Software Design Document

for

Mathworks (Sensor Fusion for Autonomous Systems)

Version 2.0

Prepared by Hagop Arabian, Daniel Gallegos, Roberto Garcia, Gerardo Ibarra, David Neilson, Patrick Emmanuel Sangalang, Jonathan Santos, Deepanker Seth, Angel Tinajero, Xiao Hang Wang

Mathworks

May 12th, 2023

Table of Contents

Revision History

Name	Date	Reason for Changes	Version
First Draft	12/9/2022	Initial Draft of Doc	1.0.0
Final Draft	5/12/2023	Final Draft of Doc	2.0.0

1. Introduction

1.1 Purpose

The purpose of this document is to explain the functions of the web application in detail. The purpose of the web application is to provide drivers lane change assistance. The web application is designed by a machine learning algorithm that uses sensor fusion.

1.2 Document Conventions

Bullet points will indicate sections. External links will be underlined blue.

1.3 Intended Audience and Reading Suggestions

The intended audience for the Software Design Document are software developers, project managers, and faculty advisors.

1.4 System Overview

The web application is used to help a driver assist in a lane change. The driver should receive a notification that tells them when to change lanes. We are implementing our machine learning algorithm called YOMO (You Only Merge Once) that is tested by the KITTI Vision Benchmark Suite dataset from the Karlsruhe Institute of Technology (KIT). The backend will upload and convert the data into a usable format, create a web server, and generate the frames to send to the frontend. The frontend will visually display all relevant data to the application.

2. Design Considerations

The following section details the Assumptions and Dependencies, General Constraints, Goals and Guidelines, and the Development Method.

2.1 Assumptions and Dependencies

- The application is reliant on the KITTI datasets provided from the Karlsruhe Institute of Technology.
- A flask server is needed to host the application.
- REACT, Flexbox, and Matplotlib are required to visualize the car's point of view.

2.2 General Constraints

- **Software Environment**
    - Developers are required to have knowledge in Python.
    - Developers are required to have a basic knowledge of HTML.
    - Developers are required to have a basic knowledge of JavaScript.
    - Developers are required to have a basic knowledge of REACT.
- **End-User Environment**
    - A computer with a web browser to interact with the frontend.
    - A computer with a mouse, keyboard, and monitor are required.
- **Interoperability requirements**
    - Data is gathered from the Karlsruhe Institute of Technology website.
- **Performance requirements**
    - The frontend should display the video comparison, algorithm output and 2D LiDAR minimap as quickly as possible.

2.3 Goals and Guidelines

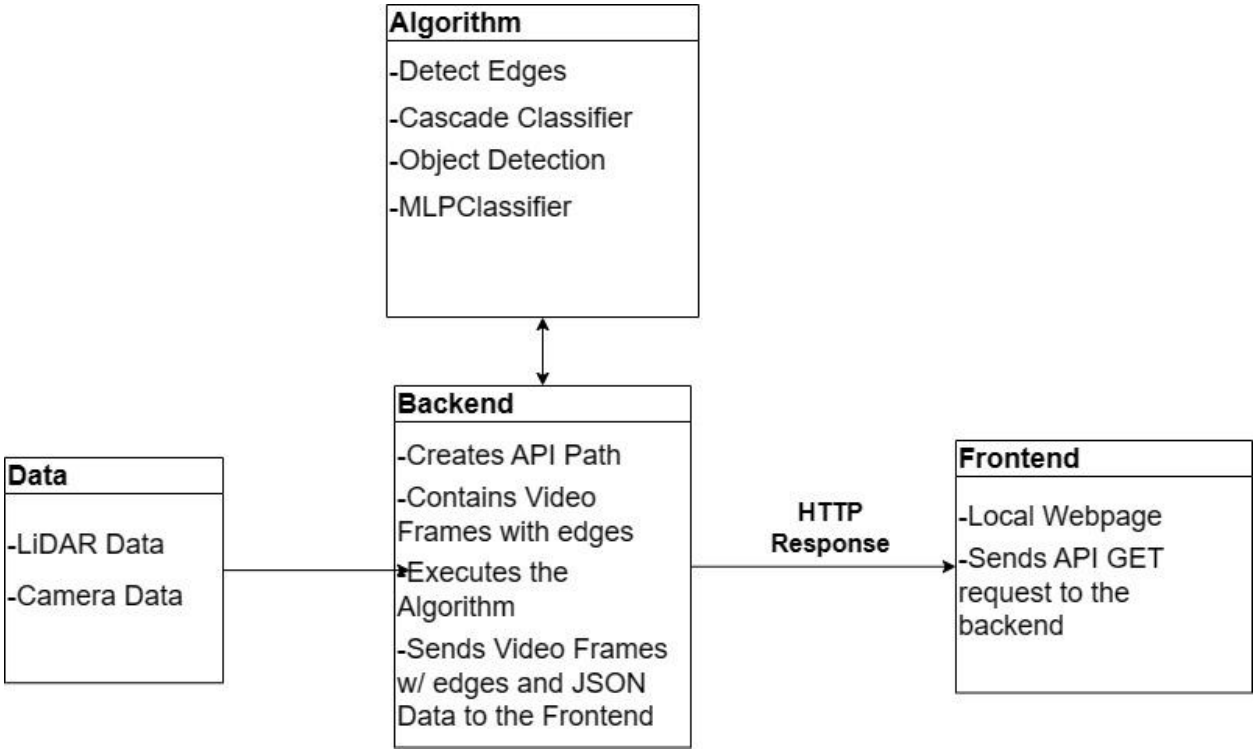
- The application should provide a simple notification to the driver whether it is safe or not to change lanes.

