Thursday, September 10, 2020

Advanced driver-assistance system on Jetson Nano Part 1 - Intro & Hardware design

1731 words 9 min read

Viet Anh
@vietanhdev

Recently, I have built a prototype of an advanced driver-assistance system (ADAS) using a Jetson Nano computer. In this project, I have successfully deployed 3 **deep neural networks** and some **computer vision algorithms** on a super cheap hardware of Jetson Nano. I decided to write this post series to share about how this system was designed and implemented. In this series, I will introduce the overall design of the system, 3 deep neural networks I used for environment analysis and some tutorials on TensorRT - the core technology to optimize neural networks for NVIDIA's system. In this post, let's get started with an introduction to my project and the hardware design of this system.

I. Introduction

1. Background and motivations

Currently, smart driver assistance functions are gradually being improved and become a new criterion in the technology race among car manufacturers. However, there are a large number of old cars and also a large number of new low-end car models without an advanced driver-assistance system (ADAS).

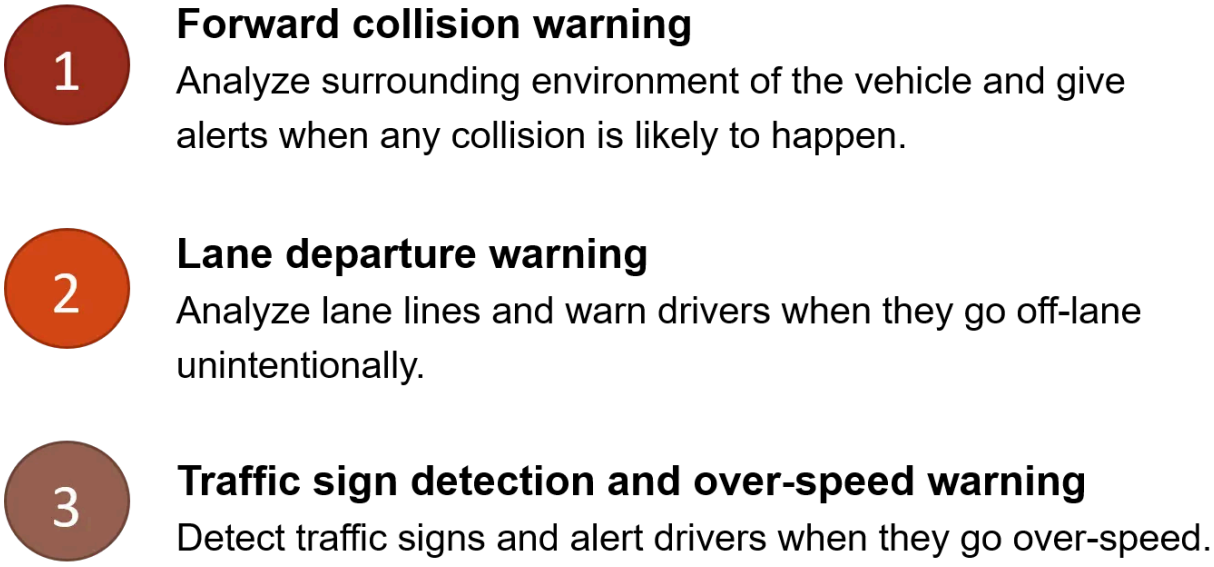
For this market, technology companies also develop separated products to setup on used car models or car models without integrated ADAS. In this type of product, **MobileEye 630** is a popular device, which is developed by MobileEye, a subsidiary of Intel. **MobileEye 630** provides intelligent features such as forward collision warning (FCW), lane departure warning (LDW), intelligent high beam control (IHC), speed limit indication (SLI), and traffic sign recognition (TSR). In Vietnam, **WebVision** is a company specializing in providing dashcam products with intelligent driver assistance technologies. **WebVision A69 AI** with camera recording function, lane departure warning, forward collision warning, and moving reminder when the traffic light turns green. **WebVision S8**, in addition to the dashcam function, also warns drivers when they go over speed.

It cannot be denied that **Intel Mobile Eye** or **WebVision** systems have reached a relatively good level of perfection. However, the **Mobile Eye 630** system, which is currently sold in Vietnam, lacks a user interface for drivers to calibrate the device easily. WebVision devices, although equipped with useful additional functions such as map navigation, however, ADAS features are only partially equipped. For example, the **WebVision A69 AI** does not have a sign recognition function, and the **WebVision S8** product lacks two important functions of an ADAS system: collision warning and lane departure warning. The traffic sign recognition feature in **WebVision S8** is also done using the map data stored in the device in combination with GPS instead of a camera, which requires frequent updates. This may not be feasible in practice, and it is not helpful when drivers drive to new areas. Through this analysis, I recognize the need for a better and more completed advanced driver-assistance system for old and low-end cars and it is my reason to develop this system.

