesh*

[2,3,4] *Department of Mechanical Engineering, Military Institute of Science and Technology (MIST), Dhaka 1216, Bangladesh*

***Abstract*—This study presents the design and implementation of a low-cost object-detecting night vision device based on the Raspberry Pi platform for integration with small arms used by the Armed forces. The system employs an infrared (IR)-sensitive camera module, IR LEDs, and a Raspberry Pi microcomputer to enable image capture and video streaming in low-light and no-light environments. Image processing and real-time video analytics, including motion detection, object tracking, and basic recognition, are performed using Python-based libraries and OpenCV. The compact and energy-efficient configuration enhances its suitability for applications such as remote surveillance, target aiming, monitoring hostile movement, securing restricted areas, and conducting night-time reconnaissance operations. Compared to conventional night vision technologies, the proposed solution offers a lightweight, scalable, and cost-effective alternative. The use of open-source hardware and software ensures accessibility, adaptability to mission-specific requirements, and potential for further development. This work demonstrates the feasibility of employing open-source embedded platforms to create affordable and customizable night vision systems for defense, security, and tactical applications.**

1. INTRODUCTION

A night-vision device (NVD), also referred to as a night optical/observation device (NOD) or night-vision goggle (NVG), is an optoelectronic system that enables visualization in low-light or no-light environments. These devices play a vital role in military and security operations as well as civilian activities such as hiking and night-time exploration. Modern object detection in night vision devices relies on advancements in sensors, signal processing, wearable systems, and portable integrations, though significant challenges remain in low-light environments. Bhuvaneswary et al. [1] developed a Raspberry Pi–based military surveillance robot with multiple sensors and a night camera, enabling remote monitoring in war zones. In [2], crime analysis methods such as k-NN, Genetic Algorithms, and Decision Trees were applied, highlighting the role of AI in detection tasks. Raspberry Pi–based approaches using Adaboost, bagging, and reweighting [3] demonstrated promising results but were constrained by power supply needs. Enhancements such as multiscale features and key point representation [4, 5] improved accuracy but performed poorly under extreme low-light. Traditional low-light enhancement methods, including histogram mapping [6–8] and Retinex-based optimization [9, 10], adapt illumination but struggle with noise. The ExDark dataset [11] provides low-light benchmarks, while domain adaptation techniques [12] attempt to merge pretrained models for improved performance. Inter-frame differencing [13] and its double-difference variation [14] offered efficient detection but faced limitations like ghosting and unreliable texture detection.

Deep learning frameworks have further advanced the field: Song et al. [15] used an enhanced YOLOv5s network for remote sensing images, while Peng et al. [16] proposed NLE-YOLO for low-light detection with improved receptive fields. Wang et al. [17] introduced DK_YOLOv5 using ExDark and Mine_ExDark datasets for underground detection.

Rahim et al. [18] applied YOLOv4 for real-time distance monitoring, and Zhao [19] improved YOLOv7 with hybrid modules for better edge extraction. Nazib [20] developed a Kalman filter–based tracking system for low-contrast surveillance, while Namana [21] demonstrated YOLOv8's superior performance on ExDark using HPC and image augmentation.

Despite their importance, the advanced night vision devices currently used by the Bangladesh Army are highly expensive, limiting their widespread deployment. Administrative restrictions and the shortage of trained personnel for maintenance further constrain their effective utilization.

To overcome these limitations, this study proposes the design and development of a cost-effective, fabricated night vision device based on the Raspberry Pi platform. The system employs an IR-sensitive camera module and IR LEDs to capture images and videos in darkness, while real-time processing with Python and OpenCV enables object detection, motion tracking, and enhanced visibility. Unlike conventional bulky systems, the proposed device is compact, affordable, and programmable, allowing for integration of additional sensors and algorithms to adapt to varied mission requirements. By offering an accessible and scalable solution, the project seeks to broaden the operational use of night vision technology across multiple defense and security applications.

.

2. MATERIALS & METHODOLOGY

A good methodology is critical to building a system that works properly. Delivering buggy systems to the users can always lead to dissatisfaction. In these sorts of applications, such as software model-based or programming-based equipment, it is hard to destroy until the basic programming data is cracked or changed. For our project, we have followed the successive refinement development process as we aim to refine the designed system through successive updates of the requirements and design (Figure 1).

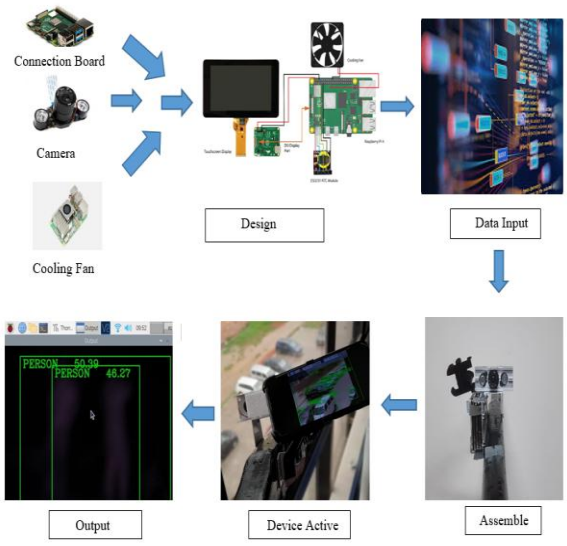


Figure 1: Methodology Flow Chart

In the SSD algorithm, there are 3 Steps. These are discussed below:

a) Object-ident.py — Basic Object Detection Script

Purpose:

i. This is your initial prototype, a simple script that uses a pre-trained MobileNet SSD model to perform object detection on live video frames from your webcam or camera.
ii. Detects multiple object types.
iii. Draws bounding boxes and class labels.
iv. Uses a confidence threshold to filter out weak detections.
v. Real-time display but no filtering or logging.

b) Object-ident-2.py — Improved Real-Time Detection with Label Filtering

Purpose:

i. This script builds upon the first by improving efficiency and accuracy in real-time detection.
ii. Resizes frames for faster processing.
iii. Adds label filtering, possibly to ignore unwanted classes.
iv. Enhances real-time performance.
v. More refined display of output.

c) Object-ident-3.py — Detection with Timestamp & Logging

Purpose:

i. This is the most advanced version, adding logging and timestamping features for monitoring and later review.

ii. Saves detected object labels along with timestamps.
iii. Keeps a record (log) of detections.
iv. Essential for post-analysis or evidence in surveillance.
v. Maintains all features from the earlier scripts.