- The application should provide a simple web page interface for developers to perform research and development.

2.4 Development Methods

The team was split into four small groups: data, algorithm, backend, and frontend. The data team was responsible for gathering useful datasets to implement and test the algorithm. The algorithm team was responsible for creating and training a machine learning algorithm when it is safe to change lanes. The backend team was responsible for creating a flask server that would generate relevant data and upload files. The front-end team was responsible for creating a webpage that would display all the relevant data. Communications with all members was distributed through Discord and Zoom.

3. System Architecture



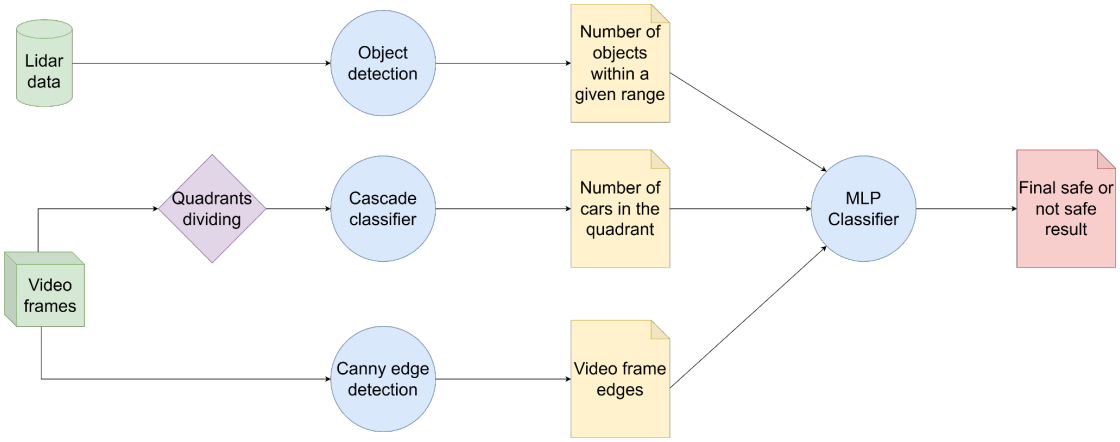
The flowchart above shows the application flow of our project. The application flow consists of a total of four parts. The data collection being either a csv or text file is sent into the backend. The backend converts the data into a JSON file and

creates a server. Then, the backend communicates with the algorithm to generate frames and video streams. Finally, the backend sends all the relevant data to the frontend. The front-end displays all the data into a webpage.

