

Dynamove: Improving Driver Productivity with Advanced Driver Assistance Systems for Trucking Fleets

A Project Report

submitted by

ROHAN GURUNANDAN RAO

in partial fulfilment of the requirements

for the award of the degree of

DUAL DEGREE (BACHELOR AND MASTER OF TECHNOLOGY)



**DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.**

MAY 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **Dynamove: Improving Driver Productivity with Advanced Driver Assistance Systems for Trucking Fleets**, submitted by **Rohan Gurunandan Rao**, to the Indian Institute of Technology, Madras, for the award of the degree of **Dual Degree (B.Tech and M.Tech)**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Radha Krishna Ganti
Research Guide
Associate Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 10th May 2019

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Prof. Radha Krishna Ganti for the continuous support during my Dual Degree study, as well as for providing motivation and encouragement for pursuing this as a startup, and also for his patience and immense knowledge. His guidance helped me throughout my project and writing of this thesis. I could not have imagined having a better advisor and mentor for my Dual Degree study.

I am also grateful to my team, a continuously varying team of students who worked with me on this idea, and in particular, the core team consisting of Abhijit, Raj, Aditya and Arjun, who provided a host of ideas to develop this further. I am also fortunate to have had the opportunity to discuss with experts in the field of transportation, from people ranging from the Govt. of India's Niti Aayog, to large companies including Waymo, Velodyne, NVIDIA, Intel, Tata Motors and Bosch, to startups like PIX Moving (China), Zoox (USA), Autoware (Japan) and accelerators like Y Combinator (USA) and Axilor Ventures (India).

I would like to thank IIT Madras for providing me outstanding education and the best professors, and for providing the resources and funding for trying to take this project further in the form of a startup. In particular, the Nirmaan pre-incubation program was of great help, apart from research funding I received as part of this project. Last but not the least, I would like to thank my family and my friends for providing emotional support during this challenge.

ABSTRACT

KEYWORDS: traffic; telematics; computer vision; FPGA; ARM; GAN

India unfortunately has one of the highest road accident rates in the world, at nearly 150,000 people killed every year, and many more injured. Cities like Bangalore and Mumbai are also notorious for their horrible traffic, leading to enormous loss of productivity as well as fuel. To solve these problems, we conceptualized, designed and developed a product and deployed it into various small and mid-sized Indian trucking fleets operating in Bangalore, Chennai and Mumbai. A brief description of the product is described below.

Dynamove is an intelligent transportation startup that is building retrofittable driver assistance and vision-based telematics systems, with a special focus on issues and constraints faced in emerging markets such as India. Our product consists of an AI-enabled dual camera device and an in-vehicle telematics module which makes road vehicles safer, smarter, connected, and more reliable. Through a host of sensors driven by cutting-edge artificial intelligence, computer vision, and behavioural modelling, our plug-and-play module helps drivers stay alert and focused, and help them drive better by coaching them based on their driving. These help in reducing fuel, maintenance, and pilferage losses while making journeys safer and drivers better.

Development of this product required a lot of iterations and pivoting, as we met different key stakeholders through events and competitions, and understood the market better. This report covers work on software defined radio communication with FPGAs, computer vision accelerators on FPGAs, autonomous car perception software development, deep learning optimization for ARM chips with binarization, image enhancement methods for cameras using generative adversarial networks (GANs), semantic segmentation for Indian road scenes, and algorithms for measuring driver alertness, distraction and productivity.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	i
ABSTRACT	ii
LIST OF FIGURES	vi
ABBREVIATIONS	vii
1 Introduction	1
2 Software Defined Radio on an FPGA	3
2.1 The FM Radio Protocol	4
2.2 The algorithm for FM Demodulation	5
2.3 Obtain IQ Samples from the RTL-SDR	5
2.4 Plot the PSD of the received signal	6
2.5 Shift and decimate signal	7
2.6 Unwrap angle and differentiate in time to get phase	8
2.7 Real-time FM processing	8
3 Computer Vision Algorithms on an FPGA	9
3.1 Grayscale Conversion	10
4 Detecting drowsiness, distraction and drunkenness with computer vision	12
4.1 Face Detection	12
4.2 Facial Feature Tracking	13
4.3 Head Pose Orientation Estimation	13
4.4 Blink Detection with Eye Tracking	13
4.5 Gaze Tracking using Region Based Thresholding	14
5 Object Detection Algorithms on Embedded Devices	15
6 Building Perception Systems for Fully Autonomous Cars	21

