RPI DASHCAM: Vehicle Surveillance and Crash Detection System

Team Members:

Manchala Sashank	2023EEB1221
Mudavath Sriman	2023EEB1225
Pasula Kohith	2023EEB1232
Mohak Singla	2023EEB1224

Introduction

Dashboard cameras ("dashcams") have become invaluable for vehicle safety and accountability. A small, inexpensive dashcam can **provide critical evidence** in accidents or disputes. For example, insurance companies note that recorded footage from a dashcam "provides direct evidence … of what caused an accident", helping to resolve claims quickly and reduce fraud. Our team's goal was to build an **affordable, open-source dashcam** using a Raspberry Pi, driven by the motivation to enhance safety without high cost. By using common hardware and free software, the dashcam can be customized by hobbyists or fleets for **low-cost monitoring and crash evidence**. In summary, we aim to demonstrate that a low-cost Pi-based system can address real-world needs for vehicle surveillance and accident documentation, while enabling learners to understand end-to-end hardware–software integration.

Problem Statement

While commercial dashcams exist, they are often **proprietary, expensive, and inflexible**. Many lack customizable features (like automatic uploads or crash alerts) and can strain budgets, especially for fleets or DIY enthusiasts. An open-source, low-cost solution is needed that users can modify or extend. By leveraging the Raspberry Pi platform, our project addresses the need for a **flexible dashcam system**: one that can record video continuously, integrate additional sensors, and be programmed to perform smart tasks. This fills a gap where users want a **customizable, transparent dashcam** (for example, to adjust how events are logged or alerts are sent) rather than a locked-down commercial device.

Objective of the Project

Our objective was to evolve a basic Pi dashcam into a **smart surveillance system with crash detection and alerts**. In practical terms, the system must **continuously record video** while the vehicle is running, and **detect collisions or shocks** via a vibration sensor. Upon a crash event, the system should **automatically capture the relevant footage** and send an alert (e.g. an email) with a video snippet and timestamp. It *not only records driving footage continuously but also features an innovative crash detection system.*". Our system realizes this by tying together video recording and sensor monitoring. We have implemented this using modular Python scripts (running as system services) so that on each boot the dashcam starts recording autonomously. In effect, our dashcam transitions from a simple recorder into a **smart, safety-enhancing device**

Components & Tools

The dashcam uses the following hardware components and software tools:

- **Raspberry Pi 3 Model B**– The core computer running all code. Acts as the central controller, featuring a 1.2 GHz quad-core ARM Cortex-A53 CPU and 1 GB RAM, sufficient for basic video recording and sensor tasks. It includes built-in Wi-Fi, Bluetooth 4.1, 4 USB ports, and GPIO pins for hardware expansion. The CSI camera port and microSD storage make it ideal for lightweight, low-cost dashcam setups.

Figure: Raspberry Pi single-board computer (Model B shown) with key interfaces labelled (CSI camera port, USB, GPIO, etc.).

