ECE 445

Senior Design Laboratory

Final Report

Raspberry Pi-Based Real-Time Traffic Monitoring System

Team #18

Yiyang Cheng (yiyang28@illinois.edu)
Yucong Gao (yucongg3@illinois.edu)
Ding Jiang (dingj3@illinois.edu)
Zetong Lang (zetong3@illinois.edu)

TA: Shuang Wu

May 15, 2026

Abstract

Urban traffic congestion remains a persistent problem for city infrastructure management because traffic density changes continuously and often requires timely observation. Traditional monitoring approaches may depend on manual observation, fixed infrastructure, or expensive dedicated sensing hardware, which limits their flexibility and makes low-cost deployment difficult. This project presents a Raspberry Pi-based real-time traffic monitoring system designed to provide an affordable and scalable method for observing traffic density, counting vehicles, and reporting congestion conditions through a live dashboard.

The system consists of four main functional parts: image acquisition, edge-based vehicle detection, traffic data processing, and dashboard-based visualization. A camera module connected to a Raspberry Pi captures continuous video of a road scene. The Raspberry Pi processes the video stream locally using computer vision methods to detect vehicles, estimate the number of vehicles in the monitored region, and generate traffic-density information. The processed data are then transmitted to a dashboard that displays relevant traffic conditions and produces congestion alerts when the detected density exceeds a predefined threshold. This structure reduces the need for expensive external processing hardware and allows the system to operate as an independent monitoring node.

The hardware design focuses on stable video capture and reliable outdoor operation. The Raspberry Pi, camera module, and power supply are integrated into a compact monitoring unit. A weather-resistant enclosure and adjustable mounting structure are used to protect the electronics from environmental conditions such as rain, dust, and temperature variation while maintaining a stable camera angle. These hardware choices support continuous operation and help preserve detection reliability during long-term monitoring.

The software design focuses on real-time performance and usable traffic information. The vehicle detection and counting module converts camera input into traffic-density data, while the dashboard organizes this data into a clear visual interface for users. The alert function provides an automatic response mechanism when congestion reaches a selected level. Together, these features allow traffic managers or users to observe current congestion conditions without manual counting.

This report describes the motivation, design procedure, system architecture, implementation details, verification process, and cost considerations of the traffic monitoring system. The completed design demonstrates how low-cost embedded hardware, computer vision, and web-based visualization can be combined to create a practical traffic monitoring platform. The project also identifies important engineering tradeoffs among processing speed, detection accuracy, environmental protection, camera placement, and dashboard latency. Overall, the system provides a foundation for a deployable, low-cost traffic monitoring solution that can be expanded in future work through improved detection models, multiple camera nodes, and more advanced traffic-flow analysis.

Contents

1 Introduction

Urban traffic congestion is a continuing challenge for city infrastructure management because vehicle density can change rapidly throughout the day. Traditional traffic monitoring methods often depend on manual observation, fixed traffic infrastructure, or expensive dedicated sensing equipment. These methods may be difficult to deploy widely and may not provide timely information for traffic managers. As a result, there is a need for a low-cost and scalable monitoring system that can observe traffic conditions continuously, count vehicles, estimate congestion levels, and report useful information with minimal delay.

This project develops a Raspberry Pi-based real-time traffic monitoring system that uses a camera module to capture traffic video and analyze vehicle density. The system is intended to function as a compact edge-computing node. Instead of sending raw video to a remote server for all processing, the Raspberry Pi performs local image processing to detect and count vehicles in the camera view. The processed traffic data are then sent to a live dashboard, where users can view vehicle count, traffic density, and congestion alerts.

The main objective of the project is to provide an affordable traffic monitoring platform that can be deployed in outdoor environments. The design combines hardware, software, and mechanical protection into one integrated system. The hardware portion includes the Raspberry Pi, camera module, power supply, and supporting connections. The software portion includes vehicle detection, vehicle counting, data processing, dashboard visualization, and congestion alert generation. The mechanical portion includes a weather-resistant enclosure and adjustable camera mounting structure to protect the electronics and maintain a stable camera angle.

1.1 Problem Statement

Traffic managers need timely and reliable information about congestion levels in order to understand road conditions and support traffic-flow decisions. However, manual counting is labor-intensive and cannot provide continuous monitoring. Dedicated traffic sensors and commercial traffic camera systems can be expensive, especially when many locations must be monitored. A low-cost embedded monitoring system can help reduce this limitation by collecting local traffic data and reporting congestion conditions in real time.

The problem addressed by this project is the lack of an accessible and affordable system for continuous traffic-density monitoring. The system must be able to capture road images, identify vehicles, estimate traffic density, and report the results through a user interface. It must also be able to operate outdoors, where rain, dust, temperature changes, and camera movement can reduce reliability.

