A blue and black text

Description automatically generated

**Software Engineering Department**

Ort Braude College of Engineering  
Capstone Project Phase B – 61998

**AlertWatch**



**24-2-D-8   
Advisor:**  
Shimon Faitelson  
fshimon@braude.ac.il

**Student:**  
Lior Jigalo  
lior.jigalo@e.braude.ac.il  
[GitHub Repository](https://github.com/audiblemaple/AlertWatch)

Contents

[Project definition 3](#_Toc188870187)

[Abstract 3](#_Toc188870188)

[1. Project Description 4](#_Toc188870189)

[2. Project Architecture 5](#_Toc188870190)

[3. Activity Diagram 6](#_Toc188870191)

[4. Project Building blocks 7](#_Toc188870192)

[5. Project Challenges 8](#_Toc188870193)

[6. Utilized Tools 11](#_Toc188870194)

[7. Project Results & conclusions 12](#_Toc188870195)

[8. Learned Lessons 13](#_Toc188870196)

[9. Project Criteria 14](#_Toc188870197)

[10. Physical Testing 15](#_Toc188870198)

[11. Software validation 17](#_Toc188870199)

[12. Hardware validation 20](#_Toc188870200)

[13. User Guide 21](#_Toc188870201)

[14. Maintenance Guide 21](#_Toc188870202)

[15.Technician Guide 21](#_Toc188870203)

[16. References 21](#_Toc188870204)

# Project definition

# Abstract

In modern environments, whether in vehicles, offices, or other settings, maintaining alertness and preventing fatigue-related incidents has become a critical concern. "AlertWatch" presents an innovative solution utilizing advanced computer vision and hardware to monitor individuals and detect signs of drowsiness in real-time. This project leverages facial recognition and facial landmark detection to continuously assess the alertness levels of a person. Upon detecting signs of drowsiness, "AlertWatch" triggers immediate alerts to prevent potential accidents, lapses in productivity or even security concerns. This paper details the development process of "AlertWatch," focusing on its implementation of real-time monitoring through cameras and software algorithms, as well as outlining the development process and hardware options. "AlertWatch" enhances safety and efficiency and sets new standards for proactive alertness management through technology.

# 1. Project Description

Alertwatch is a driver monitoring system based on the low power Hailo15H SOM (System on module), and Bedrock R7000 designed and manufactured by SolidRun.  
It Is designed to increase safety measures in many environments including automotive, military and more. The project utilizes two hardware units, Bedrock R7000, an AMD based IPC (Industrial PC) which is used as a vehicles ECU (Electronic control unit) which manages the vehicles state.  
A Hailo 15H SOM is used as a detection unit running two AI models, One, InsightFace SCRFD model for face detection based on MobileNet-like backbones, and a second model, for face landmarks detection based on a resnet18. The Hailo15H is analyzing the video stream captured through a MIPI-CSI camera which are processed by these AI algorithms to monitor key indicators such as the eye aspect ratio (EAR), blink patterns, and other facial expressions that signify drowsiness. One of the main features of the project is its low power hardware on edge devices powered by embedded Linux environments. The processed data is analyzed to identify signs of drowsiness, such as prolonged eye closure or reduced blink frequency. When the system detects such patterns, it promptly generates alerts to prevent potential accidents or productivity losses.