6.1	Move-It Hackathon, March 2018	21
6.2	DIY Robocars KuaiKai, May 2018	23
7	Image Enhancement methods for Camera Systems	29
8	Generating and training on Indian datasets	31
9	The final product: An Alerting and Monitoring System to Improve Driver Productivity	37
9.1	Overall System Design Patent	38
9.2	Road Facing Camera Patent	39
9.3	Spatiotemporal Deblurring Patent	39
9.4	Driver Facing Camera Patent	39
9.5	Web Architecture	40

LIST OF FIGURES

2.1	An illustration of the architecture of the RTL-SDR.	3
2.2	The SDR# software for Windows that can capture and demodulate incoming RF signals and display/play the FM radio on a computer. . .	4
2.3	The set of steps involved in demodulating the FM Radio signal in Python.	5
2.4	Power spectral density of an FM signal around 97.4 MHz	7
2.5	The central frequency lobe for the FM radio signal	7
3.1	The software architecture of the PYNQ-Z1 FPGA	9
3.2	The base overlay for the PYNQ-Z1 with AXI, HDMI and peripherals	10
4.1	Detecting drowsiness after auto-calibration	14
4.2	Tracking facial features, head pose, eye gaze and pupil dilation, to detect drowsiness and drunkenness	14
4.3	Different device orientations/positions and with varying amounts of vibration, on India's roads	14
5.1	Image Classification, Object Detection (single object), Object Detection (multiple objects) and Instance Segmentation	15
5.2	Object Detection, Semantic Segmentation and Instance Segmentation	16
5.3	The Jetson Tegra SoC block diagram with integrated NVIDIA Pascal GPU, NVIDIA Denver 2 + ARM Cortex-A57 CPU clusters, and multimedia acceleration engines	17
5.4	End-to-end AI Pipeline including sensor acquisition, processing, command and control	17
5.5	Performing a convolution of binary input and binary weights using an XNOR-bitcount operation	19
6.1	Autoware Software Stack: Sensing, Computing and Actuation . . .	21
6.2	The Velodyne LIDAR Ultra Puck (VLP-32C), with 32 channels . .	22
6.3	Data collection for the Chinese Police Gesture Dataset	23
6.4	Start grid detection algorithm	24
6.5	Processed grid detection output	25
6.6	Sign detection algorithm	25

6.7	Object detection for pedestrians and cars	26
6.8	Redundancy for the different vision tasks	27
6.9	Car following algorithm using size of bounding box and tracking . .	27
6.10	Camera+LIDAR on the front	28
6.11	The insider view	28
6.12	Navigating the S-Bend	28
7.1	Sample Deblurred Images. Arranged from left to right as blurred, de- blurred and ground truth.	30
8.1	AutoNUE Data Acquisition Setup	31
8.2	L to R: example image, prediction with pretrained model, prediction after training on this dataset, ground truth	32
8.3	Output Image 1	33
8.4	Output Image 2	33
8.5	Output Image 3	34
8.6	Example 1: Input US/Europe road -> AI Generated Indian Road . .	34
8.7	Example 2: Input US/Europe road -> AI Generated Indian Road . .	35
8.8	Example 3: Input US/Europe road -> AI Generated Indian Road . .	35
8.9	Example 4: Input US/Europe road -> AI Generated Indian Road . .	36
9.1	The first prototype of the device	37
9.2	Overall System Design	38
9.3	Road-Facing Camera ADAS Design	39
9.4	Driver Behaviour Monitoring System Design	40
9.5	A Web-Based Fleet Dashboard	41

ABBREVIATIONS

FPGA	Field Programmable Gate Array
SDR	Software Defined Radio
HLS	High Level Synthesis
GAN	Generative Adversarial Network
CNN	Convolutional Neural Network
GPU	Graphics Processing Unit
NPU	Neural Processing Unit
V2X	Vehicle to Everything
ADAS	Advanced Driver Assistance Systems

CHAPTER 1

Introduction

This project began around May 2017. Flipkart, one of India's largest e-commerce companies, which was recently acquired by Walmart, conducted a Gridlock (2019) Hackathon as part of their 10th Anniversary Celebrations. They were interested in solutions to ease or eliminate Bangalore's horrible traffic.

I found a team of 5 others who were interested in participating in this competition, and we started discussing ideas under the team name Dynamove (2019). I came up with the name as an amalgamation of the words dyna and move, since I was interested in using reconfigurable hardware (FPGAs) for improving mobility.