3.1 Data

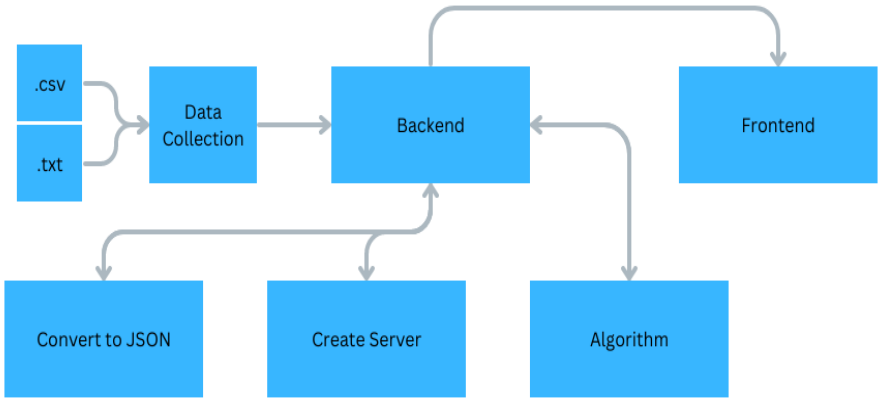
The uploaded LiDAR and camera data require preliminary processing for use in subsequent components. The raw KITTI dataset, that is provided from the Karlsruhe Institute of Technology, is uploaded into the backend server. The server provides the necessary conversion in order to test the algorithm. A total of eight datasets were utilized for the algorithm.

3.2 Algorithm



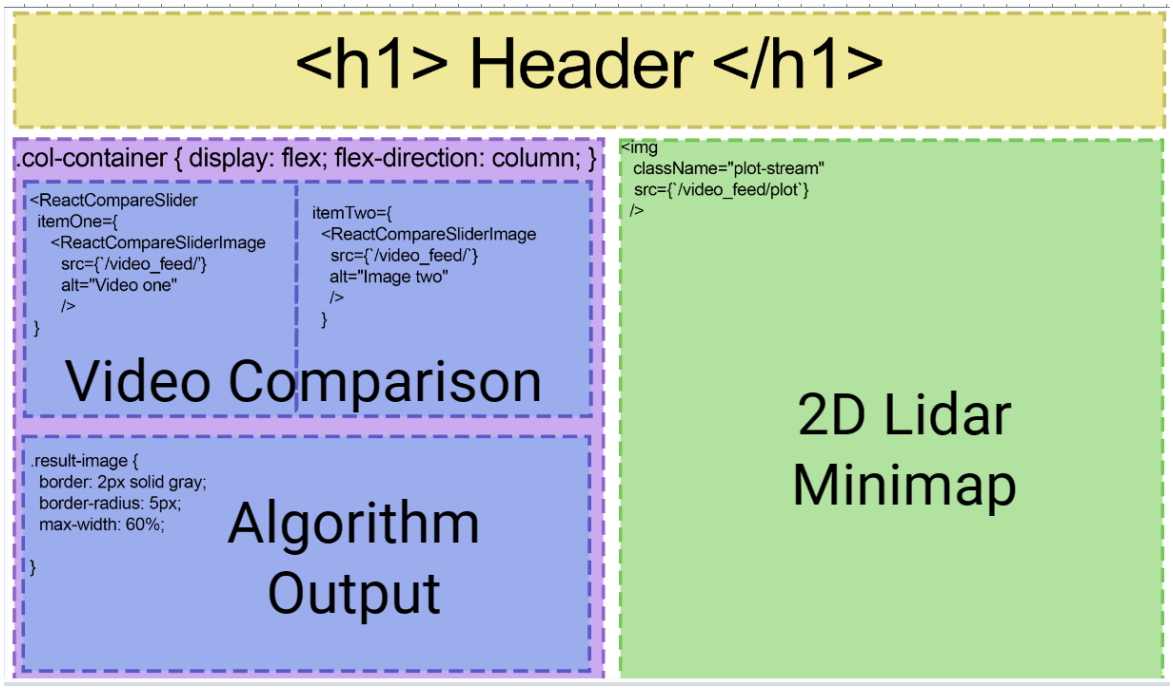
- LiDAR data and video frames from the KITTI dataset are used as input for the YOMO algorithm.
- The edge detection is used to identify edges in the video frames.
- Each frame is divided into four quadrants.
- The algorithm focuses on the third and fourth quadrants.
- The cascade classifier is used to identify the number of cars in each quadrant.
- The LiDAR data is used to compare the distances of all the objects and calculate their proximities.
- The resulting data are fused by the Multi-Layer Perceptron Classifier to determine if it is safe for the vehicle to change lanes.