These are the Object Detection Methodology that we followed in the process of the algorithm.

2.1 Components

2.1.1. Raspberry Pi Connection Board

The Compute Module 4 IO Board is a companion board for Raspberry Pi Compute Module 4 supplied separately). It is designed for use both as a development system for Compute Module 4 and as an embedded board integrated into end products. The IO board is designed to allow you to create systems quickly using off-the-shelf parts such as HATs and PCIe cards, which might include NVMe, SATA, networking, or USB. The major user connectors are located along one side to make enclosures simple. The Compute Module 4 IO Board also provides an excellent way to prototype systems using Compute Module 4 (figure 2).

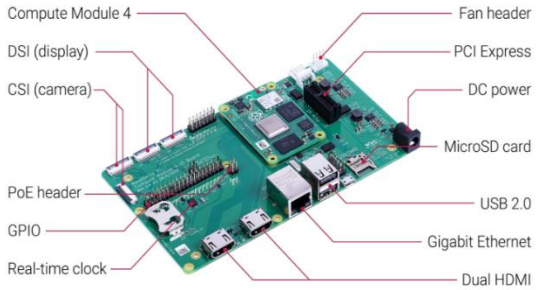


Figure 2: Raspberry Pi Connection Board 4

Specification:

a. Processor: Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
b. Memory: 1GB, 2GB, 4GB or 8GB LPDDR4 (depending on model) with on-die ECC
c. Connectivity: 2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0,
d. Video & sound:
i. 2 × micro-HDMI ports (up to 4Kp60 supported)
ii. 2-lane MIPI DSI display port
iii. 2-lane MIPI CSI camera port
e. Multimedia: H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics
f. SD card support: Micro SD card slot for loading operating system and data storage
g. Input power:
i. 5V DC via USB-C connector (minimum 3A1)
ii. 5V DC via GPIO header (minimum 3A1)
iii. Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
h. Environment: Operating temperature 0–50ºCRaspberry Pi 4 Model B – Raspberry Pi Ltd 4, MTBF1
i. Ground Benign: 211 000 hours
j. Production lifetime: Raspberry Pi 4 Model B will remain in production until at least January 2034.

2.1.2 Night Vision Camera

The RPI IR-CUT Camera (B) offers superior imaging quality for both day and night use. Designed for Raspberry Pi, it features a 5 MP OV5647 sensor capable of capturing high-resolution 1080p images and videos. Its removable IR-CUT filter ensures true-to-life colors during the day while eliminating distortions. At night, the included infrared LEDs provide excellent visibility in low light conditions. With adjustable focus and compatibility with all Raspberry Pi models, this versatile camera module is perfect for surveillance, robotics, and lot projects. Compact in design, it includes mounting holes for easy and stable integration into any setup.

Features:

a. Day & Night Clarity: Embedded IR-CUT filter eliminates color distortion for vibrant daytime imaging.
b. Night Vision Support: Infrared LEDs provide visibility in low-light or nighttime conditions
c. High-Resolution Sensor: 5 MP OV5647 sensor for capturing sharp 1080p images and videos.
d. Adjustable Focus: Customize the focus distance for precise imaging
e. Universal Compatibility: Supports all Raspberry Pi board revisions.
f. Secure Mounting: includes 4 screw holes for easy and stable installation.
g. Power Output: Provides 3.3V power output for additional infrared or flash LEDs

Table 1: Raspberry Pi Camera Specifications

Lens Type	Fixed with adjustable focus	Mounting Holes	4 screw holes
Field of View (Diagonal)	40°	Dimensions	31mm x 32mm
CCD Size	1/4 inch	Shipment Weight	0.086 kg
Aperture (F)	1.4	Shipment Dimensions	10 x 8 x 4 cm
Focal Length	8 mm	Night Vision	Supported with an infrared LED
IR-CUT Filter	Removable	Power Output	3.3V

General Specifications:

a. Day & Night Clarity: Embedded IR-CUT filter eliminates color distortion for vibrant daytime imaging.
b. Night Vision Support: Infrared LEDs provide visibility in low-light or nighttime conditions
c. High-Resolution Sensor: 5 MP OV5647 sensor for capturing sharp 1080p images and videos.
d. Universal Compatibility: Supports all Raspberry Pi board revisions.
e. Power Output: Provides 3.3V power output for additional infrared or flash LEDs

Raspberry Pi Camera Different Module

Here is the photo of the Transmittance (%) vs. Wavelength (nm) plot for the Raspberry Pi HQ and GS Camera Modules, serving as their transmissivity curve (figure 3).

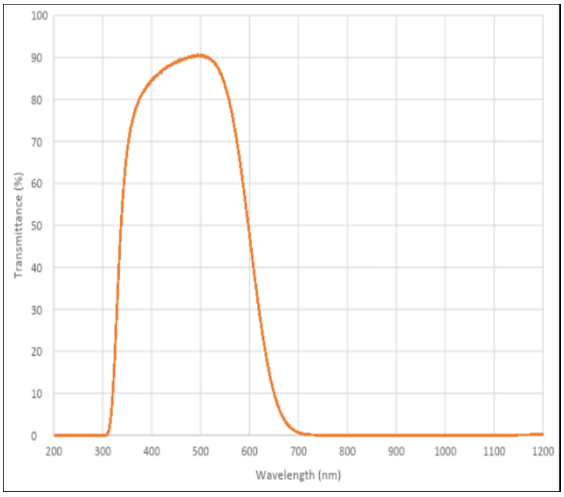


Figure 3: Transmissivity Curve for HQ (High Quality) and GS (Global Shutter) Camera

The transmittance is very low (~0%) for wavelengths below 300 nm, indicating strong UV light blocking. Between approximately 320 nm and 600 nm, the transmittance rapidly increases and reaches a peak of around 90%, meaning the material allows most light in this range to pass through. This range corresponds to the visible light spectrum. After 600 nm, the transmittance drops sharply, reaching near 0% again by 700 nm, suggesting that infrared light is also blocked. This material or filter acts as a visible light bandpass filter, transmitting light efficiently in the visible range while blocking both ultraviolet and infrared regions. Such characteristics are common in optical lenses, sensors, or protective eyewear designed to limit exposure to UV and IR radiation.