The initial idea was to build Vehicle to Everything (V2X) systems using Software Defined Radio (SDRs) for FPGAs. Since they would be reconfigurable, we could change the frequency and support various protocols, and enabling vehicles to communicate with each other at traffic intersections and on highways was proven to reduce stop-and-go traffic significantly.

Within a few weeks, we expanded the V2X idea to include Advanced Driver Assistance Systems (ADAS) on FPGAs. This ADAS would include V2X features too, and everything would run on the same FPGA. The first steps were to implement basic IQ modulation and demodulation using an off-the-shelf SDR, in parallel with some image processing (like lane detection) on the same FPGA board.

As I had prior experience with the Xilinx Zynq series FPGAs, and since they also had ADAS as one of the suggested applications, I decided to go ahead with them. We chose to use the Digilent PYNQ-Z1 (2019) for our testing, which was available at a student discount of \$65 and had a Zynq ZC7020 processor on-board. It also had support for an operating system that would allow me to access FPGA overlays via Python functions. They provided a base overlay with access to all major peripherals, and it was possible to modify this to add custom accelerators or blocks into the design.

We were also looking into building a Software Defined Radio by using open source

projects like the HackRF (2019). We ordered high bandwidth ADCs and DACs, FPGAs, and began designing a PCB. We also wanted to build a full infotainment system by integrating this with an augmented reality heads-up display (ARHUD). We subsequently decided to go ahead with the Analog Devices AD9361 (2019) RF front-end that would connect to the FPGA via an FMC connector and handle the RF waveform acquisition, allowing the FPGA to perform the processing. For testing, we also used the RTL-SDR (2019) USB based module that had limited range and bandwidth, but was good for prototyping.

In short, this project began as a result of an interest in FPGAs, and the expectation that we could integrate every aspect of intelligent transportation onto a single FPGA with some peripherals, including all features of Advanced Driver Assistance Systems (ADAS), Vehicle to Everything (V2X) Communication, an Augmented Reality Heads Up Display (ARHUD) and even a predictive maintenance/fuel optimization feature.

Endless iteration and prototyping, as well as discussion with key stakeholders over the next two years, led us to refine our original idea, while also being awarded special mentions, awards and accolades at a number of national and international events. These included Flipkart's Gridlock (2019) Hackathon, the Rajasthan (2017) Hackathon 2.0, Shaastra's Startup Wars, KPIT Sparkle (2018), the India Innovation Growth Programme 2.0, two autonomous car competitions in China, the European Conference for Computer Vision, Niti Aayog's MOVEHack (2018), and the Pioneer (2018) Fellowship. We also published 2 papers at the DSP (2018) in Vehicles conference at Nagoya University in Japan and filed 4 patents as part of this work.

CHAPTER 2

Software Defined Radio on an FPGA

The RTL-SDR (2019) is a very cheap \$25 USB dongle that can be used as a computer-based radio scanner for receiving live radio signals within a limited area. Most of the software and drivers required for demodulation are open source and available free of cost. The device has a frequency range of 50 MHz to 2.2 GHz, and a maximum sampling rate of 3.2 MSPS. It has a programmable RF down-converter, and so can be used as a wide band radio scanner, and also receive/play a limited set of signals.

The architecture of the device is illustrated below. As described above, it has a wide band programmable RF front-end, which can capture a bandwidth of around 3.2MHz, convert it into a digital signal, and transfer the raw data via USB to a PC for further processing.

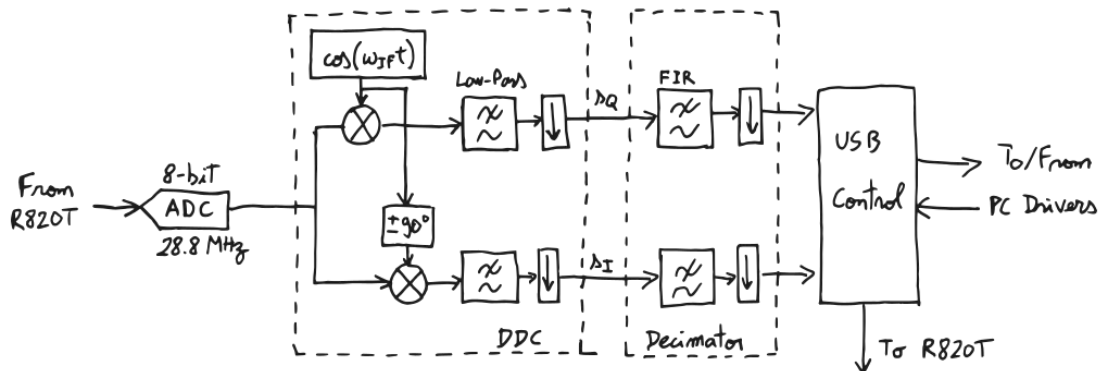


Figure 2.1: An illustration of the architecture of the RTL-SDR.