2. Purpose and scope

The purpose of this project is to design a prototype of a completed advanced driver-assistance system targeting old and low-end cars that are not equipped with, or lack of some driver assistance functions. The implemented product should be a system with hardware and software to provide three main functions: **(i) forward collision warning with forward vehicles and pedestrians, (ii) lane analysis and lane departure warning, (iii) sign detection for maximum speed limit signs and over-speed warning.** In the scope of this project, because of the limitation in experimental conditions, I developed and used a simulation to provide camera stream and car data stream instead of using a real camera and a

real connector to connect with car electronic system. However, the completed design with a physical camera and car connector is still considered.



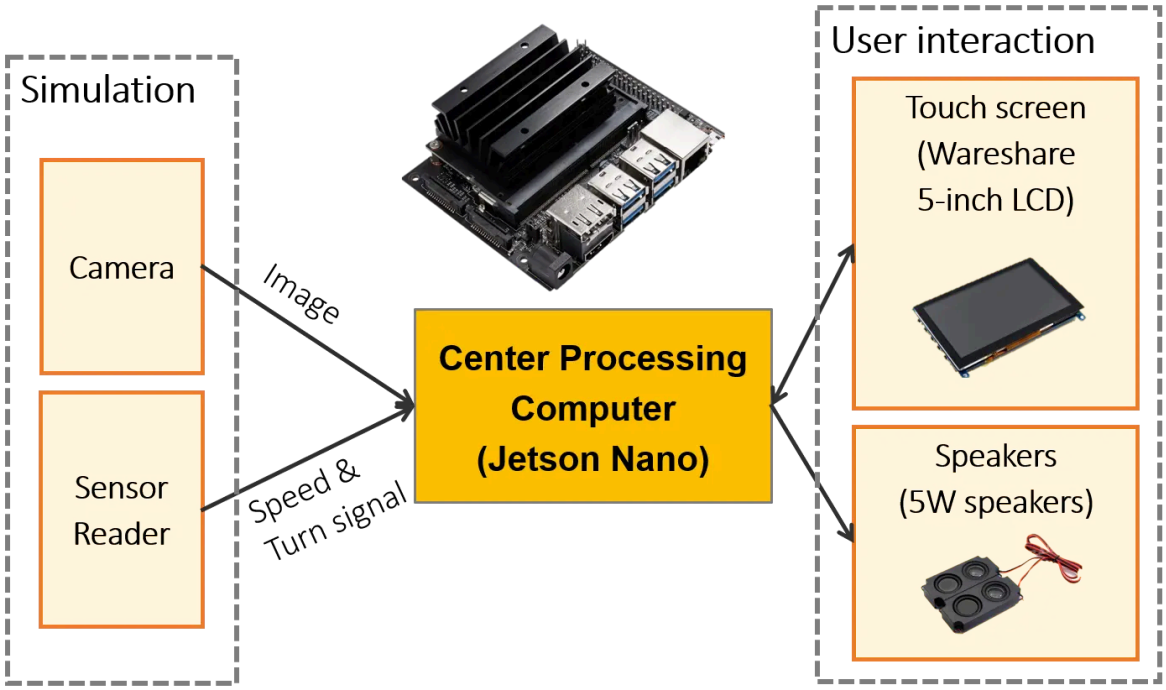
3. Solutions

For the hardware of the proposed system, I choose **NVIDIA Jetson Nano**, a small, powerful computer that lets you run multiple neural networks in parallel to deploy the final system. Jetson Nano is suitable for this project as it is a powerful hardware architecture with a cheap price to deploy deep learning models. This will keep the production cost relatively low compared to other similar systems. I also attach a 5-inch screen and two small speakers to build a user interface.

In software design, I use three neural networks to build the core of the system. For collision warning, this system uses **CenterNet** - an object detection network with **ResNet-18 backbone** to achieve a good detection speed and acceptable accuracy. Besides, perspective transform with calibration is used to estimate the distances from system vehicle to other vehicles ahead. The results from the object detection network are utilized to detect the location of traffic signs. After that, the system crops all traffic sign images and passes them through a classification network employing **ResNet-18** architecture to distinguish signs. For lane departure warning function, the combination of **U-Net** and **ResNet-18** backbone is used for lane line segmentation and Hough line transform is utilized to find lane lines. After that, the detected lines are used to identify lane departure situation using a rule-based algorithm. After training and fine-tuning, three networks are optimized to run on embedded system of **Jetson Nano** computer

using NVIDIA Tensor-RT technology. This technology from NVIDIA helps neural networks run faster with much lower memory consumption.