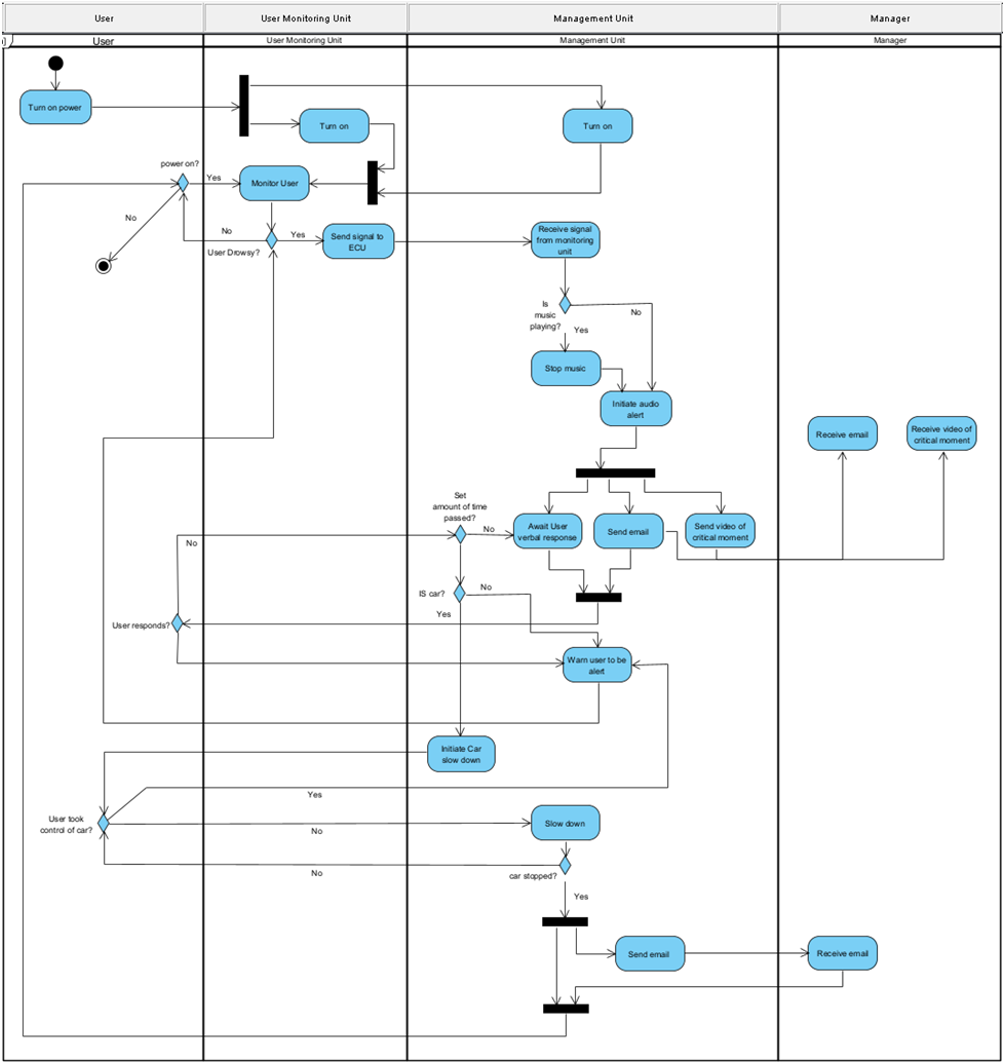
AlertWatch operates on custom Linux images, optimized through tools like Yocto or BuildRoot, ensuring the system’s performance is lightweight and efficient. It is designed to adapt to varying environments, including vehicles, offices, and industrial setups, with seamless integration into existing infrastructures. The system’s proactive approach enables early detection and intervention, enhancing safety and productivity.

By addressing challenges such as lighting variability and hardware constraints, AlertWatch delivers reliable and accurate monitoring under diverse conditions. The project demonstrates significant potential for applications in automotive safety, workplace productivity, and critical operations, setting a new standard for real-time alertness management through the integration of cutting-edge AI and embedded system technologies.

# 2. Project Architecture



# 3. Activity Diagram



# 4. Project Building blocks

The projects building blocks are the R7000 IPC, Hailo15 SOM + carrier and two AI models. I had to research the market for existing models I found a pre-trained lightweight SSD model for face recognition and Google’s MediaPipe model that I could potentially use, however, I decided to train my own Resnet18 model for 68 facial landmarks and use Hailo’s SCRFD insight face pre-trained and pre-compiled model.

Project development building blocks:

1. **Custom Resnet18 model:**

I wrote and trained a custom Resnet18 model using PyTorch to detect 68 facial landmarks based on various facial features. The model was trained on the ibug\_300-W dataset that includes 6000 images as a training set, each image has a corresponding text file with the true values of the landmarks and 666 images as a testing set. The model was trained for 7 full days on an i5 8600K, 16GB RAM & GTX 1060 machine using CUDA. The model was tested using a simple OpenCV example using OpenCV haarcascades for basic face detection. The next step was to convert this model to a format That the Hailo software can understand and run.

1. **Face detection model:** For face detection on Hailo hardware, I utilized the pre-compiled InsightFace SCRFD (Single-Stage RetinaFace Detector) model provided by Hailo. This model is designed specifically for efficient and high-performance face detection tasks. SCRFD is known for being lightweight and suitable for deployment on edge devices like Hailo AI processors while maintaining a suitable detection accuracy.
2. **Monitoring WEB UI:**

I developed a simple web-based user interface that runs on the ECU’s Node.js server. This UI serves as a centralized platform for real-time monitoring of the cars state such as speed and the video feed of the detection unit including post processing.