The following figure is a free software + driver toolkit called SDR# for Windows that provides a GUI and functionality to use the RTL-SDR module for scanning the frequency range, and playing AM/FM radio.

All the demodulation and processing of the raw I-Q samples is being performed on the computer itself, and then routed to the audio devices. A number of different features are included in the software, including an FFT, recording, noise reduction, and more.

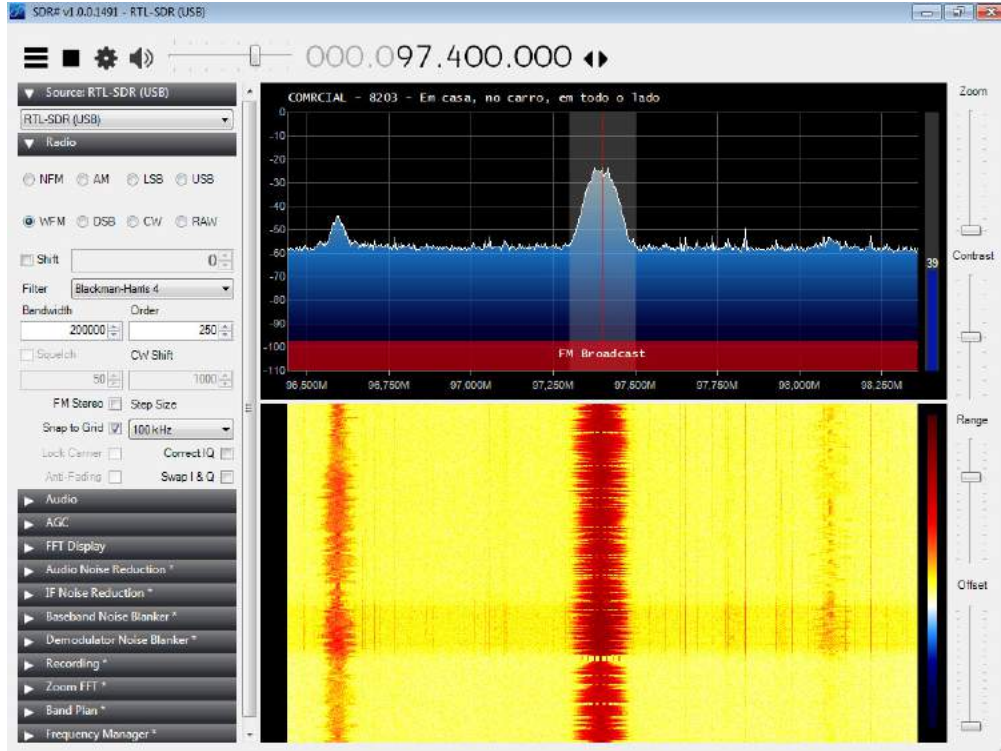


Figure 2.2: The SDR# software for Windows that can capture and demodulate incoming RF signals and display/play the FM radio on a computer.

2.1 The FM Radio Protocol

In Frequency Modulation (FM), the message signal is encoded in the instantaneous frequency of the carrier waveform. This takes on the following form:

$$\begin{aligned}
 s(t) &= A \cos[2\pi \int_{-\infty}^t f_c + f_{\Delta} m(\tau) d\tau] \\
 &= A \cos[2\pi f_c t + 2\pi f_{\Delta} \int_{-\infty}^t m(\tau) d\tau]
 \end{aligned} \tag{2.1}$$

Thus, for the baseband signal $s_b(t)$, the magnitude is constant, but the phase carries information.

$$\begin{aligned}
 |s_b(t)| &= \frac{A}{2} \\
 \angle s_b(t) &= 2\pi f_{\Delta} \int_{-\infty}^t m(\tau) d\tau
 \end{aligned} \tag{2.2}$$

And so, differentiation of the unwrapped phase of the signal will give us the message signal that is being transmitted.

$$\frac{d\angle s_b(t)}{dt} = 2\pi f_{\Delta} m(t) \quad (2.3)$$

2.2 The algorithm for FM Demodulation

We wanted to implement the same basic functionality in Python, then in C, so that the computationally intensive portions could be transferred to the FPGA fabric and accelerated there. Once the RTL-SDR driver and Python library were installed, the FM demodulation algorithm involved the following steps.