1.2 Solution Overview

The proposed solution is a Raspberry Pi-based traffic monitoring system with a camera and a live dashboard. The camera captures video frames from a selected road region. The Raspberry Pi processes the frames using computer vision techniques to detect vehicles and

estimate the number of vehicles in the monitored area. The traffic data are then transferred to a dashboard that displays current traffic conditions and produces congestion alerts when the vehicle density exceeds a predefined threshold.

The system is divided into four major functional blocks. The first block is image acquisition, which is responsible for capturing traffic video through the camera module. The second block is edge processing, which performs vehicle detection and counting on the Raspberry Pi. The third block is data processing, which converts vehicle count into useful traffic-density information. The fourth block is dashboard visualization, which presents the processed data and alerts to the user.

1.3 High-Level Requirements

The system must satisfy three main requirements. First, the dashboard must collect enough traffic data to estimate the degree of congestion and generate alerts when congestion reaches a selected threshold. Second, the physical structure must support continuous operation under outdoor conditions such as rain, snow, dust, and temperature variation. Third, the dashboard must display relevant traffic data with low latency so that the reported information remains useful for real-time monitoring.

These requirements guide the design choices throughout the project. The Raspberry Pi was selected because it provides a low-cost computing platform with sufficient capability for camera input and basic computer vision processing. The camera module was selected to provide continuous visual data for vehicle detection. The enclosure and mounting structure were included to improve long-term reliability and reduce the effect of environmental conditions on the electronics and camera position.

1.4 Team Responsibilities

The project work is divided among four team members. Ding Jiang is responsible for developing the image processing algorithm on the Raspberry Pi, including vehicle detection, vehicle counting, and processing optimization. Yucong Gao is responsible for building the real-time dashboard, including data visualization, traffic-density display, and congestion alert functions. Yiyang Cheng is responsible for the hardware setup of the Raspberry Pi traffic monitoring system, including Raspberry Pi configuration, camera module setup, power supply integration, and hardware-level reliability. Zetong Lang is responsible for designing the outdoor enclosure and adjustable mounting structure, including weather protection and stable camera positioning.

Together, these responsibilities cover the full system from data capture to user-facing traffic visualization. The rest of this report describes the design procedure, implementation details, verification methods, cost analysis, and final conclusions of the traffic monitoring system.

2 Design

2.1 System Architecture

The traffic monitoring system is designed as a Raspberry Pi-based edge-computing platform for real-time vehicle detection, vehicle counting, and congestion classification. The system uses a camera to capture a live road scene and processes the video locally on the Raspberry Pi. The output is an annotated video frame that shows detected vehicles, vehicle type, confidence score, frame rate, vehicle count, and congestion level.

The system is divided into five major functional blocks: video input, vehicle detection, vehicle counting, congestion classification, and visual rendering. The video input block reads frames from either a camera or a video file. The vehicle detection block applies a YOLO-based object detection model to locate vehicles in each frame. The counting block summarizes the detected vehicles by class. The congestion classification block converts the total vehicle count into low, medium, or high congestion status. The rendering block overlays bounding boxes, labels, statistics, and the congestion bar onto the displayed frame.

This architecture was selected because it keeps the system simple, modular, and suitable for a first working prototype. Each block has a clear function, which makes the system easier to test and debug. The design also allows individual modules to be improved later without rewriting the full program.

2.2 Hardware Design

The hardware portion of the system consists of a Raspberry Pi computing unit, a camera module, a power supply, and an outdoor enclosure. The Raspberry Pi is used as the main processing device because it supports Python-based computer vision, camera input, and low-cost deployment. The camera module is used to capture the monitored road area and provide continuous video frames for vehicle detection.

The power supply is selected to provide stable operation for the Raspberry Pi and camera during the demonstration. Since the system is intended for traffic monitoring, stable power is important because unexpected power loss would interrupt data collection and prevent continuous observation.

The outdoor enclosure is used to protect the electronic components from environmental conditions. The enclosure is designed to reduce exposure to rain, dust, and temperature variation while keeping the camera view unobstructed. An adjustable camera mounting structure is also included so that the camera angle can be tuned during installation. This adjustment is important because the detection accuracy depends strongly on whether vehicles are clearly visible in the camera frame.

2.3 Mechanical and Enclosure Design

The mechanical design of the traffic monitoring system focuses on protecting the electronics, supporting stable camera placement, and allowing the system to operate as a compact

outdoor monitoring unit. The enclosure is designed to hold the Raspberry Pi, camera module, and power-related components while reducing direct exposure to rain, dust, and physical disturbance. Since the camera angle directly affects vehicle detection performance, the mounting structure is also designed to keep the camera stable during operation.