1. **Operating systems:**

I developed two custom Linux images based on 2 different build systems. The ECU image was developed using BuildRoot, I configured the system packages including python, OpenCV, Hailo drivers, runtime libraries and Node.js as well as system services. The Hailo15H unit had to be developed using Yocto, since all the artifacts and BSP elements are only available for Yocto.

# 5. Project Challenges

1. **Training computing power constraints**

Training AI models is an expensive process, the main challenge here is getting training the model to an adequate level and testing, due to the lack of powerful hardware, testing the base model with different transformation values was a time-consuming process. After the model was validated as adequate to the hardware on hand, it took 7 full days to train it which was a blocking process since the project is based on this model.

1. **Buildroot custom components**

While Buildroot is more straightforward than Yocto and I had some prior experience with it, I never developed custom packages for it, I had to develop multiple custom packages and driver that will be compiled into the image, such as:

* WhisperCPP – A C++ implementation of openAI whisper for voice recognition
* HailoRT driver
* HailortCLI & PYHailoRT

I was not familiar with the process of adding new packages and drivers and had to read the documentation as well as build and rebuild the image many times.

1. **Hailo’s Yocto support outdated**

The main challenge with developing the Linux image for Hailo 15H is that their Yocto version is a non-updated “Kirkstone” which is an LTS (long term support) version, this causes the dependencies to be old and creates unnecessary constraints for the image creation while configuring the Yocto packages.

1. **Yocto’s huge space demand**

Yocto is known for its space and memory complexity since it downloads all the sources and extracts them locally which might often require up to 250GB for a single build, as well as often hitting the 16GB memory limit of the system during build times which forced me to upgrade the build machine to 64GB of ram for stable builds.

1. **Hailo’s runtime system complexity**

The main challenge of the whole project was the complexity of using Hailo’s runtime environment correctly. The model I developed using PyTorch, had to be translated to a format that Hailo can run depending on the specific architecture. This involves the following steps:

1. Translating from .pth to .onnx
2. Parsing .onnx to .har (hailo archive file)
3. Testing unoptimized .har file
4. Optimizing .har to allow the Hailo hardware to run the model (quantization)
5. Testing optimized .har file
6. Compiling optimized .har file to .hef (hailo executable file)
7. Testing compiled .hef

These steps are mainly straight forward, however, the most challenging step was the optimization step. During optimization, the model needs a calibration set to recalibrate the new, quantized weights. Hailo provides an option to use a random calibration set which passes some random data in the required format, to the model to calibrate. This resulted in the model inferring a random blob of dots on the face. While preparing a calibration set of my own also gave a result of random dots scattered all over. I tried to fix this issue for about 4 months, while simultaneously requesting support on Hailo’s community forum and trying to reach Hailo’s internal support team. Finally, I was able to fix this issue by myself after I identified that the calibration set data was in a UINT8 format instead of a FLOAT32, after I recreated the calibration set using the new data types, All I needed to do was tweak the post processing function.

1. **SCRFD post-processing complexity**

Hailo has many demo programs to demonstrate the capabilities of their hardware in whats called a Tappas repository, all of these demos are Gstreamer pipelines, which I cannot use in my python environment, What I had to do is read the C++ source code of the SCRFD post-processing functionality and rewrite it using python which posed great challenges in implementation, due to extensive abstraction of classes and data types as well as performance considerations.

1. **Computing power constrains**

Due to the low power nature of the Hailo15H device performance considerations had to be done during the whole development process, specifically, sending the frames as encoded text as Hailo15H does not have H264 & H265 encoders. Also, the post-processing functions of SCRFD is most computationally intensive, causing the software to process at a rate of 18 FPS which is not acceptable. By implementing a frame skip functionality, I was able to run face detection on a frame and use the same bounding box for X frames of my choosing, which allowed me to run face detection every X frames instead of each frame, skipping 5 frames, resulted in performance improvement to 23 FPS. While optimizing the pre and post processing implementation, allowed reaching the current camera limit of 30 FPS.