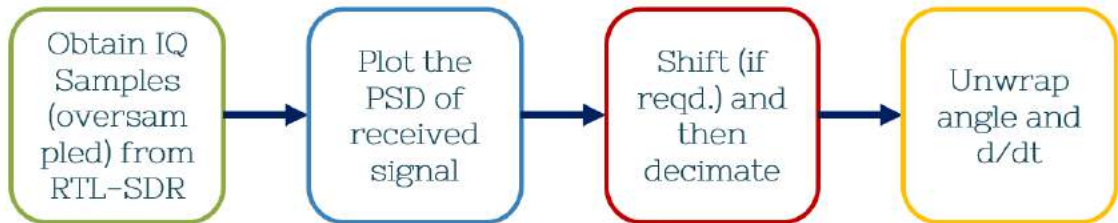


Figure 2.3: The set of steps involved in demodulating the FM Radio signal in Python.

2.3 Obtain IQ Samples from the RTL-SDR

In 2008, Eric Fry (Osmocom (2018)) wrote software and libraries that could convert a TV tuner chip like the one inside the RTL-SDR, into a software defined radio. Compiling and installing the required libraries in Linux allows accessing the raw I-Q samples via Python as follows.

```

from rtlsdr import RtlSdr
from contextlib import closing

#we use a context manager that automatically calls .close()
#on sdr when the code block finished successfully
#or an error occurs
#initializing an RtlSdr instance automatically calls open()
with closing(RtlSdr()) as sdr:

```

```
sdr.sample_rate = sample_rate = 2400000
sdr.center_freq = 97.4e6
sdr.gain = 20
iq_samples = sdr.read_samples(10*sample_rate)
```

2.4 Plot the PSD of the received signal

Once the raw I-Q samples have been obtained as described in the code snippet above, the power spectral density (PSD) can be plotted to ensure that there is a radio channel in that frequency band.

```
from scipy import signal
from scipy.fftpack import fftshift
import matplotlib.pyplot as plt

#computer Welch estimate without detrending
f, Pxx = signal.welch(iq_samples, sample_rate, detrend=lambda
    x: x)
f, Pxx = fftshift(f), fftshift(Pxx)

plt.semilogy(f/1e3, Pxx)
plt.xlabel("f [kHz]")
plt.ylabel("PSD [Power/Hz]")
plt.grid()

plt.xticks(np.linspace(-sample_rate/2e3, sample_rate/2e3, 7))
plt.xlim(-sample_rate/2e3, sample_rate/2e3)
```