Here is the graph of the HQ Camera's spectral response without the IR-Cut filter**, as shown in** Figure 4, Relative Response (%) vs Wavelength (nm)**.**

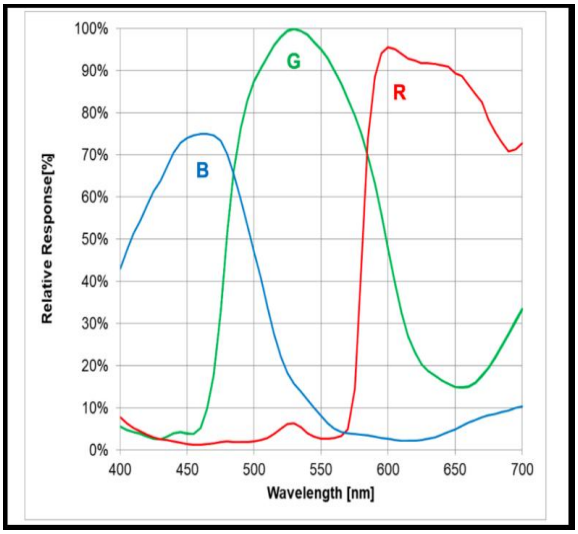


Figure 4: HQ Camera without IR-Cut filter

This graph is important for understanding how color sensors capture light and convert it into digital color information. The green channel has the highest peak response, often around 100%, which is typical due to the human eye's greater sensitivity to green. There's some overlap between the curves, which helps the sensor interpret color mixtures. Each curve drops off in response outside its optimal wavelength range, reducing sensitivity to unrelated colors.

Here is the photo illustrating the spectral response of the Raspberry Pi Global Shutter Camera without IR-Cut filter (Figure 5), showing its sensitivity per channel, normalized to 0-1.0.

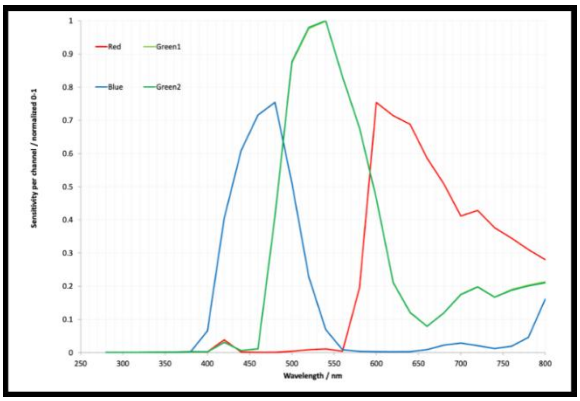


Figure 5: Raspberry Pi Global Shutter Camera without IR-Cut filter

This graph characterizes how an image sensor with a Bayer filter captures light across different wavelengths. The overlapping yet distinct sensitivity ranges for red, green, and blue channels are essential for accurate color image reconstruction.

Observations:

1. Green channels dominate the mid-spectrum (around 540–550 nm) and have the highest peak sensitivity, which aligns with human visual perception.
2. Blue channel responds best to shorter wavelengths (~460 nm) and tapers off quickly after 500 nm.
3. Red channel is responsive to longer wavelengths (~600 nm) and maintains some sensitivity into the near-infrared.
4. Both green channels are very similar, suggesting minimal variation between them—possibly due to slight manufacturing or sensor layout differences.
5. The sensor has little to no sensitivity in the UV range (<400 nm), as expected for most CMOS/CCD sensors with Bayer filters.

2.1.3 Display

Raspberry Pi Display or normal smart phone is used for the connection.

For Raspberry pi the display configurations are,

a. Pre-existing Linux/Windows/Mac drivers
b. 800 x 480 touchscreen
c. Supports Windows XP SP3, Windows 7, Windows 8, Windows 8.1, Windows 10, Android 4.2, Windows CE7, Ubuntu and Debian.

2.2 Contrast Change Method.

It is very clear that traditional algorithms are not perfect for detecting objects at night. If we think about the human perception of the human visual system, then we understand that the eye catches objects due to the rapid change of contrast. So here we use that module, which performs in two stages. In the first stage, objects are detected using contrast change measured by taking sub-image interframe differences. In the second stage, motion prediction is performed and considering the nearest data association, which gives fine feedback to the first stage.

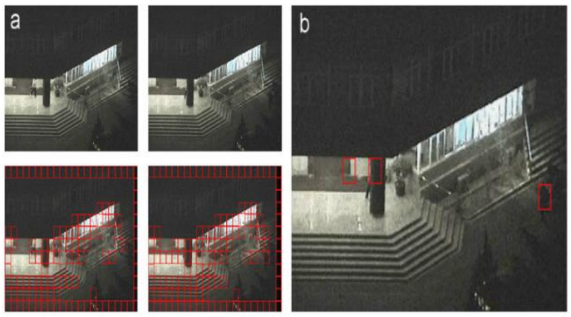


Figure 7: a) Object detection based on contrast, b) moving object detection based on contrast change

2.3 Software

The Raspberry Pi OS 64-bit version is a Debian-based Linux operating system developed specifically for the Raspberry Pi's ARM64 architecture. This version is designed to take full advantage of 64-bit processors like the Broadcom BCM2711 found in the Raspberry Pi 4 and Raspberry Pi 5. Unlike the earlier 32-bit version, the 64-bit OS allows applications to use more than 4 GB of RAM, which is crucial for models with 8 GB of memory, thereby enhancing performance for memory-intensive tasks.

Raspberry Pi OS 64-bit supports a broader range of modern applications and libraries that require a 64-bit environment, including newer versions of Python, Node.js, and Docker. It also improves performance in multi-threaded applications and scientific computing tasks. The system uses the APT package manager and supports Flatpak and Snap packages for additional software.