1. **MIPI-CSI camera**

The MIPI-CSI camera outputs frames in a 4K 60 FPS RAW format, because Hailo15 does not have H264, H65 encoders conversion on the CPU limits the performance to 12 FPS. Hailo uses some custom Gstreamer plugins to efficiently pass frames from the camera through the ISP and encode using Gstreamer, since I could not find a way to integrate this Gstreamer pipeline into my image capture using OpenCV and the tight time frames I decided to use USB cameras instead, However, this could be done.

# 6. Utilized Tools

1. **PyTorch:** I used PyTorch as a machine learning framework for developing and training the Resnet18 model. Its dynamic and intuitive API allowed me to efficiently implement and test the model. I also used PyTorch's ecosystem to evaluate and verify the trained model.
2. **Photoshop:** I used Adobe Photoshop for creating any UI or web assets for this project.
3. **Play.ht:** I utilized Play.ht, a text-to-speech platform, to create realistic voiceovers and audio content. Its ability to generate natural-sounding speech from text helped streamline the production of voice-based applications and multimedia projects, ensuring a high-quality and engaging user experience.
4. **Jetbrains tools:** JetBrains tools, such as WebStorm and PyCharm, were essential for efficient development. These IDEs provided intelligent code completion, debugging, and version control integration, significantly enhancing productivity in programming languages like Python, and JavaScript.
5. **Hailo Dataflow compiler:** I used the Hailo Dataflow Compiler to optimize and compile deep learning models for deployment on Hailo AI hardware. It streamlined the process of converting pre-trained models into a format compatible with Hailo processors.
6. **Postman:** I used Postman for initial testing of WebSocket communication between the two units.
7. **GitHub:** GitHub served as my primary platform for version control. I used it to manage the code repository.
8. **Whisper:** I used Whisper, an open-source speech-to-text tool by OpenAI, to transcribe temporary audio files into text. Its accuracy and support for offline processing made it ideal for my application allowing the system to interact with the user and vice versa.
9. **V4L (video subsystem for linux):** The video subsystem for Linux allowed me to interface with the camera on the Linux systems I built.

# 7. Project Results & conclusions

I achieved the main goals of my project, Though I faced great challenges, and the Face landmark detection model can be greatly improved, the performance of the software overall meets my estimated performance closely. I consider the project a success. My company considered participating in the 7-10 January 2025 CES event in Las Vegas, and my project would have been displayed there.

**My decision-making process:** My development approach was to keep the architecture modular, in tandem with python’s cross platform runtime I was able to make the detection unit be an ARM machine as well as an X86 machine while minimizing architecture checks on runtime. Separating the dependency of the ECU unit from the detection unit while communicating over WebSocket messages, made the architecture even more modular.

I Also separated all functions to separate python packages allowing me to reuse any part of the code I want while keeping the same code base I built. In addition to modularity, I prioritized user experience, which had to be seamless and as less annoying and invasive as possible, while interacting with the system.

# 8. Learned Lessons

My development process was modular and organized, however, there are several aspects where improvements could help. First, improving the inference pipeline, by running separate threads for each task. A thread that will preprocess the face detection frame and push to a stack, a thread that will run the face detection, post process and push to a stack, a thread that will preprocess the frame for face landmark detection and push to a stack, a thread that will run the face landmark detection, post process including alertness analysis and push to a stack, and finally a thread that will run the WebSocket server to send the frames.

The MIPI-CSI camera on Hailo support is tricky since it outputs frames in RAW format and relies on the ISP to control the camera convert frames to the required format. This means that the regular approach to capturing frames is not suitable here, I had to read some more documentation on that since I did not use the MIPI-CSI camera due to time constraints of the project.

The model training process could be improved, using some external services that allow usage of powerful hardware with an hourly payment, due to the long time it took to train the model, I hesitated to use it due to the monetary constraints of the project.

# 9. Project Criteria

Looking back at the set criteria for the “AlertWatch” project I met most of the criteria. By training a custom AI model for face landmarks detection, successfully porting it to the Hailo hardware, pre and post processing and inferring the user’s alertness levels. However, I have not managed to meet all of them:

* 1. **FR-1.3** The system must detect yawns, this feature was implemented, however, due to time constraints, I had to remove it in favor of more fundamental features due to it being a stretch goal.
  2. **FR-1.4** The system must estimate where the user is looking, this feature was also implemented and removed due to the performance impact it caused and the time it would take to fix that issue as it was declared as a stretch goal.
  3. **FR-6** The system must ensure data privacy and security, the project somewhat achieves these goals.  
     Meaning, the users’ data is not being sent anywhere it can be stored, which ensures privacy, However, the communication between the detection unit and the ECU is now based on a secure WebSocket but on a regular one, meaning its prone to a man in the middle attack.
  4. **FR-10** The system must allow the user to control the cars speed, this requirement was omitted due to its meaningless nature, initially it made sense, but later, as I made the cars speed simulation, it made sense to remove it.