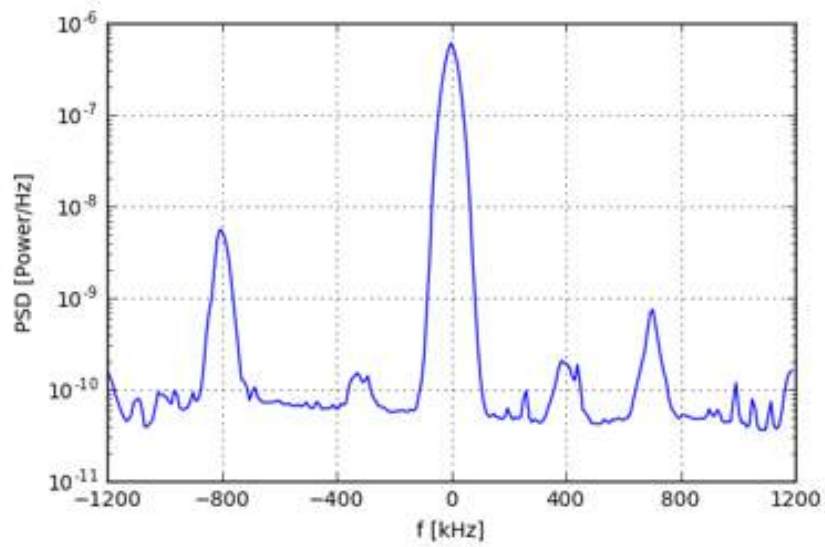


Figure 2.4: Power spectral density of an FM signal around 97.4 MHz

2.5 Shift and decimate signal

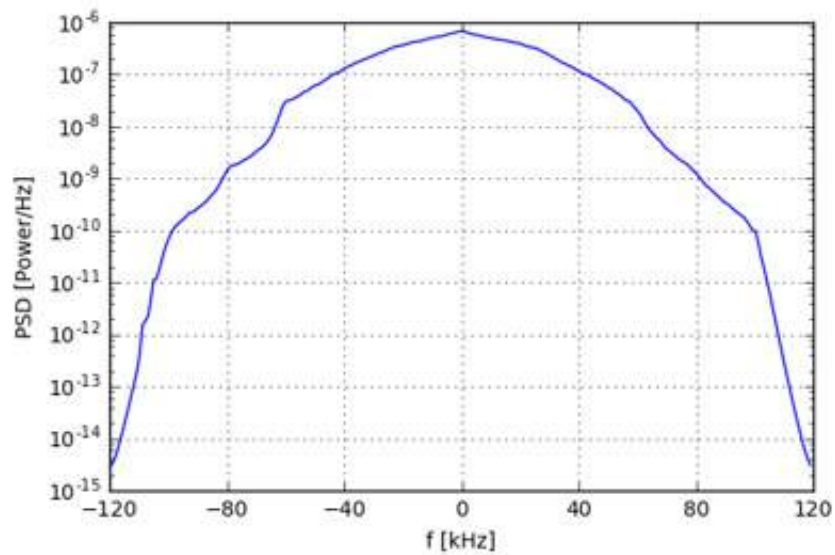


Figure 2.5: The central frequency lobe for the FM radio signal

2.6 Unwrap angle and differentiate in time to get phase

```
angle_commercial = np.unwrap(np.angle(iq_commercial))  
demodulated_commercial = np.diff(angle_commercial)
```

2.7 Real-time FM processing

The code described in the sections above was reading a 10 second stream into a buffer into memory, and then performing the processing. Doing this in real-time would require an asynchronous streaming read function, which was available in the `rtlsdr` library. Having obtained the samples, the most intensive part of the processing was to differentiate the unwrapped phase of the baseband signal to obtain the message. This was the part implemented on the FPGA in the form of an accelerator, and then Python functions were implemented to access the memory locations to send data, initiate the processing, and receive the output.

CHAPTER 3

Computer Vision Algorithms on an FPGA

For the purpose of optimizing different ADAS algorithms to run on a low-cost embedded system, we tried to implement software based computer vision algorithms on the FPGA fabric. We used the PYNQ-Z1 (2019) FPGA board, from Digilent, for which Xilinx provided a Python-based overlay with an architecture as described below:

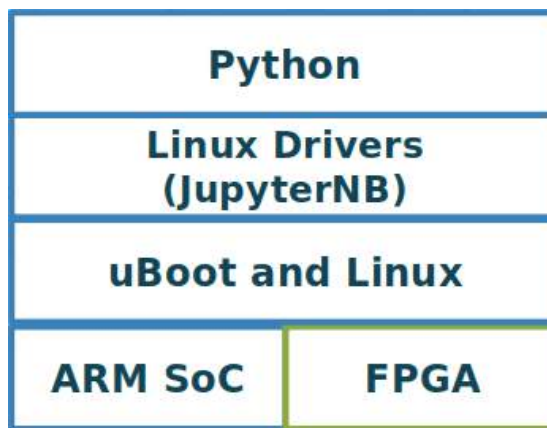


Figure 3.1: The software architecture of the PYNQ-Z1 FPGA

The ARM dual-core processor is coupled with FPGA fabric on the Xilinx Zynq series SoCs. The FPGA is connected to various peripherals, including the HDMI, the memory, and high speed interfaces like PCIE. The CPU is connected to the fixed peripherals like USB, I2C and Ethernet. The CPU runs a Linux kernel with uBoot, and there are drivers that connect from the CPU to the FPGA fabric. The OS runs a Jupyter Notebook where we can write high-level Python code to interface with accelerators on the FPGA.

The FPGA base overlay (below) has a set of basic IP blocks that are designed so that the FPGA can be used with basic GPIO and peripheral usage functionality straight out of the box. This base overlay can be modified to include additional accelerators into any of the paths.