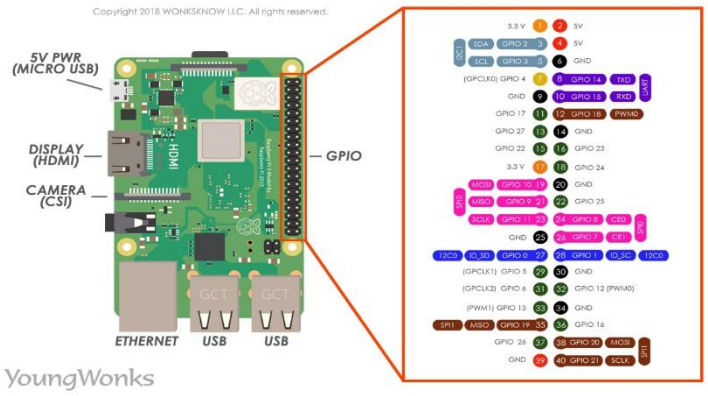
- **Raspberry Pi Camera Module v1.3 (5 MP)** – A CSI-connected camera for video capture. We chose the official Pi Camera for its compact size and direct Pi support.
- **SW-420 Vibration Sensor Module** – An off-the-shelf shock/vibration switch used to detect sudden jolts. The SW-420 is a high-sensitivity, **non-directional** sensor: it outputs a digital HIGH when stable and goes LOW when it detects vibration 5 . By mounting it in the vehicle, it effectively signals collisions or strong impacts.
- **USB Flash Drive (16–32 GB)** – A removable flash drive for storing recorded video. We automount the drive at `/media/usb` whenever inserted to offload video from the SD card.
- **MicroSD Card (16–32 GB)** – Holds the Raspberry Pi OS and our scripts; used for booting and temporary storage until USB takes over.
- **Jumper Wires and Breadboard** – For wiring the SW-420 sensor to the Pi's GPIO (sensor VCC to 5 V, GND to ground, and D0 to a GPIO pin).
- **Additional Items:** A helmet (for demonstration purposes), a micro-USB cable and power supply (stable 5 V @ 3 A for the Pi).

Software and Tools:

- **Raspberry Pi OS (Lite)** – The Linux OS (32-bit) running on the Pi. Lightweight and supports all needed drivers and services.
- **Python 3 (GPIOZero, smtplib, etc.)** – We used Python for scripting. GPIO Zero reads the SW-420 signal, and `smtplib` handles email alerts.
- **Picamera2 (libcamera-based)** – A Python library for camera control on the Pi. Picamera2 is the current recommended camera API on Raspberry Pi, built on the open-source libcamera *which supports modern camera interfaces. This allows efficient 720p video capture.*
- libcamera (and related utilities) – Underlying camera infrastructure in Raspberry Pi OS. We used lib camera-vid for low-level capture.
- Systemd – For auto-starting our scripts at boot. Each module (recording, crash detection) is wrapped in a systemd service so the dashcam runs hands-free.
- Cron– We use simple cron jobs to monitor USB insertion and trigger file transfer scripts when the USB drive is plugged in.

- Email/SMTP* – We configured SMTP (e.g. Gmail's SMTP server) with an app password to send crash alerts. Emails contain the timestamp and video clip of the detected event.

Circuit Diagram:



System Design & Architecture

The dashcam's design is modular. At a high level, there are three interacting components: a **Video Recording Module**, a **Storage Management Module**, and a **Crash Detection Module**. Their interactions can be summarized as follows:

• **Video Recording Module (main.py):** Continuously captures video from the Pi Camera. It records the video at 1280×720 resolution, saving files with timestamped names (e.g. `vid_2025-05-11_14-30.h264`). Files are initially written to the SD card root (e.g. `/home/pi/videos/`) to ensure recording even if no USB is present.

• **Storage (USB) Manager:** Monitors for a USB drive insertion at a predetermined mount point (`/media/usb`). Upon detecting a valid USB mount, this module immediately **copies all recorded videos** from the SD card to the USB drive and **deletes** them from the SD card to free space. This ensures continuous operation without filling the internal storage. We use a combination of `/etc/fstab` entries and cron/udev scripts to auto-mount the USB consistently at `/media/usb`

• **Crash Detection Module (crash_alert.py):** Continuously polls the SW-420 sensor via a GPIO pin (with software debounce). If vibration is detected (signal goes LOW), it timestamps the event and triggers an **alert routine**. This routine packages the event time and a short video snippet into an email sent via SMTP. The email subject and body include the crash time and optional GPS coordinates (if GPS were added in future). In essence, this module converts physical shocks into immediate alerts paired up with a software debounce to avoid unnecessary/false alerts.

These modules run concurrently under the Pi OS. Systemd services ensure they all start at boot: for example, `dashcam.service` runs `main.py`, and `crash-alert.service` runs `crash_alert.py`. This design means *no user intervention* is needed after powering on the Pi; recording and monitoring begin automatically.