3.3 Backend



- The purpose of the backend server is to create a web server to generate frames and stream video feed.
- The backend has the ability to upload files and convert them into a JSON format.
- The LiDAR data is stored in a csv (comma-separated values) and text format.
- The backend should read the data and perform the initial formatting. The server should convert LiDAR data into a dataframe structure provided from the Pandas library.
- The backend takes the dataframes as parameters to call the functions in the algorithm.
- The algorithm stores the results in a JSON format, and the algorithm provides a final result from the MLP Classifier to send to the backend server.
- The backend provided the frontend with the results from the algorithm.

3.4 Frontend



- Provides the visualization of the car's point of view.
- The frontend compiles and displays several important components of the project in the web application.
- The visual above demonstrates three important components: video comparison, algorithm output, and 2D LiDAR minimap.
- The video comparison displays a comparison slider that shows either the original or edge detected frame depending on the direction of the slide.
- The algorithm output should display a "SAFE" or "DANGER" notification depending on the surrounding objects around the vehicle.
- The 2D LiDAR minimap should display a miniature map of the moving vehicle with surrounding objects in a 2D spectrum.

4. Architectural Strategies

- **Use of a particular type of product (programming language, database, library, etc.)**
    - Python
    - HTML
    - JavaScript
    - REACT
    - Flask
    - CSS
    - Flexbox
    - Matplotlib
    - OpenCV
- **Reuse of existing software components to implement various parts/features of the system**
    - Implementing the YOMO algorithm version 1.0 from the fall semester.
- **Future plans for extending or enhancing the software**
    - Creating a mobile application that can abstract the output of the algorithm.
    - Improve the algorithm where it can assist the driver in braking.
    - Improve the visualizations for the data and outputs in an easy comprehensible way.
    - Monitor if the algorithm abides by traffic laws in the environment it is used.
- **User interface paradigms**
    - A computer with a mouse and keyboard is required to navigate the application.

5. Policies and Tactics

5.1 Choice of which specific products used

- Programming Languages:
    - Python
    - JavaScript
    - HTML
- Github - Deployment
- Discord & Zoom - Communications

5.2 Plans for ensuring requirements traceability

- The requirements of the application are traceable through Github.

5.3 Plans for testing the software

- The application will be tested constantly as new components and features are implemented.

5.4 Plans for maintaining the software

- Maintenance of the application will be conducted as soon as bugs arise.

5.5 System Deliverables

- The Web App was developed in Visual Studio and React.

6. Detailed System Design

6.1 Overall Responsibilities

Data collection, aggregation, initial pre-processing, and delivery are done by the KITTI vision benchmark suite. Model training of machine learning algorithms is done by computers and MacBooks.

6.2 Overall Constraints

The accuracy of the data collected by the sensors is crucial for the success of any algorithm that relies on it. Even the slightest errors or inconsistencies in the data can significantly impact the algorithm's output. The speed of algorithm processing is also an important factor to consider, especially when dealing with real-time applications. While some algorithms can process data in real-time, others may require more time to complete their computations. When transferring data using the HTTP protocol, there's always a risk of data loss. HTTP is a stateless protocol, meaning that it doesn't store any information about the previous interactions between the client and the server. As a result, if the connection is lost during data transfer, the data may not be fully transmitted, leading to data loss.

6.3 KITTI Vision Benchmark Suite

6.3.1 Responsibilities

The database contains the video frames and lidar data needed to train the model.

6.3.2 Constraints

The database may not cover all possible scenarios, causing the algorithm to be unable to cope with some unexpected situations on the road.

6.4 Machine Learning Algorithms

6.4.1 Responsibilities

Machine learning algorithm components are responsible for making predictions based on input data using machine learning models that are pre-trained during the development phase.

6.4.2 Constraints

Machine learning algorithms may give incorrect predictions due to noise or extreme values in the input data.

6.5 Algorithms Fusion

6.5.1 Responsibilities

The algorithm fusion component is responsible for combining the predictions of the machine learning algorithm from the sensor with the predictions of the computer vision machine learning algorithm to give the final structure.

6.5.2 Constraints

The fusion algorithm may favor a single algorithm, such as the sensor-only part or the computer vision-only part, and end up giving a biased result.