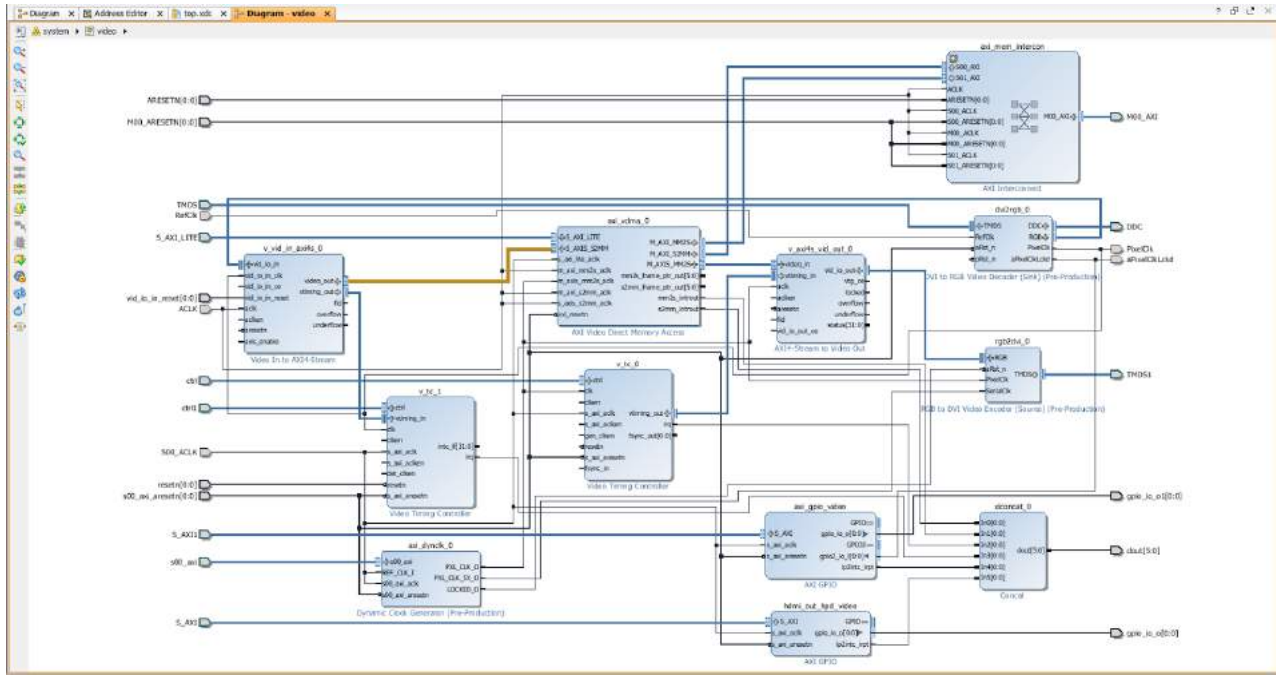


Figure 3.2: The base overlay for the PYNQ-Z1 with AXI, HDMI and peripherals

3.1 Grayscale Conversion

Running grayscale conversion is a necessary stage of any edge detection, thresholding, or corner detection pipeline. All of these operate on gradients, which can only be calculated with a single channel of colour. The code below implements grayscale conversion naively in software, using a nested-for-loop. This takes nearly 50 seconds per frame!

```
for y in range(0, height):
    for x in range(0, width):
        offset = 3 * (y * MAX_FRAME_WIDTH + x)
        gray = round((0.299*frame_i[offset+2]) + \
                    (0.587*frame_i[offset+0]) + \
                    (0.114*frame_i[offset+1]))
        frame_i[offset:offset+3] = gray,gray,gray
```

Running this same grayscale conversion on the board using OpenCV compiled onto the CPU using the NEON Floating Point Unit (FPU) on the ARM processor and other optimization directives, provides a simple method for Canny edge detection as follows. This has a latency of 1 second per frame. Much faster, but still not nearly real-time for ADAS applications.

```
import cv2  
frame-canny = cv2.Canny(np-frame, 100, 110)
```

Using the Xilinx Vivado HLS Suite, I was able to add the Sobel Edge detection accelerator into the HDMI pipeline of the PYNQ. This utilized the FPGA adequately and was able to run with a latency below 15ms on a 1920x1080 size image, meaning it could run faster than 60fps on a full HD video stream.

The compute requirements for 1920 x 1080 x 60fps is equivalent to that required for 12 cameras of size 640 x 480 running at 30fps. This is important for ADAS as it proves that we can place cameras all around the vehicle and process multiple streams simultaneously.

Similarly I was also able to implement the FASTX Corner Detection algorithm and place the accelerator into a similar pipeline to run the full HD stream in real-time.

