Software Design Document (SDD)

Software design is a process by which the software requirements are translated into a representation of software components, interfaces, and data necessary for the implementation phase. The SDD shows how the software system will be structured to satisfy the requirements. It is the primary reference for code development and, therefore, it must contain all the information required by a programmer to write code. The SDD is performed in two stages. The first is a preliminary design in which the overall system architecture and data architecture is defined. In the second stage, i.e. the detailed design stage, more detailed data structures are defined, and algorithms are developed for the defined architecture.

This template is an annotated outline for a software design document adapted from the IEEE Recommended Practice for Software Design Descriptions. The IEEE Recommended Practice for Software Design Descriptions have been reduced in order to simplify this assignment while still retaining the main components and providing a general idea of a project definition report. For your own information, please refer to IEEE Std 10161998 for the full IEEE Recommended Practice for Software Design Descriptions.

Team1

Fatigue and somnolence detection

Software Design Document

Name (s): S.Baroudi & N.Fassi

Date: (11/12/2019)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of the ADAS fatigue and somnolence detection system. It will help the different team members being updated about the system and knowing the steps to achieve nicely, pricesely and on the scedule the requirements.

1.2 Scope

Drowsiness while driving causes typical signs of fatigue. Somnolence Recognition System continually analyzes driver's grimaces via a camera installed on the driver's wheel, so that it can recognize signs of sleepiness before the driver falls asleep.

1.3 Overview

This documents is organized in six (five?) main parts ; an introduction part, explaining the goals of the project and its scope, a system overview describing the context and the design of the system, a system architecture part to develop the relationship between the different modules of the system, a data design part that would help devoloppers handle the entries and the input/output data, and finally, a human interface design were we will see the system from the users perspective.

1.4 Definitions and Acronyms

ADAS: advanced assistance aide system
CAN-BUS : controller area network bus
ECU : electronic control unit
Req: short for requirement
DFD : data flow diagram
OS: operating system
DEV: development
FIFO: first in first out

2. SYSTEM OVERVIEW

The aim of this functionality is to prevent accident caused by somnolence. For this, the system will be designed as follows:

rotative cameras will detect the driver’s face and eyes and decide, trought image processing, if he is sleepy,   if so, the driver will be alerted by a non violent sound signal.
This solution will be developed on an embedded plateform to be easily integrated on the vehicle main system without disturbing the driver’s comfort and its data will be transfered to the dashboard using a CAN-Bus, since it is he most used communication protocol for the automotive industry and that it presents some interesting advantages.