Implementation Details

- **Video Recording:** We used the Picamera2 Python library (libcamera backend) to configure and start video capture. To balance quality and storage, we set a 1280×720 resolution at 30 fps. The script records videos, creating files named by timestamp (e.g.`vid_2025-05-11_14-30.h264`), as shown in similar projects. Each completed segment is closed cleanly before starting the next. Picamera2 (built on libcamera) handles the camera pipeline in hardware-accelerated C code for efficiency.

- **Filename Formatting & Storage Path:** Each file is prefixed with `vid_` and includes the date and time, ensuring chronological order and readability. Videos are saved initially to local storage (SD card) under `/home/pi/videos/`. This avoids write interruptions if the USB isn't present at boot.

- **Auto-Transfer to USB:** We mount the USB drive at `/media/usb`. A background script (launched via cron or udev) polls for the USB's presence. When detected, it copies all `.h264` files from `/home/pi/videos/` to the USB and then removes them from the SD card, preventing storage overflow. (We configured `/etc/fstab` to ensure a consistent mount point) After transfer, the storage scripts may also remove video segments older than a retention threshold (e.g. 30 days).

- **Crash Detection:** The SW-420 vibration sensor is wired to a GPIO input (we chose GPIO 17). We use a small Python script (`crash_alert.py`) that watches the pin in a loop with a brief delay. By default, the SW-420 output is HIGH; a shock pulls it LOW. On detecting a LOW (and with a short software debounce to avoid noise), the code logs the current timestamp. It then composes an email using Python's `smtplib`. The email subject includes "Crash detected" with the timestamp, and the body notes the event time. The script attaches the video file that covers the crash time (the most recently recorded file). We found that enabling Gmail's two-factor auth and using an **app password** was necessary to send mail (solving the common "SMTP login failed" issue). This approach effectively sends an instant alert with evidence (timestamped video) whenever a crash is sensed.

- **Live Streaming (Attempted):** We also explored setting up live video streaming. We tried using tools like `mediaMTX` for HLS streaming (as used in some Pi dashcam projects 9), but ran into complications with camera drivers and latency. While we achieved a basic `libcamera-vid`

RTSP stream on LAN, it was not robust enough for vehicular use. As a result, we recommend future work use established streaming frameworks or dedicated camera modules for live feed.

• **Time/Date Synchronization:** Since the Raspberry Pi lacks a real-time clock, we enabled network time synchronization at boot. The Pi OS (via `systemd-timesyncd`) regularly queries NTP servers (e.g. `time.nist.gov`) to set the system clock. In practice we ran `sudo timedatectl set-ntp true` so that on startup the Pi automatically gets the correct UTC time. This ensures all video timestamps and email alerts carry accurate date/time, which is critical for evidence logging.

Features Implemented

- **Auto-Recording on Boot:** Upon power-up, systemd launches the video recorder automatically – no manual start required.
- **Segmented Video Files:** Video is saved in usb(with date/time names) to simplify retrieval and avoid huge monolithic files.
- **Storage Management:** A USB flash drive is auto-mounted at `/media/usb`. When plugged in, the system copies all video segments to USB and deletes them from the SD card, providing effectively limitless storage.
- **Crash Detection & Alerts:** The SW-420 sensor continuously monitors for shocks. On detection, an email alert is sent with the crash timestamp and attached video snippet, enabling real-time notification of accidents.
- **Modular Script Architecture:** Each component (recording, crash alert, USB handler) is a separate Python script, run as a service. This modularity eases debugging and future extension.
- **Time Sync:** NTP synchronization is enabled so the system clock is accurate, ensuring all footage and alerts are correctly timestamped.

Challenges & Learnings

• **USB Mounting Issues:** We initially had problems with the USB drive not mounting consistently or being read-only. The solution was to create a fixed mount point (/`media/usb`) in `/etc/fstab` and use `systemctl daemon-reload`. This ensured stable mounting and allowed automatic read/write access once a USB is inserted. We learned that on Raspberry Pi OS, changes to fstab require a reboot or daemon reload for `udev` to recognize them.