Figure 1 shows the overall computer-aided design (CAD) model of the traffic monitoring unit. The design integrates the main enclosure, camera position, and support structure into one assembly. This layout allows the system to be installed near a road segment while keeping the internal electronic components protected.

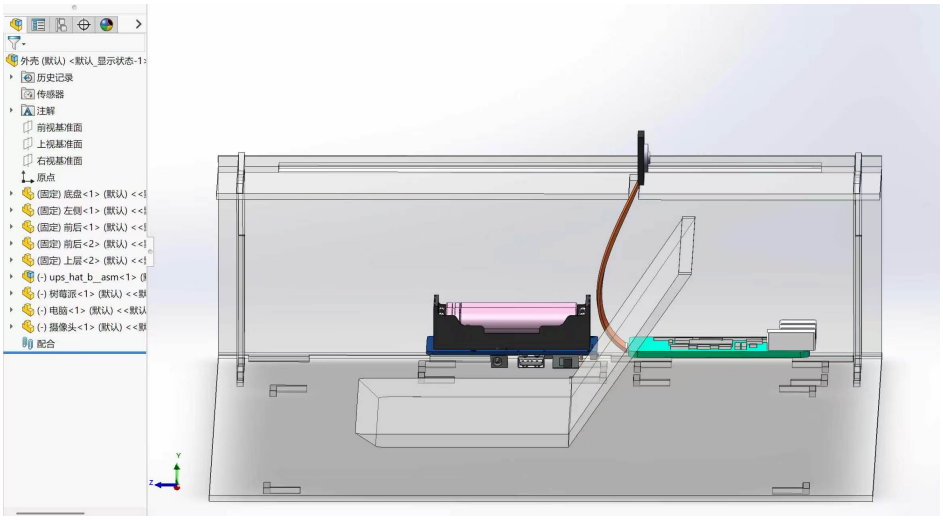

Figure 1: Overall CAD model of the traffic monitoring system enclosure and mounting structure.

Figure 2 shows the camera mounting design. The camera is positioned so that it can face the monitored road region and capture vehicles within the field of view. A stable camera mount is important because changes in camera angle can affect detection accuracy, vehicle count, and congestion classification.

Figure 3 shows the CAD model related to the power or battery placement. The power section is arranged inside the enclosure so that it can support continuous operation of the Raspberry Pi and camera module. Keeping the power components inside the protected structure helps improve reliability during outdoor use.

2.4 Software Design

The software is implemented in Python and uses OpenCV for video input and display. The detection module uses a YOLO model to identify vehicles in each frame. The main program accepts command-line arguments so that the system can be configured for different input sources, camera backends, confidence thresholds, image sizes, display modes, and congestion thresholds.

The input source can be either a live camera or a recorded traffic video. This makes the system easier to test because recorded videos can be used during software development,

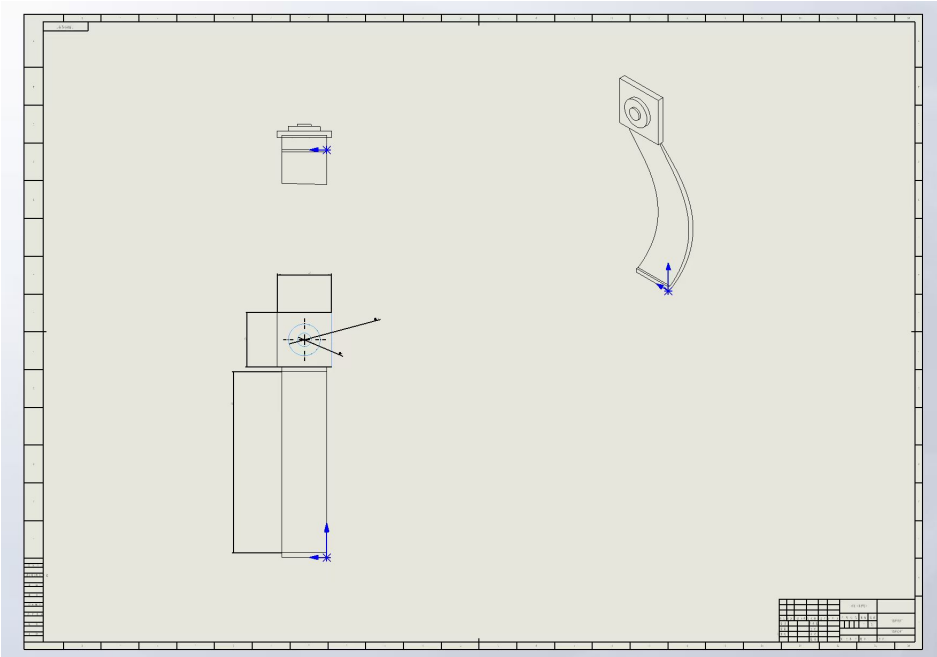

Figure 2: CAD model of the camera mounting section.

