CarPi

by Nicole Alvarado (35983), Jonathan Stuermlinger (34602), & Ben Koellner (34603)

Introduction

by Jonathan Stuermlinger (34602)

For our AMC project, we developed a basic traffic monitoring system using a Raspberry Pi 5, designed to detect and count vehicles as they pass through a designated section of road captured by a camera. The collected data is uploaded to a web-based dashboard, which could provide a foundation for traffic flow analysis and supporting studies on air pollution, noise levels, and overall urban livability. Such information can also play a valuable role in infrastructure planning and broader environmental monitoring initiatives.

Material & Methods

Hardware Components

by Nicole Alvarado (35983) & Ben Koellner (34603)

The project was developed using the following hardware components:

- Raspberry Pi 5
- Raspberry Pi High Quality Camera (CS Mount)
- MicroSD card (32 GB)
- Power supply
- HDMI cable
- Monitor
- Mouse and keyboard

Hardware Assemblance

To prepare the hardware environment for traffic monitoring, the Raspberry Pi 5 was assembled to create a functional and interactive environment for real-time object detection. The hardware integration process is as followed:



Fig. 1: System Build

1. Insert the camera’s ribbon cable into the CSI (Camera Serial Interface) port of the Raspberry Pi by lifting the plastic latch and pressing back down to secure the cable.
2. Screw the CS-mount lens onto the HQ camera board and attach the other side of the camera’s ribbon cable.
3. Connect the USB keyboard and USB mouse into the USB ports of the Raspberry Pi.
4. Connect the monitor using an HDMI to micro-HDMI cable.
5. Once flashed, insert the 32 GB microSD card.
6. Plug in the USC-C power supply to the Raspberry Pi’s power port, this will automatically boot the system.
7. Mount the camera in a fixed position using a tripod facing the desired street area.

System Setup

Raspberry Pi Initialization

The Raspberry Pi 5 was set up using a flashed MicroSD card containing the latest version of Raspberry Pi OS (64-bit). This was done via the Raspberry Pi Imager by selecting:

- Device: Raspberry Pi 5
- Operating System: Raspberry Pi Os (64-bit).
- Storage: 32 GB MicroSD card.



Fig. 2: Raspberry Pi imager

Installing OpenCV and Required Packages

To enable object detection, OpenCV version 4.8.1 was compiled. The following commands were executed on the Raspberry Pi terminal:

```
sudo apt-get update && sudo apt-get upgrade
sudo nano /etc/dphys-swapfile
```

Essential libraries and tools were installed:

```
sudo apt-get install -y \
build-essential cmake pkg-config \
libjpeg-dev libtiff-dev libpng-dev \
libavcodec-dev libavformat-dev libswscale-dev libv4l-dev \
libxvidcore-dev libx264-dev \
libgtk-3-dev \
libatlas-base-dev gfortran \
python3-dev python3-numpy python3-pip \
python3-picamera2 python3-opencv
```

OpenCv library and its additional modules were retrieved from Github, unzipped, and set up for the build process:

```
cd ~
wget -O opencv.zip https://github.com/opencv/opencv/archive/4.8.1.zip
wget -O opencv_contrib.zip
https://github.com/opencv/opencv_contrib/archive/4.8.1.zip

unzip opencv.zip
unzip opencv_contrib.zip

mv opencv-4.8.1 opencv
mv opencv_contrib-4.8.1 opencv_contrib
```

Compilation and installation of OpenCV performed as follows:

```
cd ~/opencv/
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_contrib/modules \
      -D BUILD_EXAMPLES=ON ..

make -j$(nproc)
sudo make install
sudo ldconfig
```

Creating a Python Virtual Environment for the dashboard

To prevent conflicts with the system's main Python package repository, the dashboard was developed within a virtual environment to manage its required packages independently.

```
sudo apt install python3-venv python3-full
python3 -m venv ~/dashboard-env
source ~/dashboard-env/bin/activate
pip install flask pandas plotly
```

Files Setup and Execution

Provided below is a downloadable ZIP containing the pre-trained COCO object detection model used in this setup. It also includes a list of all detectable object classes that the COCO library has been trained to recognize. All project files, including the object detection, dashboard scripts and vehicle log, were placed into the previously downloaded archives (Object_Detection_Files) and transferred to the Raspberry Pi Desktop using a USB stick.

object_detection_files.zip

To execute the object detection script, enter the following command in the terminal.

```
cd /home/pi/Desktop/Object_Detection_Files
python3 object_detection_final.py
```

Once the detection process is running, you can launch the live dashboard by activating the virtual environment and starting the server. The dashboard can then be accessed via a web browser using the address shown in the terminal.

```
source ~/dashboard-env/bin/activate
cd /home/pi/Desktop/Object_Detection_Files
python dashboard.py
```

Results

by Nicole Alvarado (35983)

Code

Three Python scripts were developed as part of the system's functionality.