3. RESULTS & DISCUSSIONS

The Raspberry Pi-based night vision device successfully captured and transmitted low-light images with enhanced clarity. Integration of infrared technology and real-time video streaming validated system efficiency. Results demonstrated reliable

performance in dark environments, proving its potential for surveillance and security applications in military and remote monitoring operations. The number of pictures is captured and take into comparison to other available devices developed before. The result is quite satisfying compared to other devices available.

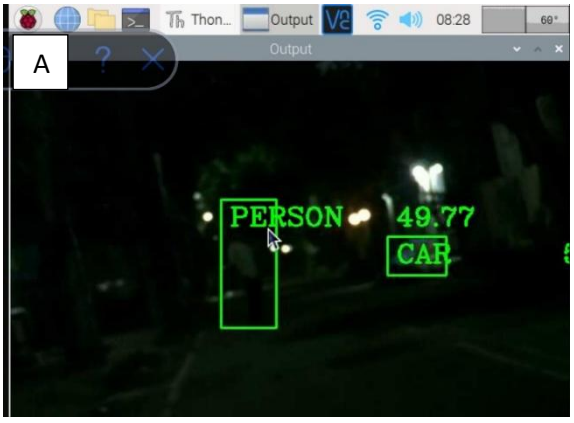
Several pictures were captured to see how the object is detected during daytime for moving and a stationary object. Figures 8 and 9 below are taken from various distances to see the effectiveness of the lenses. Though it's not the main concern for this research, the Effectiveness of the lenses from a far distance depends on the resolution and focal length.

Figure 8: (A, B) Multiple Objects are identified from 75-meter distances.

Figure 9: (A, B) various things are detected including chair, table and car

3.1 Night Time Object Detection

Night Times are classified as low light condition, No Light Condition and Natural Light Condition. For the all cases the objects are perfectly visible from 10 to 75 meters distance (figure 10).



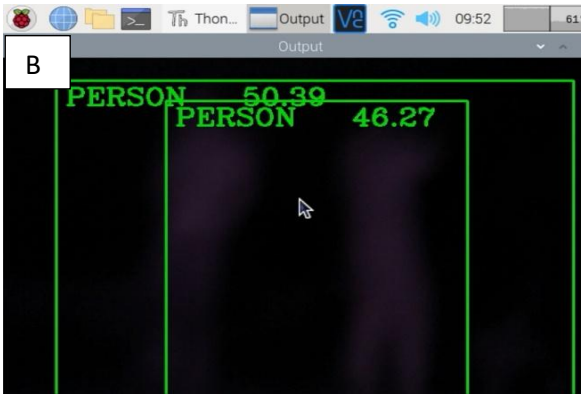


Figure 10: (A, B) Nighttime object detection.

Weather Forecast Report and Moonlight Condition on the Test Day:
Date: 22 May, 2025
Time: 2300 Hour
Table 2: Weather and Moon Condition Report from AccuWeather.com

Temperature	28.3°
Precipitation	0.2 mm
Wind	17.6Km/h
Humidity	79 %
Cloud Cover	73%
Visibility	9 km
Weather	Patchy Rain Possible
Moon Phase	Waning Crescent
Illumination	28-33%
Moon Rise	1.24 a.m
Visibility With Night Vision	Fair

3.1.1 Data Table for Distance
Table 3: Distance Chart for a 3 Megapixel IR-Cut Night Vision Camera

Object Type	Approx. Detection Distance (Daylight)	Approx. Detection Distance (Night / IR)
Person (Standing/Moving)	8–12 meters	4–6 meters
Car	15–25 meters	8–12 meters
Bicycle/Motorbike	10–15 meters	6–8 meters
Animal (Dog/Sheep)	5–8 meters	3–5 meters
Large Object (Bus/Truck)	20–30 meters	10–15 meters

5.3.3 Case Study
a) Case-1
i. Pre-Condition: Normal eye view, some object is visible naturally
ii. Post Condition: The Object is identified, which is not even marked in normal eye view.

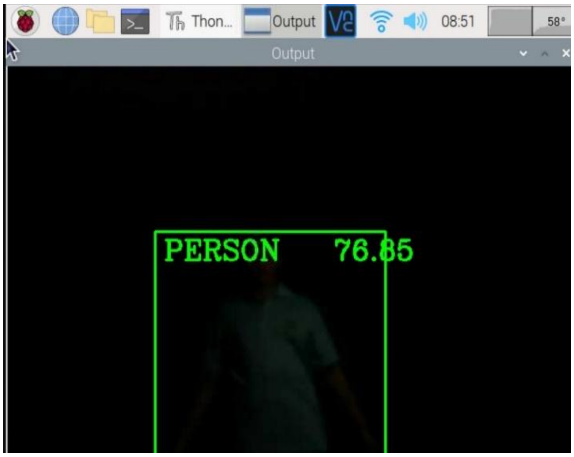


Figure 11: Case-1

b) Case-2
i. Pre-Condition: Normal eye view is totally null and dark. Nothing at all is identified by normal eye view.
ii. Post Condition: Object is clearly visible and well identified

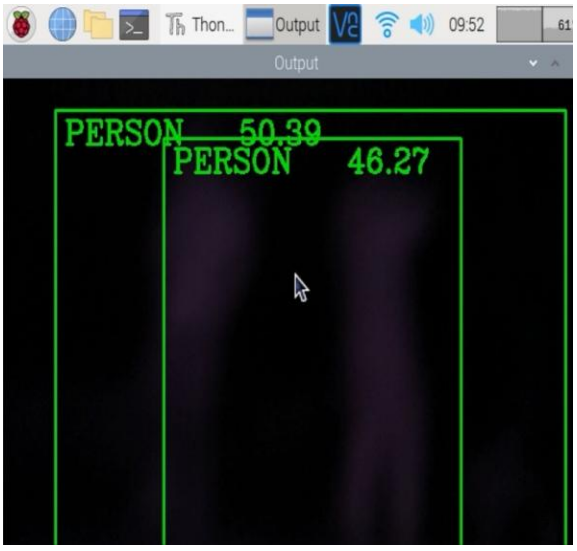


Figure 12: Case-2

c) Case-3
i. Precondition: Everything is clearly visible by normal eye view.
ii. Post Condition: Clearly seen by the camera and object is detected