# 10. Physical Testing

The system was tested using real world scenario recording, 30 minutes of driving at night and 30 minutes of driving during the day, monitoring the false negatives and false positives. The videos had the same number of actual events (long eye closure, simulated falling asleep, head movement) and were shot at 60 FPS and test was run twice on each video. Tweaking the software to be as precise as possible. The key part that allows the system to work reliably, is the FPS value, if the FPS value is low, the system will not be able to reliably measure the blink duration and be sure that the timings are correct. This means that high FPS is crucial for reliable operation of the system, in our case any FPS value above 25 (initial expected performance), is suitable for reliable operation.

Day Run:

The day run video was able to detect 100% of events in both runs:

Run 1:  
System was able to achieve an average FPS of 55.

|  |  |
| --- | --- |
| Event | Detected / total |
| Prolonged eye closure | 6 / 4 |
| Falling asleep | 3 / 3 |
| Total blinks | 543 |
| Average eye aspect ratio (eye openness value) | 0.35 |
| Average blink duration | 0.15 sec |

Run 2:  
System was able to achieve an average FPS of 49.

|  |  |
| --- | --- |
| Event | Detected / total |
| Prolonged eye closure | 5 / 4 |
| Falling asleep | 3 / 3 |
| Total blinks | 543 |
| Average eye aspect ratio (eye openness value) | 0.35 |
| Average blink duration | 0.17 sec |

Night Run:

Run 1:  
System was able to achieve an average FPS of 52.

|  |  |
| --- | --- |
| Event | Detected / total |
| Prolonged eye closure | 6 / 4 |
| Falling asleep | 4 / 3 |
| Total blinks | 628 |
| Average eye aspect ratio (eye openness value) | 0.32 |
| Average blink duration | 0.17 sec |

Run 2:  
System was able to achieve an average FPS of 48.

|  |  |
| --- | --- |
| Event | Detected / total |
| Prolonged eye closure | 7 / 4 |
| Falling asleep | 5 / 3 |
| Total blinks | 641 |
| Average eye aspect ratio (eye openness value) | 0.29 |
| Average blink duration | 0.21 sec |

The night run posed greater challenges for the system, primarily due to poor lighting conditions. This significantly impacted the accuracy of face landmark detection, resulting in false positives. Specifically, during the night run, the system registered two false positives in the first run and three in the second run for prolonged eye closure events, as well as one false positive in the first run and two in the second run for falling asleep events. Similarly, the day run also produced false positives, but these were primarily due to the model's quality constraints. When the user turned their head sideways beyond a certain angle, the face landmarks model struggled to maintain precise landmark placements, leading to less accurate measurements.

To conclude, the models reliably detected all actual events and had some false positives which can be improved by using an infrared camera and improving the model’s quality with investing some more resources for the training process, this can further improve the overall system performance and reliability.

# 11. Software validation

Software validation was done in parts for each unit, ECU JavaScript testing and detector python testing.  
JavaScript was tested using the jest package:

|  |  |  |  |
| --- | --- | --- | --- |
| File | Test case | Functionality tested | status |
| Websocket.js |  |  |  |
|  | initWebSocket | initialize a WebSocket server and call startSpeedBroadcast | PASS |
|  | broadcast | send data to all connected clients | PASS |
|  | handleClientMessage | lock alerts when receiving an 'alert' message, then unlock after playSound | PASS |
|  | sendWelcomeMessage | send a welcome message upon client connection | PASS |
| carManager.js |  |  |  |
|  | accelerateCar | set carState to accelerating and increase speed | PASS |
|  | decelerateCar | set carState to decelerating if speed > 0, or stopped if speed <= 0 | PASS |
|  | cruiseDrive | set carState to cruising | PASS |
| Sound.js |  |  |  |
|  | playSound | spawn 'aplay' with the correct file and resolve on exit code 0 | PASS |
|  | askForUserConfirmation | return userResponded if the transcript includes a confirmation phrase return noResponse if transcript includes a no-response phrase  return failedToParse if transcript has none of the known phrases  reject if whisper-cli fails (exit code != 0) | PASS |
| Util.js |  |  |  |
|  | printToConsole | Do not log if DEBUG != 1 | PASS |
|  | getSystemData | return parsed system data when commands succeed return fallback data if commands throw an error | PASS (Linux only) |
|  | getRandomInt | return a random integer < max produce multiple different values over many calls | PASS |