6.6 User Interface

6.6.1 Responsibilities

The user interface is responsible for providing the user with the algorithm's predictions in a timely manner and sending a warning to the user if the algorithm gives a dangerous prediction. In addition, the graphical interface will also display the current live footage of the vehicle with some other essential information.

6.6.2 Constraints

Displaying multiple scenes at the same time may have an impact on the performance of the graphics interface, resulting in lag, delay, or unresponsiveness.

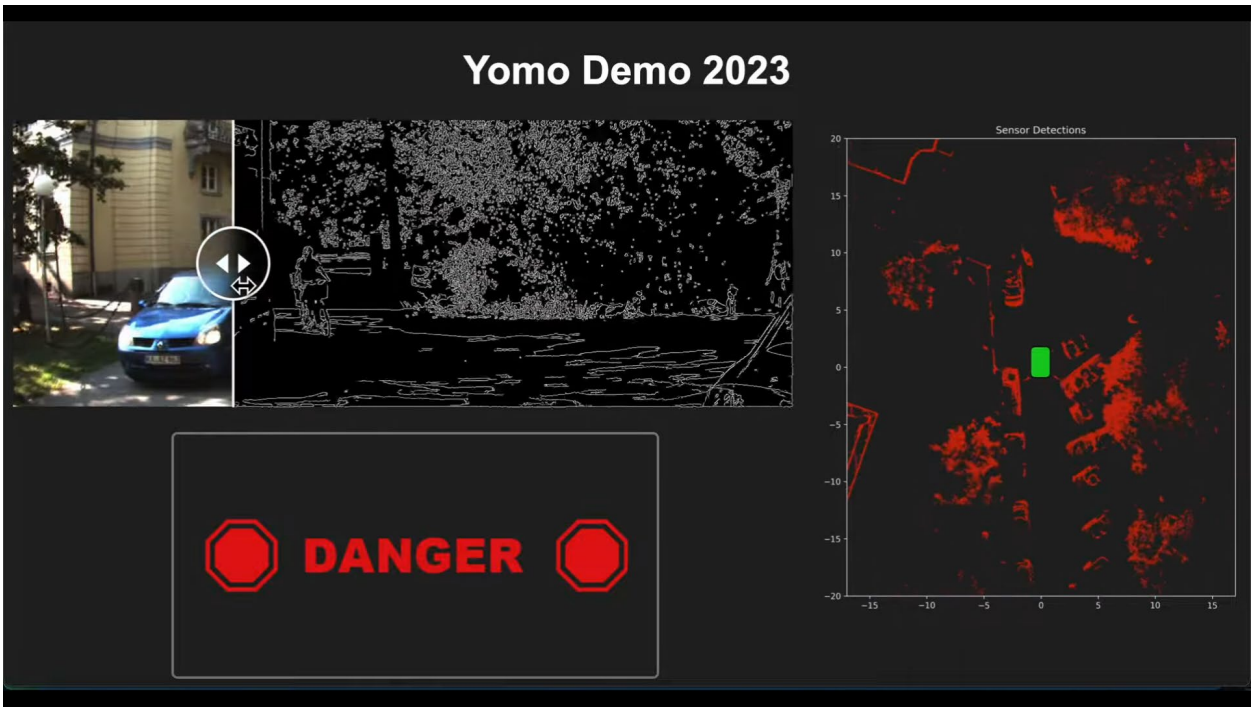
7. Developer Interface

7.1 Overview of Developer Interface

The developer interface is a simple design that should be easy to understand. The developer should be greeted with a web page featuring three components. The images in section 7.2 represent the visualizations of the machine learning algorithm. The developer should see on their top left corner a comparison slide of the original and/or edge detected video frame. The bottom left corner demonstrates a “safe” or “danger” notification that constantly changes depending on the position of the vehicle. The right side demonstrates the 2D LiDAR minimap. Developers can use the webpage for research and development purposes.

7.2 Screen Frameworks or Images

Developer Webpage:



Appendix A: Glossary

LiDAR	Light Detection and Ranging. Measures distances using lasers for the purpose of ranging objects.
Y.O.M.O	You Only Merge Once. Name of our algorithm that solves our problem regarding lane change safety.
Machine Learning	Computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyze and draw inferences from patterns in data.
Camera Sensors	Image-based sensors that provide the driver with important visual information.