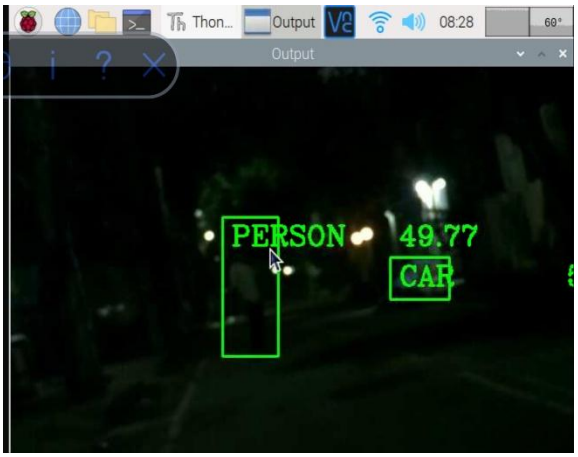


Figure 13: Case-3

3.2 Comparison with Various Research Works

We have studied several research papers from which we could find out our progress in this night vision and object detection. For low-light conditions, the picture quality was enhanced multiple times.

Table 4: Various Algorithms and Roles in Object Detection.

Algorithm	Full Form	Role In Object Detection
DLF	Deep Learning Framework	Backend infrastructure for training & inference
IAT	Illumination Adaptive Transformation	Enhances images for better feature extraction
LL Flow	Low-Light Flow Estimation	Motion tracking under dark conditions
SID	See In the Dark Dataset	Low Light Enhancement

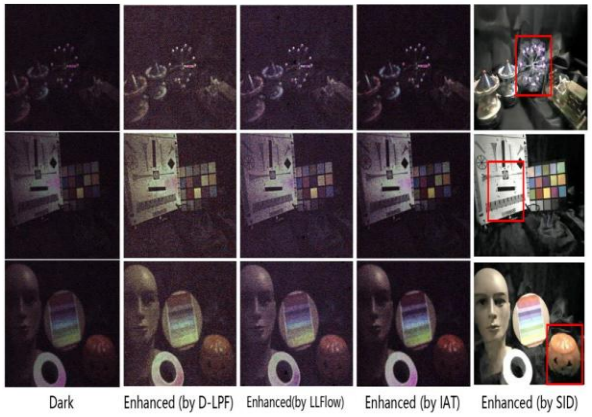


Figure 14: Comparison of our pictures from different procedure

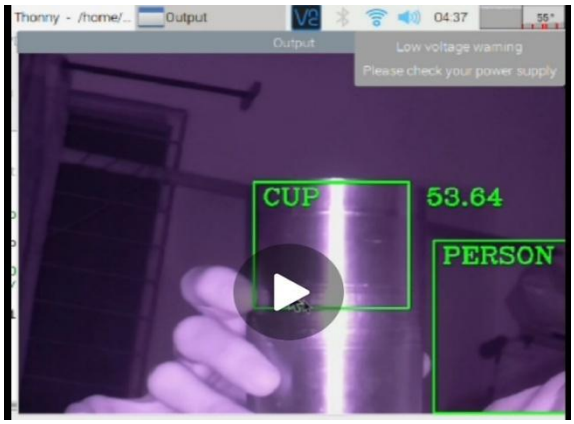


Figure 15: Enhanced by Raspberry Pi (SSD Algorithm)

So, the output result after comparing with various object detection algorithms, we can say our algorithm is fast, accurate, and can detect multiple objects at once. The IAT, LPF, and SID have some limitations, whereas by implementing the SSD algorithm, we could successfully minimize those perspectives.

3.3 Comparison with MOG and NPB

MOG (Mixture of Gaussians) is an algorithm used mainly for background subtraction in video processing. It helps separate moving objects (like people or cars) from the static background in a video feed. NPB stands for Nonparametric Bayesian methods, a category of algorithms that use Bayesian statistics without assuming a fixed number of parameters or components in advance. One popular application of NPB is in clustering or modeling data when the number of groups or patterns is unknown.

Table 5: Comparisons between the MOG-, NPB-based algorithms on the different sequences of frames.

Situation	SSD (Mean Average Precision) (OWN)	MOG	NPB
Two persons at night	47.2%	78.65%	85%
Traffic Sequence	82.6%	55.71%	63.98%
Person at a distance	32.4%	Low than 10% (ignored)	Low than 10% (ignored)
Zooming camera sequence	Variable	Lower than 10% (ignored)	Lower than 10% (ignored)

Final Result	SSD achieved a mean Average Precision (mAP) of 47.2% in low-light conditions with multiple individuals present.	MOG performance is moderate; it can detect motion but struggles with distinguishing multiple overlapping objects in low-light scenarios.	NPB is not applicable for this purpose. It is designed for benchmarking parallel computing performance, not object detection.

3.4 Final Result

The SSD model achieved a mean Average Precision (mAP) of 47.2% in low-light scenarios with multiple individuals, outperforming traditional methods like MOG and NPB in complex conditions. SSD showed strong performance in traffic sequences (82.6%) but struggled with distant persons and zooming camera sequences. MOG performed moderately in detecting motion (78.65% in night scenes) but was less effective in distinguishing overlapping objects. NPB was not suitable for object detection tasks, as it is designed for evaluating parallel computing performance rather than vision-based detection tasks. Overall, SSD demonstrated superior adaptability in varied and challenging environments.

SSD clearly outperforms MOG and NPB in complex visual tasks, especially under low-light and dynamic conditions. While MOG shows moderate effectiveness in simple motion detection, it fails in nuanced scenarios. NPB is irrelevant for object detection tasks, emphasizing the importance of using domain-specific models like SSD for accurate results.