Python was tested using the PyTest framework

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| module | file | Test case | Functionality tested | status |
| Util |  |  |  |  |
|  | AppState.py |  |  |  |
|  |  | test\_default\_initialization | default initialization of the AppState class | PASS |
|  |  | test\_update\_blink\_rate\_empty | update\_blink\_rate() with no recent blinks (empty blink\_timestamps). Should return 0.0 if no blinks in the analysis window | PASS |
|  |  | test\_update\_blink\_rate\_with\_data | update\_blink\_rate() with some existing blink timestamps. Only the timestamps within the analysis window should remain, and blink\_rate should be computed correctly | PASS |
|  |  | test\_check\_prolonged\_EAR\_false | check\_prolonged\_EAR() when EAR is above threshold or not enough time has passed. Should return False in both cases | PASS |
|  |  | test\_check\_prolonged\_EAR\_true | check\_prolonged\_EAR() returns True if EAR has been below the threshold for longer than the prolonged\_EAR\_duration\_threshold | PASS |
|  |  | test\_add\_ear\_measurement | add\_ear\_measurement() appends the new measurement, updates the running sum, and removes old measurements outside the analysis window | PASS |
|  |  | test\_get\_average\_ear\_empty | get\_average\_ear() returns 0.0 when there are no measurements | PASS |
|  |  | test\_get\_average\_ear\_non\_empty | get\_average\_ear() with some EAR measurements. Uses the running sum for O(1) average computation | PASS |
|  |  | test\_is\_drowsy\_high\_blink\_rate | is\_drowsy() triggers "high\_blink\_rate" if blink\_rate > blink\_rate\_threshold | PASS |
|  |  | test\_is\_drowsy\_prolonged\_eye\_closure | is\_drowsy() triggers "Prolonged\_eye\_closure" if check\_prolonged\_EAR() is True | PASS |
|  |  | test\_is\_drowsy\_low\_average\_ear | is\_drowsy() triggers "low\_average\_ear" if average\_ear < 0.29 and the ear\_reset\_cooldown has elapsed | PASS |
|  |  | test\_is\_drowsy\_no\_condition\_met | is\_drowsy() returns (False, "") if no drowsiness condition is met | PASS |
|  | cv\_util.py |  |  |  |
|  |  | test\_init\_cv\_cap | Parametrized test to check init\_cv\_cap on different platforms and with/without videopath | PASS |
|  |  | test\_init\_cv\_cap\_defaults | Test that init\_cv\_cap() sets default width, height, and fps when no arguments are passed. This also confirms videopath=None on a non-Windows platform (assume Linux since it’s the target OS) | PASS |
|  | eyeController.py |  |  |  |
|  |  | test\_calculate\_EAR\_nominal | Test the EAR calculation with a typical, symmetric set of eye landmarks. We'll position the eye corners horizontally and the top/bottom landmarks at known distances | PASS |
|  |  | test\_calculate\_EAR\_zero\_horizontal\_distance | If the horizontal distance between eye[0] and eye[3] is zero, we expect the result to be either inf or nan, because floating-point division by zero doesn't raise ZeroDivisionError in NumPy | PASS |
|  |  | test\_calculate\_EAR\_identical\_points | If all points are the same, the distances are all zero. We expect inf or nan, not a ZeroDivisionError | PASS |
|  |  | test\_calculate\_EAR\_non\_symmetric\_eye | Test with a non-symmetric set of points to ensure the formula holds in a slightly tilted or asymmetric scenario | PASS |
|  |  | test\_calculate\_EAR\_random\_points | Generate random valid eye points where the horizontal distance won't be zero. We just ensure the function returns a float without errors | PASS |
|  | websocketController.py |  |  |  |
|  |  | test\_websocket\_client\_init | Test the initialization of the WebSocketClient | PASS |
|  |  | test\_websocket\_client\_connect\_success | Test that connect() successfully connects on the first attempt | PASS |
|  |  | test\_websocket\_client\_connect\_failure\_retries | Test that connect() retries connection up to 2 times on failure | PASS |
|  |  | test\_websocket\_client\_send\_data\_no\_connection | Test that send\_data() does nothing when there is no active WebSocket connection | PASS |
|  |  | test\_websocket\_client\_send\_data\_success | Test that send\_data() sends the correct JSON data over the WebSocket | PASS |
|  |  | test\_initialize\_websocket | Test that initialize\_websocket creates a WebSocketClient and calls connect() | PASS |