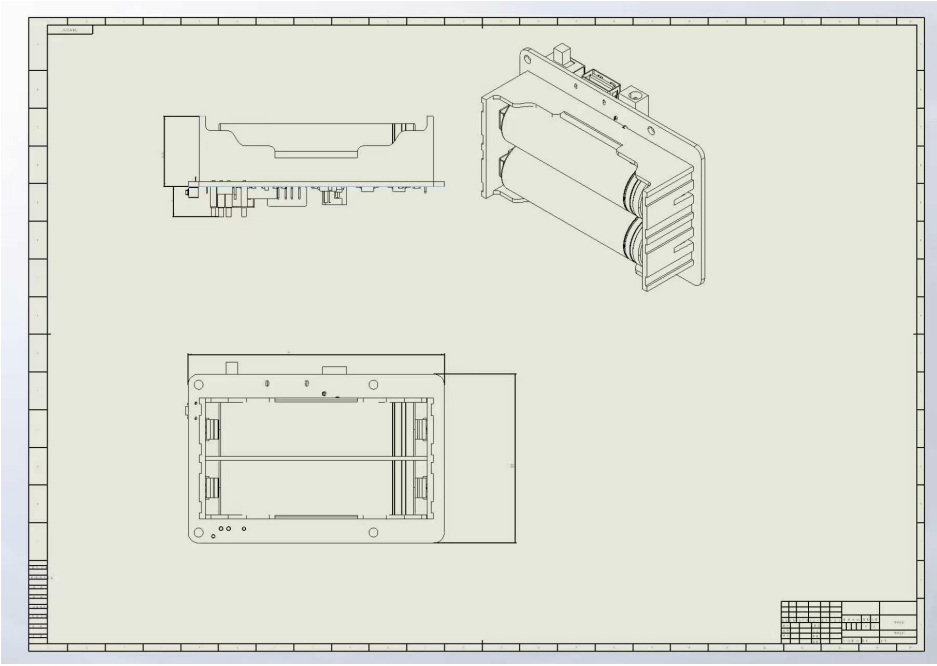

Figure 3: CAD model of the internal power component placement.

while the live camera can be used for final demonstration. The program also supports both OpenCV camera input and Raspberry Pi camera input, which gives flexibility depending on the camera hardware used.

The detection model used in the prototype is yolov8n.pt. This model was selected because it is lightweight compared with larger YOLO models and is more suitable for real-time or near-real-time processing on limited hardware. The confidence threshold is set to 0.25 by default. Detections below this threshold are filtered out, which helps reduce false detections while still allowing the system to recognize vehicles in a complex road scene.

2.5 Vehicle Detection and Counting

The vehicle detection module processes each frame and returns a list of detected objects. The system only keeps traffic-related classes, including cars, buses, trucks, motorcycles, and bicycles. Other object classes are ignored because they are not needed for the traffic-density calculation.

Each valid detection includes a bounding box, vehicle class name, and confidence score. The bounding box defines the location of the detected vehicle in the frame. The class name identifies the vehicle type, and the confidence score indicates how confident the model is in the detection result.

After detection, the traffic counting module counts the number of detected vehicles in the current frame. The system reports both the total number of vehicles and the count for each vehicle type. This allows the system to monitor different vehicle categories in real time instead of only reporting one total count. The current prototype defines traffic density as the number of vehicles visible in the current frame, rather than cumulative vehicles passing through a line.

2.6 Congestion Classification

The congestion level is determined using the total number of detected vehicles in the current frame. The system classifies the road condition into low, medium, and high congestion levels. In the current implementation, the medium congestion threshold is 3 vehicles, and the high congestion threshold is 6 vehicles.

If the detected vehicle count is below the medium threshold, the road condition is classified as low congestion. If the count is greater than or equal to the medium threshold but below the high threshold, the condition is classified as medium congestion. If the count is greater than or equal to the high threshold, the condition is classified as high congestion.

This threshold-based method was chosen because it is simple, interpretable, and easy to adjust. The threshold values can be modified depending on the camera angle, road width, monitored area, and expected traffic density. For example, a wide multi-lane road may require higher thresholds than a narrow road segment.

2.7 Visual Output

The visual output is designed to make the system result easy to understand during operation. The system draws bounding boxes around detected vehicles and labels each box with the vehicle type and confidence score. A statistics panel displays the number of vehicles in the frame, the count for each vehicle class, and the current frame rate. When congestion visualization is enabled, the system also displays a congestion label and a colored congestion bar.