4. CONCLUSIONS

Object detection and tracking at night are vital for effective 24-hour surveillance in military bases, borders, and restricted facilities. Given the high cost and limited usability of current NVDs in the Bangladesh Army, this research aimed to develop a low-cost, Raspberry Pi–based night vision system using a 3-megapixel IR camera and the Single Shot MultiBox Detector (SSD) algorithm. The device, experimentally mounted on a wooden rifle structure, employed contrast-based detection refined through frame-to-frame tracking, improving accuracy and reducing false positives from sensor noise. A custom low-light dataset including pedestrians, vehicles, and partially visible objects was used for testing. Results showed successful detection and tracking of multiple objects under poor lighting, outperforming some conventional systems while staying within Raspberry Pi's computational limits. Despite these promising results, deployment challenges remain. The current system lacks the durability, weather resistance, and precision required for battlefield use, and latency in real-time, high-speed conditions poses risks in combat scenarios. Additionally, the improvised mounting setup lacked stability and integration with military targeting systems. Nevertheless, this project demonstrates the feasibility of creating a compact, affordable, and open-source night vision solution. Future work should focus on ruggedizing the hardware, expanding datasets, and adopting optimized deep learning models to enhance robustness and efficiency. This research establishes a foundation for low-cost surveillance technologies with potential applications in both civilian and military domains.

LIST OF ABBREVIATIONS

Sign / Letter / Symbol	Representation
NVD	Night Vision Device
NVG	Night Vision Goggles
NOD	Night Optical/Observation Device
IR	Infrared
IR-CUT	Infrared Cut-off Filter
CCD	Charge-Coupled Device
CMOS	Complementary Metal-Oxide-Semiconductor
LED	Light Emitting Diode
LUX	Light Unit Exposure (used to measure illumination)
LIDAR	Light Detection and Ranging
FLIR	Forward Looking Infrared
IIT	Image Intensifier Tube
SSD	Single-shot multi-box detector
SNR	Signal-to-Noise Ratio
Pi	Raspberry Pi (used as a processing unit in DIY NVD system
MOG	Mixture of Gaussians (MOG)
NPB	Nonparametric Bayesian Method
LOD	Level of Details
COCO	Common Objects in Context
RGB	Red Green Blue

APPENDIX

cmd lines.txt

```
sudo vcgencmd get_camera
dtoverlay=vc4-fkms-v3d
dtoverlay=imx219
dtoverlay=ov5647, media-controller=1
sudo nano /etc/dphys-swapfile for swap file access
sudo nano /boot/config.txt    for config.txt access
cd ~/opencv-4.4.0/build/
make -j $(nproc)
```

COCO NAMES

Person, bicycle, car, motorcycle, airplane, bus, train, truck, boat, traffic light, fire hydrant, street sign, stop sign, parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe, hat, backpack, umbrella, shoe, eye glasses, handbag, tie, suitcase, frisbee, skis, snowboard, sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket, bottle, plate, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, chair, couch, potted plant, bed, mirror, dining table, window, desk, toilet, door, tv, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, blender, book, clock, vase, scissors, teddy bear, hair drier, toothbrush, hair brush.

CONFIG REF

```
uncomment the following to adjust overscan. Use positive numbers if console
goes off screen, and negative if there is too much border
#overscan_left=16
#overscan_right=16
#overscan_top=16
#overscan_bottom=16
uncomment to force a console size. By default, it will be display's size minus
overscan.
#framebuffer_width=1280
#framebuffer_height=720
uncomment if hdmi display is not detected and composite is being output
#hdmi_force_hotplug=1
uncomment to force a specific HDMI mode (this will force VGA)
#hdmi_group=1
#hdmi_mode=1
uncomment to force a HDMI mode rather than DVI. This can make audio work in
DMT (computer monitor) modes
#hdmi_drive=2
uncomment to increase signal to HDMI, if you have interference, blanking, or
no display
#config_hdmi_boost=4
uncomment for composite PAL
#sdtv_mode=2
#uncomment to overclock the arm. 700 MHz is the default.
#arm_freq=800
Uncomment some or all of these to enable the optional hardware interfaces
#dtparam=i2c_arm=on
#dtparam=i2s=on
#dtparam=spi=on
Uncomment this to enable infrared communication.
#dtoverlay=gpio-ir,gpio_pin=17
#dtoverlay=gpio-ir-tx,gpio_pin=18
Additional overlays and parameters are documented /boot/overlays/README
Enable audio (loads snd_bcm2835)
dtparam=audio=on
Automatically load overlays for detected cameras
camera_auto_detect=1
Automatically load overlays for detected DSI displays
display_auto_detect=1
Enable DRM VC4 V3D driver
dtoverlay=vc4-kms-v3d
max_framebuffers=2
Run in 64-bit mode
arm_64bit=1
Disable compensation for displays with overscan
disable_overscan=1
[cm4]
```

IDENTIFICATION CODE

Identification_code_original.txt

```
import cv2
thres = 0.45 # Threshold to detect object
cap = cv2.VideoCapture(1)
cap.set(3,1280)
cap.set(4,720)
cap.set(10,70)
```

```
classNames= []
classFile = ‘coco.names’
with open(classFile,’rt’) as f:
classNames = f.read().rstrip(‘n’).split(‘n’)
configPath = ‘ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt’
weightsPath = ‘frozen_inference_graph.pb’
net = cv2.dnn_DetectionModel(weightsPath,configPath)
net.setInputSize(320,320)
net.setInputScale(1.0/ 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)
while True:
   success,img = cap.read()
   classIds, confs, bbox = net.detect(img,confThreshold=thres)
   print(classIds,bbox)
   if len(classIds) != 0:
      for classId, confidence,box in zip(classIds.flatten(),confs.flatten(),bbox):

cv2.rectangle(img,box,color=(0,255,0),thickness=2)
         cv2.putText(img,classNames[classId-1].upper(),(box[0]+10,box[1]+30),

cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)

cv2.putText(img,str(round(confidence*100,2)),(box[0]+200,box[1]+30),

cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
   cv2.imshow(“Output”,img)
   cv2.waitKey(1)
```