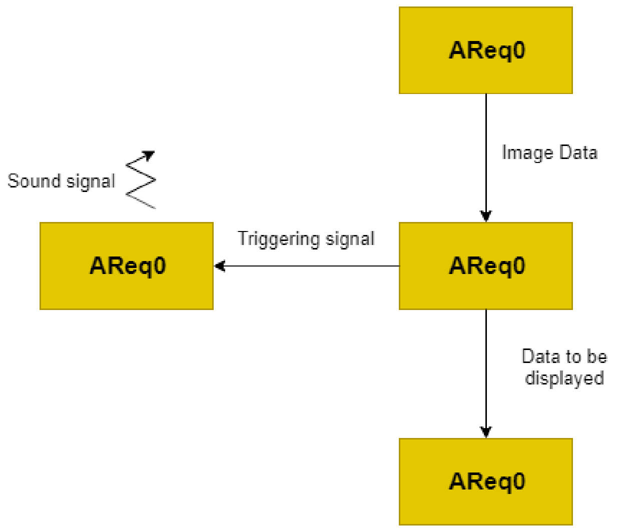
3. SYSTEM ARCHITECTURE

3.1 Architectural Design

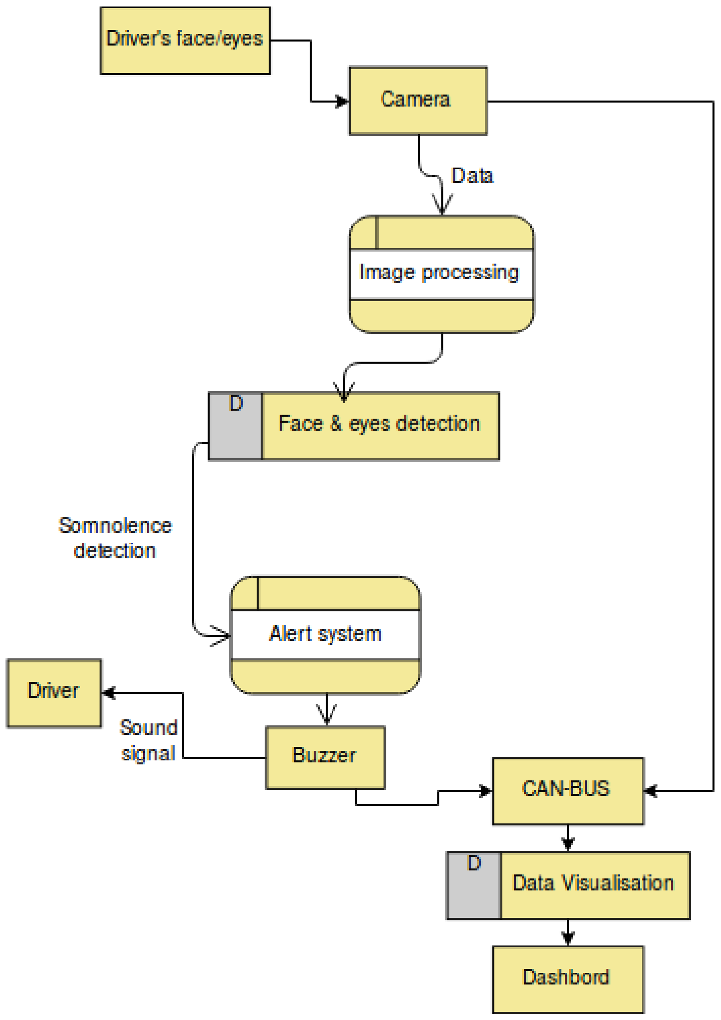
Specifications have been defined into distinct tasks:

- Designing a system that detects if the driver is tired or sleeping while driving, using eye tracking and face expressions processing.
- Alert the driver if he is sleeping with a sound signal (not sudden or loud, so that the driver does not lose control over the vehicle)
- Developing this solution on an embedded platform
- transferring data into a dashboard using a CAN-BUS

Those specifications have been divided into architecture requirements, in which each specification is linked to the adequate components:
AReq0: For this function we will need a camera, rotary support for the hardware. For the software, python and OpenCV library will be the tools used. For the first application, the image processing will be done on a still image, then on a live camera feed.
AReq1: For the sound signal, a small buzzer will be used to generate a light signal in case the driver is sleeping.
AReq2: To make this solution embedded, the system will be transferred on a Raspberry card. The data will be processed throughout this card and should be fully integrated in the vehicle.
AReq3: For this requirement, we will use a Can bus module for the communication between the different ECUs. For the dashboard, we use an interactive screen for information display.



To achieve our goals, the system will be decomposed into four main subsystems.
Below is the top-level data flow diagram (DFD) to understand to functional description:



3.2 Hardware Environment

Pi Camera Device that deliver the live video feed of the driver to the different processing components, it is also automatically compatible to the Raspberry Pi microcontroller.

Rotative camera support Device that ensures the continuous mechanical tracking of the driver's face, in case it moves out of the camera's range.

Piezo Buzzer Device that generates a sound signal using the principle of piezoelectricity. It alarms the drivers in case he is sleeping.

CAN Bus module Device that ensures the communication of data using a CAN protocol between the different nodes.

Raspberry Pi 3 Microcontroller used for processing the data coming from the camera feed, and deciding of the driver's state, therefore alerting him in case he is sleeping. This device will ensure the link between the different nodes and the dashboard used for displaying messages and alerts to the driver.

3.3 Software Environment

Linux The OS or our DEV environment used in the Raspberry Pi card, as it is the most suitable OS for our applications. We will use the distribution with the Linux kernel, the Raspbian distribution.

Python This language ensures wide and rich libraries that will be used for data processing, image processing and data transfer. Therefore, offering pre-made functions for our applications. The main libraries are:

OpenCV Library offering functions for image processing with additional functions found in the Haar cascades XML files.

Numpy Library used for numerical and mathematical operations; it is useful for array processing. Image are made of pixels that can be stored in arrays and processed after.