# 12. Hardware validation

ECU device was validated under multiple conditions and workloads, first it’s solderability was validated using an automated system. After which, we use a dedicated oven to validate this device fits the industrial requirements of a vehicles ECU. More specifically, various temperature conditions from sub-zero to 80 degrees ambient. The system was tested in an oven. Different temperatures for the system require different testing conditions. For example, in a -20 degrees environment the main concern is booting the system initially, while at high temperatures, the main concern is runtime stability.

The device was put in the oven and cooled to -20 degrees, then booted into a Linux environment, this ensures the memory is stable. After this, the oven was heated to 80 degrees while running a MemTester test until pass, we expected the system to throttle very seriously, but to remain stable.

The Detection units solderability was validated using an automated test to validate solderability and boot of the system.

# 13. User Guide

The AlertWatch project was designed to be as less intrusive as possible. Hence, in a perfect situation the user should not interact with the system at all. In general, the user should turn on the car’s power, and if needed, answer to the system. For example if the system says: “Allertness check, please say ‘I’m alert’ to confirm you’re attentive” the user should say: “I’m alert”.

# 14. Maintenance Guide

For Developer documentation of the project please visit the [projects docs page](https://audiblemaple.github.io/AlertWatch/)

# 15.Technician Guide

For support of the Bedrock ECU unit please visit the SolidRun developer center: [R7000|R8000 technical documentation](https://solidrun.atlassian.net/wiki/spaces/developer/pages/485949444/Bedrock+R8000+R7000+Technical+Documentation)

For support of the Hailo15H unit please visit the SolidRun developer center: [Hailo15H based products](https://solidrun.atlassian.net/wiki/spaces/developer/pages/677380251/HAILO+15+Based+Porducs)

# 16. References

* National Highway Traffic Safety Administration (NHTSA) - [Drowsy Driving Statistics](https://www.thezebra.com/resources/research/drowsy-driving-statistics/)
* Albadawi Y, Takruri M, Awad M. [A Review of Recent Developments in Driver Drowsiness Detection Systems](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8914892/)
* Mishra G, Datz A, De Miguel Palacio A, ["Facial Landmarks Detection: A Brief Chronological Survey & Practical Implementation"](https://www.researchgate.net/publication/353670204_Facial_Landmarks_Detection_A_Brief_Chronological_Survey_Practical_Implementation)
* Beles H, Vesselenyi T, Rus A, Mitran T, Bogdan Scurt F, Adrian Tolea B, ["Driver Drowsiness Multi-Method Detection for Vehicles with Autonomous Driving Functions"](https://www.mdpi.com/1424-8220/24/5/1541)
* Rita Antunes A, Cristina Braga A, Goncalves J, ["Drowsiness Transitions Detection Using a Wearable Device"](https://www.mdpi.com/2076-3417/13/4/2651)
* Dr.Pravin R. Kshirsagar, Pankaj D, Yuvaraj T, C.A. Sathiya M, ["Fatigue detection using artificial intelligence"](https://www.researchgate.net/publication/360732255_Fatigue_detection_using_artificial_intelligence)
* Jieun L, Tae-yong K, Beak S, Moon Y, Jeong J, ["Real-Time Pose Estimation Based on ResNet-50 for Rapid Safety Prevention and Accident Detection for Field Workers"](https://www.mdpi.com/2079-9292/12/16/3513)
* Kaiming H, Xiangyu Z, Shaoqing R, Jian S, "[Deep Residual Learning for Image Recognition](https://arxiv.org/pdf/1512.03385)"
* Erica Hersh, [How Many Times Do You Blink in a Day?"](https://www.healthline.com/health/how-many-times-do-you-blink-a-day#blinking-frequency)