IDENTIFICATION TXT

```
Identification_code.txt
import cv2
thres = 0.45 # Threshold to detect object
cap = cv2.VideoCapture(1)
cap.set(3,1280)
cap.set(4,720)
cap.set(10,70)
classNames= []
classFile = ‘coco.names’
with open(classFile,’rt’) as f:
classNames = f.read().rstrip(‘n’).split(‘n’)
configPath = ‘ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt’
weightsPath = ‘frozen_inference_graph.pb’

net = cv2.dnn_DetectionModel(weightsPath,configPath)
net.setInputSize(320,320)
net.setInputScale(1.0/ 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)
while True:
   success,img = cap.read()
   classIds, confs, bbox = net.detect(img,confThreshold=thres)
   print(classIds,bbox)
   if len(classIds) != 0:
      for classId, confidence,box in zip(classIds.flatten(),confs.flatten(),bbox):

cv2.rectangle(img,box,color=(0,255,0),thickness=2)
         cv2.putText(img,classNames[classId-1].upper(),(box[0]+10,box[1]+30),

cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)

cv2.putText(img,str(round(confidence*100,2)),(box[0]+200,box[1]+30),

cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
   cv2.imshow(“Output”,img)
   cv2.waitKey(1)
```

NEW.CMD

```
new.cmd
!/bin/bash
sudo apt-get install -y build-essential cmake pkg-config
sudo apt-get install -y libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
sudo apt-get install -y libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get install -y libxvidcore-dev libx264-dev
sudo apt-get install -y libgtk2.0-dev libgtk-3-dev
sudo apt-get install -y libatlas-base-dev gfortran
sudo pip3 install numpy
wget -O opencv.zip https://github.com/opencv/opencv/archive/4.4.0.zip
wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.
```

```
4.0.zip
unzip opencv.zip
unzip opencv_contrib.zip
cd ~/opencv-4.4.0/
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D
OPENCV_EXTRA_MODULES_PATH=~/opencv_c
ontrib-4.4.0/modules \
-D BUILD_EXAMPLES=ON ..
make -j $(nproc)
sudo make install && sudo ldconfig
sudo reboot
```

OBJECT DET 2 PY

```
import cv2
#thres = 0.45 # Threshold to detect object
classNames = []
classFile =
"/home/pi/Desktop/Object_Detection_Files/coco.nam
es"
with open(classFile,"rt") as f:
   classNames = f.read().rstrip("\n").split("\n")
configPath =
"/home/pi/Desktop/Object_Detection_Files/ssd_mobi
lenet_v3_large_coco_2020_01_14.pbtxt"
weightsPath =
"/home/pi/Desktop/Object_Detection_Files/frozen_in
ference_graph.pb"
net =
cv2.dnn_DetectionModel(weightsPath,configPath)
net.setInputSize(320,320)
net.setInputScale(1.0/ 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)
def getObjects(img, thres, nms, draw=True,
objects=[]):
   classIds, confs, bbox =
net.detect(img,confThreshold=thres,nmsThreshold=n
ms)
   #print(classIds,bbox)
   if len(objects) == 0: objects = classNames
   objectInfo =[]
   if len(classIds) != 0:
      for classId, confidence,box in
zip(classIds.flatten(),confs.flatten(),bbox):
         className = classNames[classId - 1]
         if className in objects:
            objectInfo.append([box,className])
            if (draw):

cv2.rectangle(img,box,color=(0,255,0),thickness=2)
               cv2.putText(img,classNames[classId-
1].upper(),(box[0]+10,box[1]+30),

cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)

cv2.putText(img,str(round(confidence*100,2)),(box[
0]+200,box[1]+30),

cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
   return img,objectInfo
if _name_ == "_main_":
   cap = cv2.VideoCapture(0)
   cap.set(3,640)
   cap.set(4,480)
   #cap.set(10,70)
   while True:
      success, img = cap.read()
      result, objectInfo = getObjects(img,0.45,0.2,
objects=['cup'])
      #print(objectInfo)
      cv2.imshow("Output",img) cv2.waitKey(1)
```

OBJECT DET 3 PY

```
import cv2
import time
from gpiozero import AngularServo
servo =AngularServo(18, initial_angle=0,
min_pulse_width=0.0006, max_pulse_width=0.0023)
#thres = 0.45 # Threshold to detect object
classNames = []
classFile =
"/home/pi/Desktop/Object_Detection_Files/coco.nam
es"
with open(classFile,"rt") as f:
   classNames = f.read().rstrip("\n").split("\n")
configPath =
"/home/pi/Desktop/Object_Detection_Files/ssd_mobi
lenet_v3_large_coco_2020_01_14.pbtxt"
weightsPath =
"/home/pi/Desktop/Object_Detection_Files/frozen_in
ference_graph.pb"
net =
```

```
cv2.dnn_DetectionModel(weightsPath,configPath)
net.setInputSize(320,320)
net.setInputScale(1.0/ 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)
def getObjects(img, thres, nms, draw=True, objects=[]):
  classIds, confs, bbox = net.detect(img,confThreshold=thres,nmsThreshold=nms)
  #print(classIds,bbox)
  if len(objects) == 0: objects = classNames
  objectInfo =[]
  if len(classIds) != 0:
    for classId, confidence,box in zip(classIds.flatten(),confs.flatten(),bbox):
      className = classNames[classId - 1]
      if className in objects:
        objectInfo.append([box,className])
      if (draw):

cv2.rectangle(img,box,color=(0,255,0),thickness=2)
            cv2.putText(img,classNames[classId-1].upper(),(box[0]+10,box[1]+30),

cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)

cv2.putText(img,str(round(confidence*100,2)),(box[0]+200,box[1]+30),

cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),
            servo.angle = -90
             time.sleep = 2
             servo.angle = 90
  return img,objectInfo
if _name_ == "_main_":
  cap = cv2.VideoCapture(0)
  cap.set(3,640)
  cap.set(4,480)
  #cap.set(10,70)
  while True:
    success, img = cap.read()
    result, objectInfo = getObjects(img,0.45,0.2, objects=['cup','horse'])
    #print(objectInf)
    cv2.imshow("Output",img)
    cv2.waitKey(1)
```