OpenCAN Library offering functions for the CAN protocol for our communication with the man dashboard.

3.3 Design Rationale

We have chosen this architecture because it is, in our opinion, the easiest to test and the most practical option to connect with the remaining ADAS functionalities of the system.
The fact that the main equipment is the camera allow us to focus more on the software implementation and to handle more on the connection with the CAN Bus and the dashboard.

4. DATA DESIGN

4.1 Data Description

The data collected from the camera will be transferred to the image processing system, based in the Raspberry Pi card and displayed on the dashboard. It will be stored then in the ssd memory card of the Raspberry in a specified memory space and will be eliminated respecting a FIFO logic.

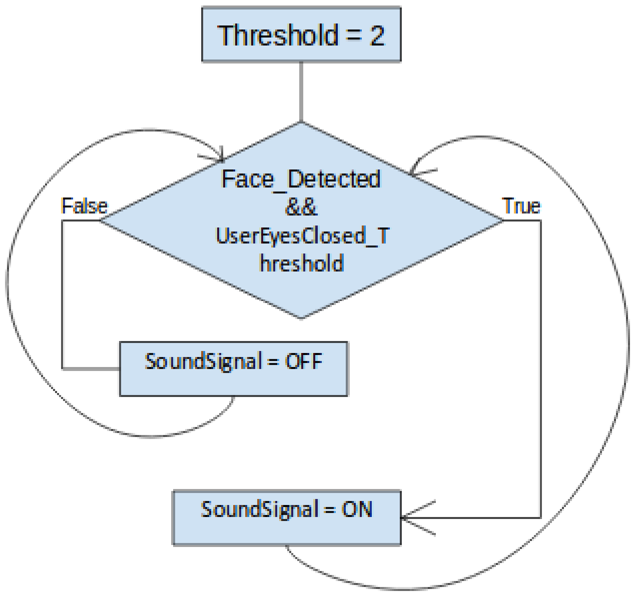
4.2 Data Structure

Data name	Type	Values
userSleeping	Boolean	True/False
SoundSignal	Boolean	True/False
alertMessage	String	The alerting message
timeThreshold	Int	Time in seconds

5. COMPONENT DESIGN

Bellow will be listed, in procedural description language (PDL), a summary of the function's algorithm:

```
Threshold = 2
IF Face_Detected = TRUE
   AND UserEyesClosed_Threshold = TRUE
THEN SoundSignal = ON
ELSE SoundSignal = OFF
```



6. REQUIREMENTS MATRIX

Requirements:
AReq0: detecting the driver's somnolence
-> test scenario:
TS0: face and eyes detection system is provided via an image processing algorithm
-> test cases:
TS0.TC1: face, eyes and eye's width are detected,
TS0.TC2: face is detected, difficulty to detect eyes width
TS0TC3: the driver's face isn't in the camera's visual field

AReq1: triggering the alarm is the case of driver's fatigue
-> test scenario:
TS1: an alarm is connected to the fatigue and somnolence detection system and is triggered when the driver is sleepy
-> test cases:
TS1.TC1: fatigue detected, alarm triggered
TS1.TC2: fatigue detected, alarm not triggered
TS1TC3: fatigue undetected, alarm triggered

AReq2: making this solution embedded on a Raspberry card
-> test scenario:
TS2: system imported on the Raspberry Pi card
-> test cases:
TS2.TC1: working system, independently on the support
TS2.TC2: system does not work on raspberry

AReq3: connecting the system via a Can bus to the dashboard
-> test scenario:
TS3: system connected to dashboard via CAN Bus
-> test cases:
TS3.TC1: system connected to CAN bus, the dashboard connected to CAN Bus
TS3.TC2: system connected to CAN bus, the dashboard not connected to CAN Bus
TS3.TC3: system not connected to CAN bus, the dashboard connected to CAN Bus
TS3.TC4: nor the system and dashboard are connected to CAN Bus

Requirements	Test Scenario	Test Case
AReq0	TS0	TS0.TC1 TS0.TC2 TS0.TC3
AReq1	TS1	TS1TC1 TS1TC2 TS1TC3
AReq2	TS2	TS2.TC1 TS2.TC2
AReq3	TS3	TS3.TC1 TS3.TC2 TS3.TC3 TS3.TC4

8. APPENDICES