• **Camera Configuration:** Achieving clear video involved tuning. We found that 720p at 30 fps balanced clarity and file size. The Pi Camera requires correct CSI ribbon seating and ribbon orientation – a hardware misflip initially led to a blank image. We also found that ambient light significantly affects exposure, so future versions should consider an optional IR-cut filter or LED for low-light.

• **GPIO and Sensor Debounce:** The SW-420 tends to output false positives (brief spikes) even with minor bumps. We fixed this by adding a short software debounce (ignore signals shorter than ~0.2 s) and a mandatory 10-second cooldown between alerts. This prevented alert

flooding. We also learned to check the Pi's GPIO pin assignments carefully: one pin we chose was reserved, causing an initial "pin busy" error. Changing to a free GPIO (and enabling pull-up) solved the issue.

• **Email Permissions:** Sending email from Python to Gmail initially failed due to Google's security. Enabling 2FA and using an "App Password" resolved this (a known workaround). We cite that this is required to allow `smtplib` to authenticate.

• **Systemd Debugging:** When writing systemd service files, we encountered path errors (using relative paths) and Python version mismatches. The fix was to hardcode absolute paths (e.g. `/user/bin/python3`) and ensure `ExecStart` points to the full script path.

• **Modularity Best Practices:** Splitting functionality into separate scripts (instead of one large monolith) greatly simplified testing. For example, we could test the crash-alert module independently of the video recorder. This approach also made it easy to restart one service without affecting others.

Future Improvements

- **GPS Location Logging:** Adding a GPS module (connected via USB or GPIO) would allow embedding location data into videos and alerts, showing exactly where a crash occurred. This improves the utility of evidence.
- **Cloud Upload:** Instead of only local USB storage, implementing an automatic upload (e.g. to Google Drive, AWS S3, or a private server) would ensure data is backed up immediately and accessible remotely. For instance, after a crash event, the system could push the clip to a cloud account.
- **Motion Detection/Parking Mode:** We could extend functionality to record when the vehicle is parked and motion is detected (e.g. security camera mode). This requires a low-power standby and triggers on camera-motion (using OpenCV or a PIR sensor).
- **Mobile App/Push Alerts:** Instead of email alone, integrating SMS or push notifications (via Twilio, Pushbullet, or a custom app) would deliver alerts faster and more reliably to a smartphone.
- **Enhanced Video Formats:** Currently videos are raw `.h264`. We could transcode to `.mp4` for wider compatibility. If performance permits, real-time encoding (e.g. using `ffmpeg` or hardware-accelerated H.265) would make playback easier.
- **AI-based Incident Analysis:** Future work might incorporate machine learning to classify incidents. For example, a neural network could analyze the sensor data and video to distinguish hard braking from actual collisions. This would reduce false positives or even recognize specific events (e.g. rear-end collision vs. pothole).
- **Unified Dashboard:** Building a web interface or dashboard (for example, on the Pi or a cloud server) could allow users to view live status, download videos, and configure settings without accessing the Pi via SSH.

Conclusion

This Raspberry Pi dashcam project demonstrates that low-cost, open-source hardware can provide robust vehicle surveillance and crash detection. We built a system that continuously records video and autonomously alerts on impacts, addressing real-world needs for safety and evidence. Importantly, our solution is modular and extensible: educators and hobbyists can add features like GPS, cloud sync, or analytics. The dashcam's adaptability makes it suitable for teaching (students can learn about GPIO, Linux services, and networking) and for practical deployment (affordable fleet vehicles, delivery drivers, personal cars). In essence, our work shows how a simple Pi and camera can become a smart black-box recorder. By leveraging common protocols (SMTP, NTP, etc.) and free libraries, we achieve sophisticated functionality without expensive hardware. This project underscores the value of DIY IoT solutions in transportation safety – providing evidence in accidents and enhancing driver awareness – all while keeping costs and complexity low.