The system can monitor the number of different types of vehicles in real time, including cars, buses, and trucks. Based on the detected vehicle count and traffic density in the camera frame, the system classifies the road condition into low, medium, and high congestion levels. This allows the output display to provide both detailed vehicle-count information and an overall congestion status for the monitored road segment.

Figure 4 shows the system output under a low-congestion condition. In this case, only a small number of vehicles are detected in the frame, and the system labels the traffic condition as low congestion.

Figure 4: Vehicle detection output under low congestion.

Figure 5 shows the system output under a medium-congestion condition. Compared with the low-congestion case, more vehicles are detected in the frame, and the congestion indicator increases to the medium level.

Figure 6 shows the system output under a high-congestion condition. In this case, the number of detected vehicles is higher, and the system marks the road condition as high congestion. This result demonstrates that the system can distinguish between different levels of traffic density and report the congestion state visually.

Figure 5: Vehicle detection output under medium congestion.

Figure 6: Vehicle detection output under high congestion.

2.8 Runtime Operation

During runtime, the program continuously reads frames from the selected video source. For each frame, the detector performs inference, the counter summarizes the detected vehicle classes, and the renderer draws the annotated result. The program also calculates the frame rate using the time difference between consecutive frames. This frame-rate value is displayed so that the user can evaluate whether the system is operating close to real time.

The program prints terminal statistics at a fixed interval. These statistics include the total vehicle count, the count for each vehicle class, and the current frame rate. This terminal output provides a simple text-based record of system behavior during testing.

The program also includes basic error handling. If a frame cannot be read from the camera or video source, the program increments a failure counter. If the number of consecutive read failures reaches the maximum allowed value, the program exits and reports an error. This prevents the system from running indefinitely when the camera is disconnected or the video source is unavailable.

2.9 Design Tradeoffs

The main design tradeoff is between detection accuracy and real-time performance. A larger object detection model may improve accuracy, but it would require more computation and reduce frame rate on the Raspberry Pi. A smaller model is faster and more suitable for real-time demonstration, but it may miss some vehicles or produce lower-confidence detections. The prototype uses a lightweight YOLO model to balance these two requirements.

Another tradeoff is the selection of congestion thresholds. Lower thresholds make the system more sensitive to traffic buildup but can also cause normal traffic to be classified as congestion. Higher thresholds reduce false congestion alerts but may delay the detection of actual congestion. To address this issue, the medium and high congestion thresholds are configurable through command-line arguments.

The system also trades detailed traffic-flow measurement for simplicity. The current prototype counts vehicles visible in the frame instead of tracking vehicles across time or counting vehicles that cross a virtual line. This approach is easier to implement and sufficient for estimating traffic density, but future versions could add tracking and line-crossing logic to measure traffic flow rate more accurately.

3 Verification

3.1 Verification Overview

The verification process was used to confirm that the traffic monitoring system satisfies the major functional requirements of the project. The system was tested at both the module level and the integrated system level. The main functions verified were video input, vehicle detection, vehicle counting, congestion classification, visual output, runtime stability, and basic fault handling.

The project success criteria require the system to collect enough traffic data to determine congestion level, display relevant traffic information with low latency, and support continuous operation in an outdoor monitoring structure. Since the system is a prototype, verification focused on whether the system could reliably demonstrate real-time vehicle detection and congestion classification under different traffic-density conditions.

3.2 Video Input Verification

The video input module was verified by running the program with a traffic video source. The system successfully opened the source, read frames continuously, and displayed the road scene in an OpenCV window. This confirmed that the input pipeline could provide image frames to the detection module without requiring manual frame loading.

The system also includes basic failure handling for video input. If a frame cannot be read, the program increases a read-failure counter. If the number of consecutive read failures exceeds the allowed limit, the program exits and reports an error. This behavior prevents the system from continuing to run without valid image input when the camera is disconnected or the video source becomes unavailable.

3.3 Vehicle Detection Verification

Vehicle detection was verified using representative road-scene outputs. The detection model identified traffic-related vehicle classes, including cars, buses, and trucks. Each detected vehicle was shown with a bounding box, class label, and confidence score. This visual result confirmed that the detection module could locate vehicles in a multi-lane traffic scene.

The detection output also showed that the system can process multiple vehicles within a single frame. This is necessary for traffic monitoring because congestion estimation depends on detecting several vehicles at once rather than identifying only one object. The result demonstrates that the detection module provides enough information for downstream vehicle counting and congestion classification.

