Software Design Document

ADAS SYSTEM

AUTOMATIC HEADLIGHT DIRECTION ECU

VERSION: 1.0

History of Changes			
Revision	**Issue Date**	**Description**	**Author**
Ver 1.0		Creation of the first draft of the SDD	ALOUANI Moncif MIRALI Hicham
Ver 1.1			
Ver 1.2			
Ver 1.3			

Table of Contents

1 Scope

1.1 Subject

This software design document describes the architecture and system design of the third module, or ECU, in the context of our last academical project that consists in the conception of an Assisted Driving Automated System (ADAS).

1.2 Project Overview

The aim of this project is the conception of an ECU that will allow us to control the headlights of a car prototype, using eye tracking technologies. Our system must be able to read the driver's gaze in four directions and to translate that gaze to the actuators that will move the headlights accordingly. In addition to eye tracking, the system should also be able to detect surrounding lighting and adjust the headlights' intensity accordingly.

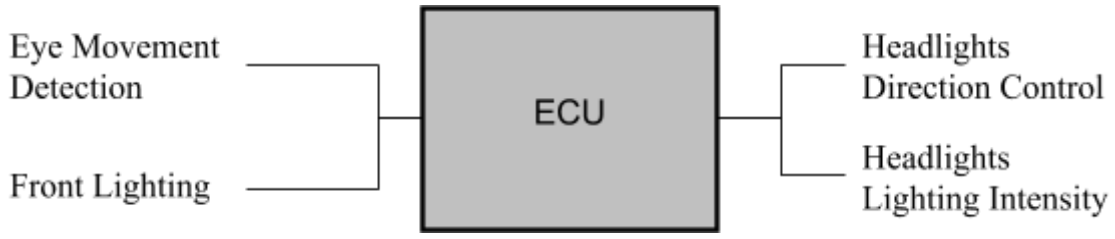

Figure 1: ECU Functionality

The overall purpose of this ECU is to be implemented inside a more global ADAS system that englobes two other ECUs (which means two other functionalities). These ECUs will be described other software design documents.

1.3 Document Overview

This document gives

- Description of one software component of the ADAS system (Automatic Headlight Direction System).
- Documentation Concerning the System (Glossary and Definition)
- Description of the System Environment and how to implement it on both a computer and Raspberry Pi 3.
- Architecture description of the system.

2 Glossary and definition

- ***Face and eye detection :***

OpenCV offers multiple methods to detect a face or an eye from an image or from multiple frames. One of them is the use of Haar feature-based cascade classifiers, it is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images, including faces and eyes, thanks to XML files implemented in the data reportory of the openCV installer.

Even though this method works, it relies principally on thresholding the white of the eyes to detect them on the face, which isn't very reliable in some circumstances, especially ours since our system is designed to be used in low lighting environments.

The facial landmark detection can also be used in our situation. The pre-trained facial landmark detector inside the dlib library is used to estimate the location of ***68 (x, y)-coordinates*** that map to facial structures on the face.

- ***Pupil detection:***

The eye is composed of three main parts: Pupil : the black circle in the middle, Iris : the bigger circle whose colors may differ, and Sclera which is the white part surrounding them.

The challenge here is to identify the pupil from the rest. In order to do so, the captured images are turned gray. The pupil would generally appear darker, thus a precise threshold shall keep only the darkest pixels demarcating the pupil, and eliminate all unnecessary data.

- ***Pupil movement tracking:***

A rectangle marks out the contour of the detected pupil. The variation of its center's coordinates (x,y) reflects the change in the position of the pupil. Eye tracking becomes then possible.

- ***Actuators' control:***

It's the control of the positioning of the headlights (direction and angle) depending on eye movements. For special eye movement angles, depending on the amplitude of the variation on either the x axis or the y axis, a horizontal or vertical direction of the lights are chosen. Otherwise, lights control directions are self-evident. It is worth mentioning that the pupil tracking tolerates a margin of error equivalent to the width and height of the detected contour

3 System environment

3.1 Software environment :

The image processing for the eye tracking feature is implemented using python 2.7, OpenCV 3.0.1 and Dlib toolkit.

For logistical purposes, our system will be first implemented and tested in a computer, under a Linux environment (Ubuntu 16.04).

First step is installing OpenCV 3.0.1. To do so we first need to update and upgrade the environment we'll be working on (the computer in this case). We then need to install the Operating System dependencies and libraries to run OpenCV. When that is done we need to install the Python libraries and set a Python virtual environment to separate the project environment and the global environment. And it is only then that we proceed to the installation and compilation of OpenCV.