II. Hardware design and implementation



Images from developer.nvidia.com www.waveshare.com, https://canable.io/.

Hardware always plays an important role in any embedded system. It specified resource constraints that software has to be optimized on. The center component to process all inputs of this project is a Center processing computer. This computer receives two inputs: (i) images from a camera, and (ii) car sensor data such as car speed and turn signal. It takes responsibility to process these inputs to issue warnings when needed. In the scope of this project, due to the limited experimental condition, I implemented a simulation module to provide alternatives to the camera and the sensor reader inputs. In order to output warnings, the center processing computer is connected with a touch screen and speakers.

Below is the list of components used in this project. These components are chosen in consideration of hardware ability, size, and price. The case for the whole system is designed and finished using crystal plastic and laser-cutting technology.

- Jetson Nano Developer Kit https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

- Sandisk Ultra 64GB class-10 SD card https://www.memoryzone.com.vn/the-nho-microsdxc-sandisk-ultra-64gb-80mbs-533x-2017.
- Wareshare 5-inch LCD touch screen https://www.waveshare.com/5inch-hdmi-lcd-h.htm.
- Wareshare 8Ω 5W Speaker https://www.waveshare.com/8ohm-5w-speaker.htm.
- 2-inch 5V cooling fan for Jetson Nano
- Acrylic clear case.

Jetson Nano computer

Released in March 2019 by NVIDIA, Jetson Nano is a powerful platform for deploying machine learning algorithms. Because of a small size board with a quiet strong GPU, it is suitable to be used as the center processing computer. One special feature of this computer in comparison with ones from other companies is that it can use TensorRT, an SDK (software development kit) for high-performance deep learning inference. This SDK includes a deep learning inference optimizer and runtime for low latency and high-throughput experience. In this project, this feature can be leveraged to run deep learning networks to analyze images from dash camera. A Sandisk Ultra 64GB class-10 SD card is used as the main disk memory. The detail of system configuration of Jetson Nano is listed below:

- CPU: Quad-core ARM Cortex-A57.
- GPU: 128-core NVIDIA Maxwell architecture-based.
- RAM: 4 GB 64-bit LPDDR4; 25.6 gigabytes/second.

This blog has a post on how to configure and optimize Jetson Nano for AI. You can get started with this post (only available in Vietnamese now).

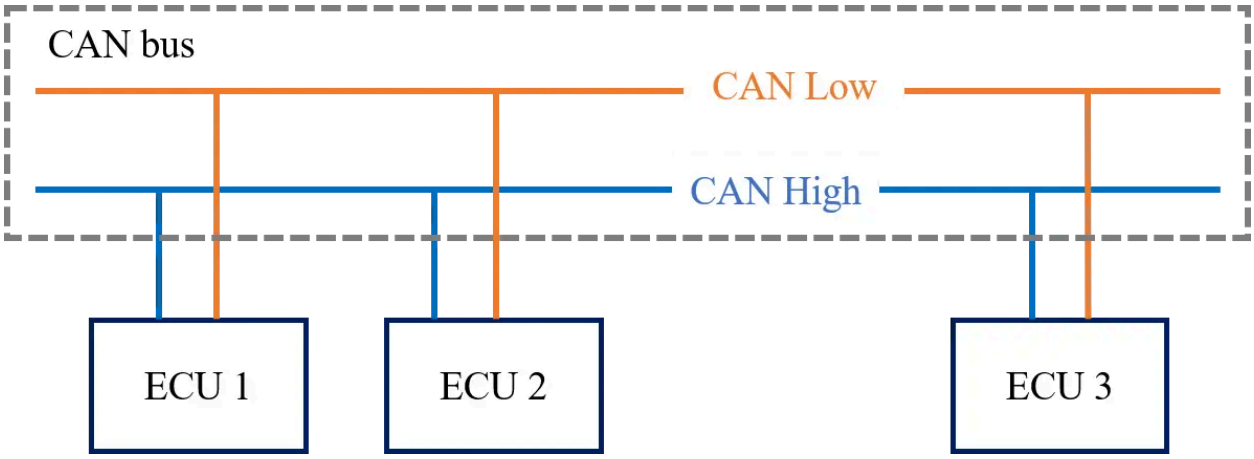
Display screen and speakers

For user interaction, Wareshare 5-inch touch screen (H model) is a good option for this project. It provides a large enough space for a comfortable user experience. Because Jetson Nano does not contain any sound card, the sound card from H model screen is a convenient way to deploy speakers. I use 5Ω - 8W

dual speaker from Wareshare to play warning and notification sounds in designed system.