3.4 Vehicle Counting Verification

The vehicle counting module was verified by checking whether the displayed vehicle counts matched the detected objects shown in the annotated frame. The system reports both the

total number of vehicles in the frame and the count for each vehicle type. These vehicle types include cars, buses, trucks, motorcycles, and bicycles.

The counting method is based on the current frame. Therefore, the reported value represents the number of vehicles currently visible in the monitored scene, rather than the cumulative number of vehicles that have passed through the road. This definition is appropriate for the prototype because the main goal is to estimate traffic density and congestion level from the visible traffic condition.

3.5 Congestion Classification Verification

Congestion classification was verified using three representative traffic-density outputs. The system classifies the road condition into low, medium, and high congestion levels according to the detected vehicle count and the configured threshold values.

Figure 4 demonstrates the low-congestion output. In this condition, the system detects a smaller number of vehicles and displays a low congestion state. Figure 5 demonstrates the medium-congestion output, where more vehicles are detected and the congestion indicator increases to the medium level. Figure 6 demonstrates the high-congestion output, where the number of detected vehicles is larger and the congestion state is marked as high.

These results verify that the system can distinguish among different traffic-density conditions and present the congestion status in a clear visual format. The congestion threshold values can also be adjusted for different road widths, camera angles, and monitored regions.

3.6 Real-Time Display Verification

The real-time display was verified by observing the annotated output during program execution. The display includes bounding boxes, vehicle type labels, confidence values, total vehicle count, class-level counts, frame rate, and congestion level. The frame-rate display allows the user to judge whether the system is operating close to real time.

The system also prints traffic statistics to the terminal at a fixed interval. This provides a text-based output in addition to the visual display and helps confirm that the internal vehicle-counting result is being updated during runtime. During testing, the output updated continuously and provided readable traffic information without obvious freezing in the demonstration video.

3.7 Outdoor Structure Verification

The outdoor structure was verified by reviewing the enclosure and mounting design. The enclosure is intended to protect the Raspberry Pi, camera connection, and power components from rain, dust, and general outdoor exposure. The mounting structure allows the camera angle to be adjusted so that the road area remains visible.

This verification confirms that the physical design supports the goal of outdoor traffic monitoring. However, long-term outdoor testing under heavy rain, snow, and large temperature

variation was not fully completed within the prototype schedule. Future testing should include extended outdoor operation to evaluate waterproofing, heat dissipation, and camera stability over time.

3.8 Fault Handling Verification

The software includes basic fault handling to improve runtime reliability. When the program fails to read frames from the selected video source, it does not immediately crash. Instead, it counts the number of consecutive read failures. If the number of failures reaches the maximum allowed value, the program exits and reports the failure.

This behavior was included to prevent the system from remaining in an invalid state. For example, if the camera is disconnected or the video file cannot provide additional frames, the program can stop safely instead of continuing with empty input. This is important for a monitoring system because invalid input should be reported clearly rather than hidden from the user.

3.9 Verification Limitations

The verification results show that the prototype can perform real-time vehicle detection, vehicle counting, and congestion classification. However, several limitations remain. First, the current system estimates congestion from the number of vehicles visible in the frame, rather than tracking vehicles across time or measuring vehicle flow rate. Second, detection accuracy depends on camera angle, lighting, occlusion, and the distance between vehicles and the camera. Third, the prototype uses a lightweight detection model to maintain speed, which may reduce accuracy compared with larger models.

Despite these limitations, the verification process confirms that the main project functions were successfully demonstrated. The system can capture a traffic scene, detect vehicles, count vehicle classes, classify congestion level, and display the result in real time.

4 Cost Analysis

The cost analysis of this project focuses on the actual direct expenses required to build and test the Raspberry Pi-based road traffic detection MVP. Since this project is developed as an academic prototype, labor cost is not included in the total cost estimation. The software tools used in this project, including Python, OpenCV, and the YOLO-based object detection framework, are open-source or free for academic and prototype development [1], [2]. Therefore, the total project cost mainly comes from the Raspberry Pi platform, the power supply board with cables, and the acrylic materials purchased for the physical enclosure.

Table 1: Actual Cost of the Traffic Detection MVP

Item	Quantity	Unit Cost (RMB)	Total Cost (RMB)
Raspberry Pi Kit	1	2079.00	2079.00
Power Supply Board and Cables	1 set	178.84	178.84
3 mm Acrylic Plate, 70 mm × 70 mm	2	44.46	88.92
5 mm Acrylic Plate, 70 mm × 70 mm	2	72.63	145.26
Total Actual Cost			2492.02

As shown in the cost table, the total actual cost of the project is 2492.02 RMB. The largest expense is the Raspberry Pi kit, which cost 2079.00 RMB and served as the main computing platform of the traffic detection system. It was responsible for running the detection program, processing camera input, counting detected vehicles, and displaying the visualization results. Although this cost is higher than a basic Raspberry Pi board alone, it is reasonable for a complete project kit because it may include necessary accessories for system setup and operation.