1. Object Detection
2. Vehicle Log
3. Dashboard

The object detection script (object_detection_final.py) uses OpenCV's DNN module with a pre-trained SSD MobileNet model trained on the COCO dataset. It captures video input from the Pi camera, identifies vehicles (car, bus, truck, motorbike), draws bounding boxes, and logs the counts into a CSV file.

```
from picamera2 import Picamera2
import time
import os
from datetime import datetime
import csv
import cv2
import numpy as np
from scipy.spatial import distance as dist
from collections import OrderedDict

# === Load class names ===
classNames = []
classFile = "/home/aless/Desktop/Object_Detection_Files/coco.names"
with open(classFile, "rt") as f:
    classNames = f.read().rstrip("\n").split("\n")

# === Load model config and weights ===
configPath =
"/home/aless/Desktop/Object_Detection_Files/ssd_mobilenet_v3_large_coco_2020
_01_14.pbtxt"
weightsPath =
"/home/aless/Desktop/Object_Detection_Files/frozen_inference_graph.pb"

assert os.path.exists(classFile), f"Missing file: {classFile}"
assert os.path.exists(configPath), f"Missing file: {configPath}"
assert os.path.exists(weightsPath), f"Missing file: {weightsPath}"

# === Load the model ===
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)

# === Vehicle detection function ===
def getObjects(img, thres, nms, draw=True, objects=[]):
    classIds, confs, bbox = net.detect(img, confThreshold=thres,
nmsThreshold=nms)
    objectInfo = []

    if len(objects) == 0:
        objects = classNames

    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, className.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

# === Log object count to CSV ===
def log_vehicle_count(vehicle_type, total):
    with open("vehicle_log.csv", "a", newline='') as f:
        writer = csv.writer(f)
        writer.writerow([datetime.now().isoformat(), vehicle_type, total])

# === Centroid Tracker ===
class CentroidTracker:
    def __init__(self, maxDisappeared=20):
        self.nextObjectID = 0
        self.objects = OrderedDict()
        self.disappeared = OrderedDict()
        self.maxDisappeared = maxDisappeared
        self.counted = set()

    def register(self, centroid):
        self.objects[self.nextObjectID] = centroid
        self.disappeared[self.nextObjectID] = 0
        self.nextObjectID += 1

    def deregister(self, objectID):
        del self.objects[objectID]
        del self.disappeared[objectID]
```

```
    def update(self, inputCentroids):
        if len(inputCentroids) == 0:
            for objectID in list(self.disappeared.keys()):
                self.disappeared[objectID] += 1
                if self.disappeared[objectID] > self.maxDisappeared:
                    self.deregister(objectID)
            return self.objects

        if len(self.objects) == 0:
            for i in range(0, len(inputCentroids)):
                self.register(inputCentroids[i])
        else:
            objectIDs = list(self.objects.keys())
            objectCentroids = list(self.objects.values())
            D = dist.cdist(np.array(objectCentroids), inputCentroids)
            rows = D.min(axis=1).argsort()
            cols = D.argmin(axis=1)[rows]

            usedRows = set()
            usedCols = set()

            for (row, col) in zip(rows, cols):
                if row in usedRows or col in usedCols:
                    continue
                objectID = objectIDs[row]
                self.objects[objectID] = inputCentroids[col]
                self.disappeared[objectID] = 0
                usedRows.add(row)
                usedCols.add(col)

            unusedRows = set(range(0, D.shape[0])).difference(usedRows)
            for row in unusedRows:
                objectID = objectIDs[row]
                self.disappeared[objectID] += 1
                if self.disappeared[objectID] > self.maxDisappeared:
                    self.deregister(objectID)

            for col in set(range(0, D.shape[1])).difference(usedCols):
                self.register(inputCentroids[col])

        return self.objects

# === Main application ===
if __name__ == "__main__":
    picam2 = Picamera2()
    picam2.configure(picam2.create_preview_configuration(main={"format":
"RGB888", "size": (640, 480)}))
    picam2.start()
    time.sleep(1)
```

```
    # Now also counting bicycle and person
    detectable_classes = ["car", "bus", "truck", "motorbike", "bicycle",
"person"]
    object_totals = {cls: 0 for cls in detectable_classes}
    line_x = 320  # vertical line position
    tracker = CentroidTracker()
    id_to_type = {}

    while True:
        img = picam2.capture_array()
        result, objectInfo = getObjects(img, 0.45, 0.2,
objects=detectable_classes)

        # Draw vertical counting line
        cv2.line(img, (line_x, 0), (line_x, 480), (255, 0, 255), 2)

        # Get centroids of current detections
        detections = []
        for box, name in objectInfo:
            x, y, w, h = box
            detections.append((name, (x + w // 2, y + h // 2)))

        centroids = np.array([c for _, c in detections])
        objects = tracker.update(centroids)

        for objectID, centroid in objects.items():
            cx, cy = centroid
            cv2.circle(img, (cx, cy), 4, (0, 0, 255), -1)
            cv2.putText(img, f"ID {objectID}", (cx - 10, cy - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

            # Match ID with detected type
            if objectID not in id_to_type:
                for (name, (dx, dy)) in detections:
                    if abs(cx - dx) < 30 and abs(cy - dy) < 30:
                        id_to_type[objectID] = name
                        break

            # Count if crossing vertical line
            if objectID not in tracker.counted:
                if cx > line_x - 15 and cx < line_x + 15:
                    obj_type = id_to_type.get(objectID)
                    if obj_type:
                        object_totals[obj_type] += 1
                        tracker.counted.add(objectID)
                        log_vehicle_count(obj_type, object_totals[obj_type])

        # Display total and per-class counts
        total = sum(object_totals.values())
        cv2.putText(img, f"Total Counted: {total}", (20, 40),
```

```
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 255, 0), 2)
        y_offset = 80
        for obj_type in detectable_classes:
            label = f"{obj_type.title()}: {object_totals[obj_type]}"
            cv2.putText(img, label, (20, y_offset),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 255), 2)
            y_offset += 30

        cv2.imshow("Output", img)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cv2.destroyAllWindows()
```