All of these steps are thoroughly described in the following link: https://www.learnopencv.com/install-opencv3-on-ubuntu/

Once all of these steps have been completed, we will be able to proceed with the coding of our proposed solution.

However our system will be ultimately implemented inside a Raspberry Pi. Luckily, the steps to install OpenCV on Raspberry Pi are to some extent similar to the the ones described above, for a computer.

Following the steps described in this link will allow us to prepare our raspberry for eye tracking functions: https://www.learnopencv.com/install-opencv-4-on-raspberry-pi/

CANopen is the protocol that assures the communication between all 4 nodes of the system.

3.2 Hardware environment:

Module Caméra V2 8MP : The camera will capture pictures of the driver's eyes.

Raspberry Pi 3 : It will be the node (the ECU) where our system will be developed and implemented.

Rotary base : The base will support the camera and rotates following the driver's face position.

5V-12V Lighting Lamp : We will need two lamps to simulate the headlights of the car. Those lamps shall be able to provide a varying intensity to respect our requirements.

10" Touchscreen : The Touchscreen will simulate the dashboard of the car and display information on the car's state.

Power Transistor: This transistor is the electrical component that will allow us to vary the lighting intensity of the lamps.

4 Architectural Design

Before describing the different blocks of our system, we first need to define the according to the project's requirements. We have redefined those requirements as follow:

- ***RW00:*** The system shall be able to detect a demarcated pupil in dark and bright environnements.
- ***RW01:*** The system shall detect the direction of the movement of the pupil of the conductor's eye.
- ***RW02:*** The camera shall be implemented in a rotative base in order to follow the driver's gaze.
- ***RW03:*** The system shall capture the driver's gaze as long as headlights are one.
- ***RW04:*** The system shall be able to detect exterior lighting intensity (sun, moon, street lights) and adjust the headlights intensity according to it.
- ***RW05:*** The communication shall be assured between all 4 nodes of the system (Lights control, Somnolence, Parking and the main controller) via CAN Bus.
- ***RW06:*** The system shall follow the AUTOSAR implementation (Basic Software, Run Time Environment, Application)
- ***RW07:*** The system shall be optimized for Raspberry Pi.
- ***RW08:*** All data meant to be viewed by the user shall be displayed on the IHM interface.
- ***RW09:*** The system shall direct the headlights in 4 directions (N,S,W,O) according to the driver's gaze.

4.1 Block Descriptor

Eye Tracking Block:

Covers :

- ***RW00:*** The system shall be able to detect a demarcated pupil in dark and bright environnements.
- ***RW01***: The system shall detect four directions according to the movement of the pupil : right, left, up and down.

Camera Adjusting Block:

Covers:

- ***RW02:*** The camera shall be implemented in a rotative base in order to follow the driver's gaze.
- ***RW03:*** The system shall capture the driver's gaze as long as headlights are on.

Headlights control Block:

Covers:

- ***RW09:*** The system shall direct the headlights in 4 directions (N,S,W,O) according to the driver's gaze.

- ***RW04:*** The system shall be able to detect exterior lighting intensity (sun, moon, street lights) and adjust the headlights intensity according to it.

CAN Communication Block:

Covers:

- ***RW08:*** All data meant to be viewed by the user shall be displayed on the IHM interface.
- ***RW05:*** The communication shall be assured between all 4 nodes of the system (Lights control, Somnolence, Parking and the main controller) via CAN Bus.

RW06 and RW07 cannot be implemented into any block since their functionality is more abstract.

4.2 Blocks

Eye Tracking Block:

- Inputs: Img_Face_Acquired (Boolean), Pupil_Detected(Boolean)
- Outputs: Eye_Is_right(boolean) Eye_Is_left(boolean) Eye_Is_up(boolean) Eye_Is_down(boolean)

Camera Adjusting Block:

- Inputs: Face_Left(Boolean), Face_Right(Boolean), Face_Left(Boolean), Face_Up(Boolean) Headlights_State(Boolean)
- Outputs: System_State(Boolean)

Headlights control Block:

- Inputs: Light_intensity_ext(Float), Eye_Is_right(boolean) Eye_Is_left(boolean) Eye_Is_up(boolean) Eye_Is_down(boolean)
- Outputs:Light_intensity_headlights(Float), Headlight_Up(Boolean), Headlight_Down(Boolean), Headlight_Left(Boolean), Headlight_Right(Boolean)

CAN Communication Block:

- Inputs : Data sent via CAN Bus User's pupil of the eye from Pi camera.
- Outputs: CRC check results Eye tracking data following the user's eyes' direction