The power supply board and cables cost 178.84 RMB. This part was necessary for providing stable power connection and supporting reliable system operation during testing. Since the Raspberry Pi-based prototype needs stable power input to avoid unexpected shutdowns or unstable camera processing, the power supply board and cable set is included as an important direct hardware cost.

The enclosure cost mainly came from acrylic plates. Two 3 mm acrylic plates with dimensions of 70 mm × 70 mm were purchased for 88.92 RMB and were used to build the protective shell of the prototype. In addition, two 5 mm acrylic plates with the same dimensions were purchased for 145.26 RMB. However, during the assembly and design verification process, the 5 mm plates were found to be unsuitable for the final enclosure design. Although they were not used in the final prototype, they are still included in the actual cost because they were purchased as part of the design iteration and prototyping process.

Compared with commercial traffic monitoring systems, the total cost of this MVP is still relatively low. Traditional traffic monitoring solutions may require industrial cameras, dedicated computing units, communication modules, cloud services, and professional installation, which can significantly increase the total cost. In contrast, this project uses a Raspberry

Pi-based edge computing platform and open-source software to achieve basic vehicle detection, traffic counting, and congestion visualization. This makes the system more suitable for small-scale demonstration, education, and prototype validation.

Overall, the project cost is acceptable for an academic MVP. The final cost mainly resulted from the purchase of the Raspberry Pi kit, the power supply board and cables, and the trial-and-error process during enclosure fabrication. In future versions, the cost could be reduced by selecting a lower-cost Raspberry Pi package, reusing existing accessories, and confirming enclosure thickness before purchasing materials. Therefore, the proposed system still demonstrates a practical balance among cost, functionality, and prototype feasibility.

5 Conclusion

This project successfully developed a Raspberry Pi-based real-time traffic monitoring system for vehicle detection, vehicle counting, and congestion classification. The system uses a camera to capture a road scene, applies a lightweight YOLO-based detection model to identify traffic-related vehicles, counts the detected vehicles by class, and classifies the current road condition into low, medium, or high congestion. The final output is displayed as an annotated video frame with bounding boxes, confidence values, vehicle statistics, frame rate, and congestion status.

The completed prototype demonstrates the core idea of an edge-computing traffic monitoring device. Instead of sending raw video data to a remote server, the Raspberry Pi processes the traffic scene locally and produces a direct traffic-density result. This design reduces system complexity and makes the prototype suitable for low-cost traffic observation in small-scale applications such as campus roads, parking-lot entrances, or local intersections.

The verification results show that the system can read a video source, detect multiple vehicles in a road scene, count different vehicle types, update the visual output in real time, and classify congestion level based on configurable thresholds. The mechanical design also provides a basic enclosure and mounting concept to support outdoor deployment. Together, the software and hardware designs satisfy the main project objective of building a functional traffic-monitoring prototype.

Several limitations remain in the current version. First, the system estimates congestion using the number of vehicles visible in the current frame rather than tracking vehicles over time. This means that the prototype measures traffic density but does not yet calculate traffic flow rate or vehicle speed. Second, the detection accuracy depends on camera angle, lighting condition, occlusion, and vehicle distance. Third, the lightweight YOLO model improves runtime speed but may miss small or partially blocked vehicles compared with larger models. Finally, the outdoor enclosure still requires longer-term testing under rain, dust, heat, and vibration before real deployment.

Future improvements should focus on adding multi-object tracking, virtual line-crossing counting, vehicle speed estimation, and automatic threshold calibration. The system could also be improved by using a more powerful edge AI accelerator, a weatherproof industrial camera, and a more robust outdoor enclosure. With these improvements, the prototype could be extended from a demonstration system into a more practical low-cost intelligent traffic monitoring platform.

References

[1] G. Bradski, "The opencv library," Dr. Dobb's Journal of Software Tools, 2000.