The vehicle_log.csv serves as the system's central data log for vehicle detection events. It is automatically generated and updated by the object detection script each time a vehicle is recognized in the camera feed. Each entry in the file consists of a timestamp, the type of vehicle detected (e.g., car, bus, truck, motorbike), and a numeric count (typically per detection). This log acts as the primary data source for the dashboard script, which reads and processes it in real time to generate visualizations. The file is stored locally in the same directory as the scripts and can be reset via the dashboard interface to clear all logged entries. The log of the data looks like this:

```
2025-07-28T16:04:24.174465,car,14
2025-07-28T16:04:27.597785,car,15
2025-07-28T16:04:30.165927,car,16
2025-07-28T16:04:34.733430,car,17
2025-07-28T16:04:46.082917,car,18
2025-07-28T16:05:21.296150,car,19
2025-07-28T16:05:27.438530,car,20
2025-07-28T16:05:32.466989,car,21
2025-07-28T16:05:46.877602,car,1
2025-07-28T16:05:58.076430,car,2
2025-07-28T16:06:20.091424,car,3
2025-07-28T16:06:29.027871,person,1
2025-07-28T16:06:43.929392,car,4
2025-07-28T16:07:02.028061,car,1
2025-07-28T16:07:05.071340,car,2
2025-07-28T16:07:07.032200,car,3
2025-07-28T16:07:18.122763,car,4
2025-07-28T16:07:29.345924,car,5
2025-07-28T16:07:46.129527,car,6
2025-07-28T16:08:30.797174,person,1
2025-07-28T16:08:34.676911,person,2
2025-07-28T16:08:39.359371,person,3
```