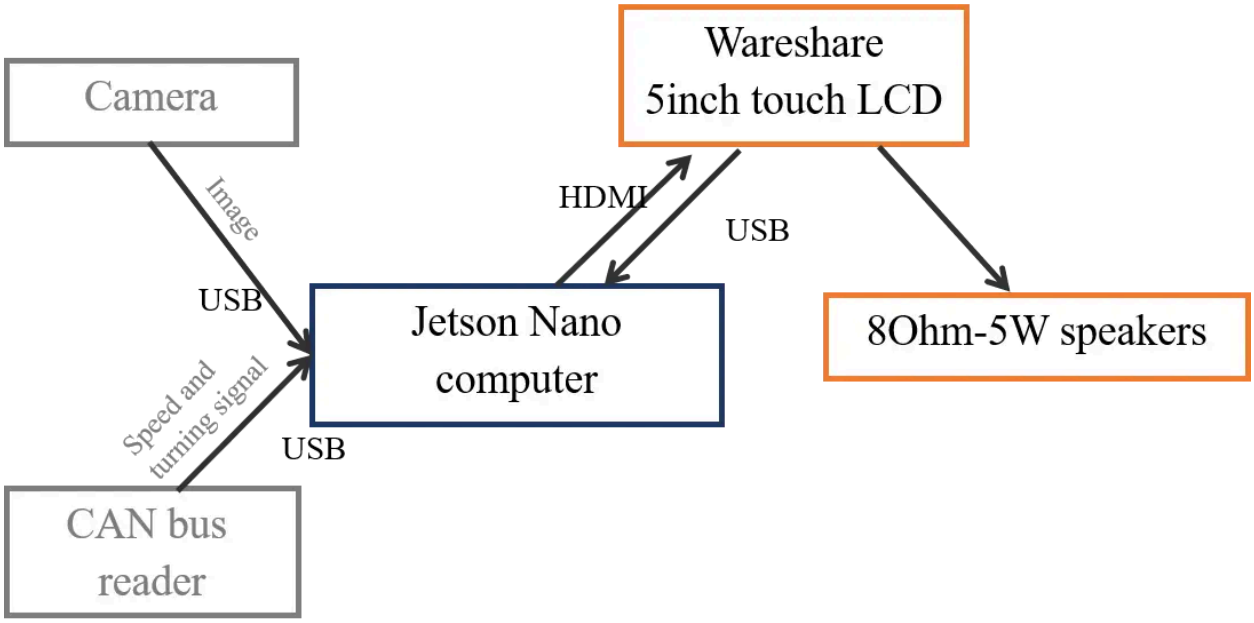
Sensor reader module

In order to determine car speed, we can use an (i) an indirect solution – estimate speed using GPS signal or (ii) a direct solution – read speed directly from the car electronic system. The first way, estimating speed from GPS can be easier and safer because we do not have to connect to the car sensing networks, which can result in incorrect interactions with car components. However, because this solution has a delay in speed estimation, we should not use it for a safety warning system. Nowadays, almost car is equipped with a Controller Area Network (CAN) as one of the main networks to exchange data between electronic components, providing us a standard way to read car sensors such as car speed or turn signal. This method is also integrated into my system.



To communicate with CAN bus of a car, we need a component called CAN bus reader, which is a USB-to-CAN adapter. However, in this project, due to the limitation in experimental condition, I only implemented a virtual CAN instead to exchange data between driving simulation and the core system.

How to connect hardware components?



At last, I want to show you a demonstration of my prototype for the system. Currently, my system can only run on video and simulated sensor streams. In the upcomming posts, I will talk more about the neural networks and the implementation of the software stack of this project.

III. Source code

1. Object detection with CenterNet

- Training code for BDD100k dataset: https://github.com/vietanhdev/centernet-bdd-data.

- Conversion code to ONNX model: https://github.com/vietanhdev/centernet-bdd-data-onnx-conversion.

2. Lane line segmentation with U-Net

- Training and conversion code to `.uff` : https://github.com/vietanhdev/unet-uff-tensorrt.

3. Traffic sign

- Training and conversion code to `.uff` : https://github.com/vietanhdev/traffic-sign-classification-uff-tensorrt.

4. Code for Jetson Nano

- Code for Jetson Nano - contains all inference code for above models: https://github.com/vietanhdev/car-smart-cam

Give me Github Star if you think it is interesting. Note that this repository does not contain the source code for training and converting AI models. ~~I'll make them public as soon as possible.~~

Update 12/10/2020: The next post is about the software of this project. Go to the next post now.

Update 15/11/2020: Add links to source code.

Load Comments

TAGS

ADAS JETSON-NANO

PREVIOUS ARTICLE NEXT ARTICLE

Hackathon: Xây dựng giải pháp biến đổi, stream giấy viết cho giáo dục từ xa

Advanced driver-assistance system on Jetson Nano Part 2 - Software design

← Back to the blog