[2] G. Jocher, A. Chaurasia, and J. Qiu, Ultralytics yolov8, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

Appendix A Software Source Code

This appendix includes the main Python program used for the Raspberry Pi-based traffic monitoring prototype. The program parses command-line arguments, initializes the detector, reads frames from the selected video source, performs vehicle detection and counting, renders the annotated output, prints traffic statistics, and handles camera read failures.

```
import argparse
import time
from typing import Union

import cv2

from traffic_demo import CongestionVisualizer, Detector, Renderer, TrafficCounter, VideoSource


def str2bool(value: str) -> bool:
    value_lower = value.lower()
    if value_lower in {"true", "1", "yes", "y"}:
        return True
    if value_lower in {"false", "0", "no", "n"}:
        return False
    raise argparse.ArgumentTypeError("show-fps must be true/false")


def parse_source(source: str) -> Union[int, str]:
    return int(source) if source.isdigit() else source


def build_parser() -> argparse.ArgumentParser:
    parser = argparse.ArgumentParser(description="Raspberry Pi road traffic detection MVP")
    parser.add_argument("--source", default="0", help="Camera index or path to video file")
    parser.add_argument(
        "--camera-backend",
        default="auto",
        choices=["auto", "opencv", "picamera2"],
        help="Video source backend. auto prefers picamera2 for integer camera indices.",
    )
    parser.add_argument("--model", default="yolov8n.pt", help="YOLO model path")
    parser.add_argument("--conf", type=float, default=0.25, help="Confidence threshold")
    parser.add_argument("--imgsz", type=int, default=640, help="Inference image size")
    parser.add_argument("--show-fps", type=str2bool, default=True, help="Whether to draw FPS")
    parser.add_argument(
        "--show-congestion",
        type=str2bool,
        default=False,
        help="Whether to draw congestion level bar and vehicle-area overlay",
    )
    parser.add_argument(
        "--congestion-medium",
        type=int,
        default=3,
        help="Vehicle count threshold for medium congestion",
```

```
    )
    parser.add_argument(
        "--congestion-high",
        type=int,
        default=6,
        help="Vehicle count threshold for high congestion",
    )
    parser.add_argument(
        "--display",
        type=str2bool,
        default=True,
        help="Whether to open an OpenCV window (set false for SSH/headless runs).",
    )
    parser.add_argument(
        "--print-interval",
        type=float,
        default=1.0,
        help="Seconds between terminal statistics output",
    )
    parser.add_argument(
        "--max-read-fails",
        type=int,
        default=30,
        help="Maximum consecutive frame read failures before exit",
    )
    return parser


def run(args: argparse.Namespace) -> int:
    source = parse_source(args.source)
    detector = Detector(model_path=args.model, conf=args.conf, imgsz=args.imgsz)
    counter = TrafficCounter()
    congestion_visualizer = (
        CongestionVisualizer(
            medium_threshold=args.congestion_medium,
            high_threshold=args.congestion_high,
        )
        if args.show_congestion
        else None
    )
    renderer = Renderer(
        show_fps=args.show_fps,
        congestion_visualizer=congestion_visualizer,
    )
    video_source = VideoSource(source, backend=args.camera_backend)

    window_name = "Road Traffic Detection MVP"
    if args.display:
        cv2.namedWindow(window_name, cv2.WINDOW_NORMAL)

    print(
        f"[INFO] source={args.source} backend={video_source.backend_name} "
        f"display={'on' if args.display else 'off'} "
        f"congestion={'on' if args.show_congestion else 'off'}"
```

```
    )

    last_print = time.time()
    last_frame_ts = time.time()
    read_fail_count = 0

    try:
        while True:
            ret, frame = video_source.read()
            if not ret:
                read_fail_count += 1
                if read_fail_count >= args.max_read_fails:
                    print(
                        f"[ERROR] Failed to read frame {read_fail_count} times in a row. "
                        "Exiting."
                    )
                    return 1
                continue

            read_fail_count = 0
            now = time.time()
            delta = now - last_frame_ts
            fps = (1.0 / delta) if delta > 0 else 0.0
            last_frame_ts = now

            detections = detector.infer(frame)
            summary = counter.count(detections)
            annotated = renderer.draw(frame, detections, summary, fps)

            if args.display:
                cv2.imshow(window_name, annotated)

            if now - last_print >= args.print_interval:
                by_class = ", ".join(
                    f"{k}:{summary.by_class.get(k, 0)}"
                    for k in ["car", "bus", "truck", "motorcycle", "bicycle"]
                )
                print(f"[STATS] total={summary.total} | {by_class} | fps={fps:.1f}")
                last_print = now

            if args.display:
                key = cv2.waitKey(1) & 0xFF
                if key in {27, ord("q")}:
                    return 0
    finally:
        video_source.release()
        if args.display:
            cv2.destroyAllWindows()


def main() -> int:
    parser = build_parser()
    args = parser.parse_args()
    try:
```

```
        return run(args)
    except RuntimeError as exc:
        print(f"[ERROR] {exc}")
        return 1
    except Exception as exc:
        print(
            "[ERROR] Unexpected failure. "
            "Check camera source, model path, and ultralytics/opencv installation."
        )
        print(f"[DETAIL] {exc}")
        return 1


if __name__ == "__main__":
    raise SystemExit(main())
```

Listing 1: Main Python program for the traffic monitoring system.