OBJECT DET PY

```
import cv2
import time
from gpiozero import AngularServo
servo =AngularServo(18, initial_angle=0, min_pulse_width=0.0006, max_pulse_width=0.0023)
#thres = 0.45 # Threshold to detect object

classNames = []
classFile = "/home/pi/Desktop/Object_Detection_Files/coco.names"
with open(classFile,"rt") as f:
  classNames = f.read().rstrip("\n").split("\n")
configPath = "/home/pi/Desktop/Object_Detection_Files/ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt"
weightsPath = "/home/pi/Desktop/Object_Detection_Files/frozen_inference_graph.pb"
net = cv2.dnn_DetectionModel(weightsPath,configPath)
net.setInputSize(320,320)
net.setInputScale(1.0/ 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)
def getObjects(img, thres, nms, draw=True, objects=[]):
  classIds, confs, bbox = net.detect(img,confThreshold=thres,nmsThreshold=nms)
  #print(classIds,bbox)
  if len(objects) == 0: objects = classNames
  objectInfo =[]
  if len(classIds) != 0:
    for classId, confidence,box in zip(classIds.flatten(),confs.flatten(),bbox):
      className = classNames[classId - 1]
      if className in objects:
        objectInfo.append([box,className])
        if (draw):

cv2.rectangle(img,box,color=(0,255,0),thickness=2)
            cv2.putText(img,classNames[classId-1].upper(),(box[0]+10,box[1]+30),

cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)

cv2.putText(img,str(round(confidence*100,2)),(box[0]+200,box[1]+30),
```

```
cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)

                servo.angle = -90
                time.sleep = 2
                servo.angle = 90
    return img,objectInfo
if _name_ == "_main_":
    cap = cv2.VideoCapture(0)
    cap.set(3,640)
    cap.set(4,480)
    #cap.set(10,70)
    while True:
        success, img = cap.read()
        result, objectInfo = getObjects(img,0.45,0.2,
objects=['cup','horse'])
        #print(objectInfo)
        cv2.imshow("Output",img)
        cv2.waitKey(1)
```

5. ACKNOWLEDGMENT

This research is funded and supported by Biomedical engineering department, Military Institute of Science and Technology.

REFERENCES

[1] Bhuvaneswary, N., Pravallika, S., Jayapriya, V., & Himabindu, K. (2021). Night surveillance military spy robot using Raspberry Pi. Annals of the Romanian Society for Cell Biology, 25(5), 5740–5747.

[2] Kaumalee Bogahawatte and Shalinda Adikari, “Intelligent Criminal Identification System”, Proceedings of 8th IEEE International Conference on Computer Science and Education, pp.633-638, 2013.

[3] K. Gopalakrishnan, S. Thiruvenkatasamy, E.Prabhakar, and R. Aarthi “Night Vision Patrolling Rover Navigation System for Women’s Safety Using Machine Learning”

[4] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng Yang Fu, and Alexander C Berg. SSD: Single-shot multibox detector. In European Conference on Computer Vision, pages 21–37. Springer, 2016.

[5] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. arXiv preprint arXiv:1904.07850, 2019.

[6] Dinu Coltuc, Philippe Bolon, and J-M Chassery. Exact histogram specification. IEEE Transactions on Image Processing, 15(5):1143–1152, 2006. (11)

[7] Haidi Ibrahim and Nicholas Sia Pik Kong. Brightness preserving dynamic histogram equalization for image contrast enhancement. IEEE Transactions on Consumer Electronics, 53(4):1752–1758, 2007. (26)

[8] Chulwoo Lee, Chul Lee, and Chang-Su Kim. Contrast enhancement based on layered difference representation of 2d histograms. IEEE transactions on image processing, 22(12):5372–5384, 2013. (35)

[9] Xueyang Fu, Delu Zeng, Yue Huang, Xiao-Ping Zhang, and Xinghao Ding. A weighted variational model for simultaneous reflectance and illumination estimation. In IEEE Conference on Computer Vision and Pattern Recognition, pages 2782–2790, 2016. (15)

[10] Xiaojie Guo, Yu Li, and Haibin Ling. Lime: Low-light image enhancement via illumination map estimation. IEEE Transactions on Image Processing, 26(2):982–993,2016. (21)

[11] Yuen Peng Loh and Chee Seng Chan. Getting to know low-light images with the exclusively dark dataset. Computer Vision and Image Understanding, 178:30 42, 2019. (44)

[12] Yukihiro Sasagawa and Hajime Nagahara. Yolo in the dark-domain adaptation method for merging multiple models. In European Conference on Computer Vision, pages 345–359. Springer, 2020. (53)

[13] R.T. Collins, A.J. Lipton, T. Kanade, et al., A system for video surveillance and monitoring, Technical Report, Carnegie Mellon University, 2000.

[14] R. Cucchiara, M. Piccardi, Vehicle detection under day and night illumination, Proceedings of the Third International Symposium on Intelligent Industrial Automation and Soft Computing, 1999.

[15] Song, R., Li, T., Li, T. (2023). Ship detection in haze and low-light remote sensing images via colour balance and DCNN. Applied Ocean

Research, 139: 103702. https://doi.org/10.1016/j.apor.2023.103702

[16] Peng, D., Ding, W., Zhen, T. (2024). A novel low-light object detection method based on the YOLOv5 fusion feature enhancement. Scientific Reports, 14(1): 4486. https://doi.org/10.1038/s41598-024-54428-8

[17] Wang, J., Yang, P., Liu, Y., Shang, D., Hui, X., Song, J., Chen, X. (2023). Research on improved YOLOv5 for low-light environment object detection. Electronics, 12(14): 3089. https://doi.org/10.3390/electronics12143089

[18] Rahim, A., Maqbool, A., Rana, T. (2021). Monitoring social distancing under various low-light conditions with deep learning and a single motionless time-of-flight camera. Plos One, 16(2): e0247440. https://doi.org/10.1371/journal.pone.0247440

[19] Zhao, D.; Shao, F.; Zhang, S.; Yang, L.; Zhang, H.; Liu, S.; Liu, Q. Advanced Object Detection in Low-Light Conditions: Enhancements to YOLOv7 Framework. Remote Sens. 2024, 16, 4493. https://doi.org/10.3390/rs16234493

[20] A. Nazib, C.-M. Oh, and C. W. Lee, "Object Detection and Tracking in Night Time Video Surveillance, " 2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Jeju, Korea, Oct. 31–Nov. 2, 2013, pp. 191–192. doi: 10.1109/URAI.2013.6677426

[21] Namana, M. S. K., & Kumar, B. U. (2024). An efficient and robust night-time surveillance object detection system using YOLOv8 and high-performance computing. International Journal of Safety and Security Engineering, 14(6), 1115–1123. https://doi.org/10.1828