Driver Monitoring System (DMS) on OKT507 Project
Documentation

Objective
To design and demonstrate a real-time Driver Monitoring System (DMS) on the OKT507 embedded platform that detects driver drowsiness using vision-based techniques and provides timely visual and audio alerts through HDMI output and onboard audio, ensuring safety-oriented behavior suitable for automotive environments.

Problem Statement
On embedded platforms like OKT507, traditional PC-based OpenCV applications cannot be directly used due to limitations such as lack of windowing systems (X11), limited compute resources, and hardware-specific display interfaces. The challenge was to port a working x86-based DMS application
to OKT507 while ensuring:
- Real-time HDMI display using framebuffer
- Accurate drowsiness detection
- Minimal false positives
- Acceptable performance on embedded hardware
- Reliable alert mechanism

Initial Approaches Attempted
Initially, the application was developed and validated on an Ubuntu x86 system using OpenCV with cv::imshow(). When moved to OKT507:
- cv::imshow() was not supported
- Display output was slow and incorrect
- Drowsiness message was always displayed
- Haar detection on full frames caused heavy lag

Early framebuffer attempts showed incorrect colors and very slow HDMI refresh due to per-frame memory mapping and full-resolution processing.

Finalized High-Level Approach
The finalized solution uses:
- Direct framebuffer rendering to /dev/fb1 for HDMI output
- Persistent mmap of framebuffer
- Frame resizing to reduce processing load
- Haar cascade-based eye detection
- Detection executed only every N frames
- Debounce logic using frame counters
- Event-based audio alert

This architecture preserves detection accuracy while achieving acceptable performance on OKT507.

Core Detection Logic

1. Read video frame
2. Resize to reduced resolution
3. Convert to grayscale and equalize histogram
4. Run Haar eye detection periodically

5. If no eyes detected for consecutive frames, mark as drowsy
6. Overlay warning text when drowsy
7. Render frame to framebuffer
8. Trigger audio once per event

Threshold & Drowsiness Decision
Drowsiness is not declared based on a single frame. Instead:
- Detection runs every 5 frames
- A counter increments when eyes are not detected
- If 'no-eye' count exceeds a threshold (e.g., 3), drowsiness is asserted
- If eyes reappear, the counter resets

This temporal filtering significantly reduces false positives.

Alert Delivery Mechanism
Two alert channels are implemented:
- Visual: "DROWSINESS DETECTED" overlay on HDMI display
- Audio: Warning sound using aplay

Audio is triggered only once per drowsy event and not repeatedly every frame, preventing annoyance
and CPU waste.

System Architecture
Components and roles:
- Video Input: AVI file simulating camera feed
- OpenCV Core: Frame processing and Haar detection
- DMS Logic: Drowsiness decision using counters
- Framebuffer Module: Writes ARGB pixels to /dev/fb1
- Audio Module: Plays alert sound
- OKT507 Hardware: Executes application and drives HDMI display

Challenges Faced
- No GUI support on target
- Framebuffer color mismatch
- Extremely slow HDMI updates
- Continuous false drowsiness alerts
- CPU load due to full-frame Haar detection
- Synchronizing video FPS with embedded timing

Key Learnings
- Embedded Linux display handling differs from desktop
- Framebuffer programming is critical for performance
- Vision algorithms must be downscaled for embedded targets
- Temporal filtering is essential to avoid false positives
- Event-driven alerts are better than continuous signaling

Current Status
The application:
- Runs on OKT507

- Displays video smoothly on HDMI (/dev/fb1)
- Demonstrates partial embedded DMS pipeline
- Audio has not been included

Future Enhancements
- Integrate live camera input
- Use face detection before eye detection
- Add head pose estimation

Conclusion
This project successfully demonstrates porting a vision-based Driver Monitoring System from a PC environment to an embedded OKT507 platform. By redesigning the display pipeline, optimizing detection logic, and introducing embedded-friendly performance techniques, a functional and reliable
DMS prototype was achieved, suitable for automotive proof-of-concept demonstrations.