The dashboard script (dashboard.py) creates a local web server using Flask and displays a real-time plot of vehicle counts using Plotly. The data is read from the vehicle_log.csv file, and the dashboard includes a reset button and the custom logo of CarPi.

```
from flask import Flask, render_template_string, redirect, url_for
import pandas as pd
```

```
import plotly.graph_objs as go
import plotly.offline as pyo
from collections import Counter
import os
from datetime import datetime

app = Flask(__name__)

@app.route("/")
def index():
    if not os.path.exists("vehicle_log.csv") or
os.path.getsize("vehicle_log.csv") == 0:
        return """
        <html><body style='font-family:sans-serif;padding:2em;text-
align:center'>
            <h2>No data yet</h2>
            <p>Waiting for vehicle detections...</p>
            <a href='/' style='display:inline-block;margin-
top:1em;'>Refresh</a>
        </body></html>
        """

    df = pd.read_csv("vehicle_log.csv", names=["Time", "Vehicle_Type",
"Count"])
    df["Time"] = pd.to_datetime(df["Time"])

    fig = go.Figure()
    total = df.groupby("Time").size().cumsum()
    fig.add_trace(go.Scatter(x=total.index, y=total.values,
mode='lines+markers', name='Total Vehicles'))

    for vtype in df["Vehicle_Type"].unique():
        sub = df[df["Vehicle_Type"] ==
vtype].groupby("Time").size().cumsum()
        fig.add_trace(go.Scatter(x=sub.index, y=sub.values,
mode='lines+markers', name=vtype.title()))

    fig.update_layout(
        title="Live Vehicle Count",
        template="plotly_white",
        xaxis_title="Time",
        yaxis_title="Cumulative Count",
        legend=dict(orientation="h", y=-0.3),
        margin=dict(t=60, b=40)
    )

    graph_html = pyo.plot(fig, output_type="div", include_plotlyjs="cdn")

    # Summary bar
    totals = Counter(df["Vehicle_Type"])
```

```
    total_count = sum(totals.values())

    summary_html = "<div style='display:flex; flex-wrap:wrap; gap:20px;
margin-bottom:20px;'>"
    summary_html += f"<div class='summary-card'><h3>Total
Vehicles</h3><p>{total_count}</p></div>"
    for vtype, count in totals.items():
        summary_html += f"<div class='summary-
card'><h3>{vtype.title()}</h3><p>{count}</p></div>"
    summary_html += "</div>"

    html = f"""
    <html>
    <head>
        <title>CarPi Dashboard</title>
        <meta http-equiv="refresh" content="10">
        <style>
            body {{
                font-family: Arial, sans-serif;
                background: #f9f9f9;
                padding: 20px;
                color: #333;
            }}
            .header {{
                display: flex;
                align-items: center;
                gap: 15px;
                margin-bottom: 20px;
            }}
            .header img {{
                width: 60px;
                height: auto;
            }}
            .summary-card {{
                background: #ffffff;
                padding: 15px 20px;
                border-radius: 8px;
                box-shadow: 0 2px 6px rgba(0,0,0,0.1);
                min-width: 120px;
                text-align: center;
            }}
            h1 {{
                font-size: 28px;
                margin: 0;
            }}
            .top-bar {{
                display: flex;
                justify-content: space-between;
                align-items: center;
                margin-bottom: 10px;
            }}
```

```
            .reset-button {{
                background: #e74c3c;
                border: none;
                color: white;
                padding: 10px 15px;
                border-radius: 5px;
                cursor: pointer;
                font-size: 14px;
            }}
        </style>
    </head>
    <body>
        <div class='top-bar'>
            <div class='header'>
                <img src="/static/carpi_logo.png" alt="CarPi Logo"
width="60">
                <h1>CarPi Dashboard</h1>
            </div>
            <form method="post" action="/reset">
                <button class="reset-button" type="submit">Reset
Log</button>
            </form>
        </div>
        {summary_html}
        {graph_html}
    </body>
    </html>
    """
    return render_template_string(html)

@app.route("/reset", methods=["POST"])
def reset_log():
    if os.path.exists("vehicle_log.csv"):
        with open("vehicle_log.csv", "w") as f:
            pass
    return redirect(url_for("index"))

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

Discussion & Conclusion

by Jonathan Stuermlinger (34602) & Ben Koellner (34603)

While our traffic monitoring system functions reliably in its current form, there were several lessons learned throughout the project. Initially, our goal was to develop the system using the Raspberry Pi 5 in combination with an AI Hat (an additional Raspberry Pi module designed to enhance computing performance) and the YOLO algorithm for advanced object detection. However, due to repeated compatibility and setup issues, we were unable to make the AI Hat fully operational and instead opted

for an alternative approach based on OpenCV and COCO (Common Objects in Context) libraries. An important takeaway from this experience is that a more successful integration of the AI Hat and YOLO framework would likely have required reaching out for external support early on, especially to better understand the specific requirements and limitations of the Raspberry Pi 5 and the AI Hat. Despite these challenges, the alternative solution we implemented proved to be a solid and functional approach for basic vehicle detection and counting.

However, the system is not without limitations. For example, detecting larger vehicles such as buses proved inconsistent, especially depending on the camera angle and environmental conditions. In our case, the monitoring frame was set up between trees, which occasionally interfered with object detection.

Beyond technical issues, permanent deployment of such a system would also require consideration of legal aspects, such as regulations around filming public roads and ensuring privacy compliance.

Currently, the system logs data locally, and access is limited to devices on the same network. A long-term implementation would require modifications to reset the vehicle count periodically (e.g., every 5 minutes or more frequently during peak hours) and to store the data in a publicly accessible database. Data visualization could be improved once sufficient data is collected for meaningful analysis.

Despite these challenges, the project demonstrates that a low-cost and accessible traffic monitoring solution is possible without specialized AI hardware. With further development, such a system could be adapted for real-world use cases in traffic analysis, planning, or environmental studies.

Video

References

-insert info

From:
https://student-wiki.eolab.de/ - **HSRW EOLab Students Wiki**

Permanent link:
https://student-wiki.eolab.de/doku.php?id=amc:ss2025:group-e:start&rev=1753800994

Last update: **2025/07/29 16:56**