Having implemented these basic functionalities, I wanted to add the feature to detect different obstacles on the roads, which required object detection algorithms to be optimized to run on a resource-constrained and memory bandwidth limited device like an FPGA. This involves algorithms like YOLO and Mobilenet SSD, along with various types of optimization.

But before I implemented those, I decided to focus on the driver side camera, and try to implement algorithms to detect his state of alertness.

CHAPTER 4

Detecting drowsiness, distraction and drunkenness with computer vision

In Winter 2017, I worked on building computer vision algorithms to monitor a person's face and identify if he is distracted, drowsy, or drunk. This requires the following set of steps, which are detailed in the sections below:

- Face Detection
- Facial Feature Tracking
- Head Pose Orientation Estimation
- Blink Detection with Eye Tracking
- Gaze Tracking using Region Based Thresholding

4.1 Face Detection

For face detection, there are three major methods for detecting faces in an image, which tradeoff speed for accuracy. The fastest, but least accurate, is the Haar Cascade based face detector introduced by Viola and Jones (2001). The second method is more accurate but slower, and uses Histogram of Oriented Gradients (HOG) features and a linear Support Vector Machine (SVM) as the classifier. Finally, the most accurate method in recent times has been the Convolutional Neural Network (CNN) based models, which take an image as input and return the bounding boxes for all faces in the image. These run the slowest, especially without hardware like GPUs. Fortunately, most of this functionality is included either in the OpenCV library, or in the dlib (King (2009)) library written by Davis King. So we did not need to implement it ourselves, only had to tradeoff the different methods.

4.2 Facial Feature Tracking

In the Computer Vision and Pattern Recognition (CVPR) conference in 2014, a paper titled "One Millisecond Face Alignment with an Ensemble of Regression Trees" (Kazemi and Sullivan (2014)) provided an extremely fast and accurate method for identifying 68 facial features on a person's face, given the face (so the pre-requisite was to do face detection). Fortunately, dlib (King (2009)) had an implementation of this as well, so it was just a single function call to implement it.

4.3 Head Pose Orientation Estimation

In this step, we need to use the facial features identified in 2D, and project them onto a generic 3D model of a head. This gives us the head in a world coordinate frame. Finally, we also make some approximations for the intrinsic parameters of the camera, and assume no distortion. This lets us solve the matrix equation using the Direct Linear Transform (DLT) and the implementation in OpenCV uses the solvePnP function for pose estimation. This gives us a 3D vector pointing perpendicular to the face. From this we can project onto the 2D image to show it on the screen. The website Mallick (2019) was useful for implementing this.

4.4 Blink Detection with Eye Tracking

In this step, we take the features that correspond to the eyes and calculate a parameter called the Eye Aspect Ratio (EAR), which is the ratio of the maximum vertical eye distance and the maximum horizontal eye distance. This EAR value is then tracked over multiple frames, and if below a certain threshold for a sufficient duration, the driver is considered to be drowsy. In addition, we also built in an adaptive model that would modify the threshold for different people by averaging over the first minute of driving. Here, the website Rosebrock (2019) was useful for the implementation.

4.5 Gaze Tracking using Region Based Thresholding

In this step, we used the eye gaze to provide additional information to the algorithm, and also because drunkenness can only be detected with computer vision by identifying slow gaze movements. The eye gaze was classified into one of 9 regions, by splitting the field of view right in front of the head orientation. This meant that a person could be looking to the left, but if his eyes were facing far right, that means he was looking at the road. It was a higher risk situation (not fully focused on the road), but neither is the driver completely distracted.



Figure 4.1: Detecting drowsiness after auto-calibration

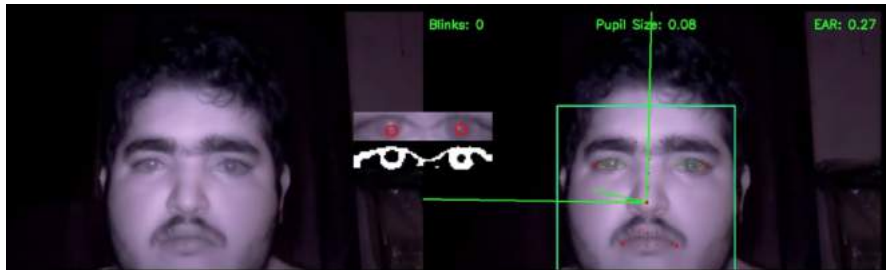


Figure 4.2: Tracking facial features, head pose, eye gaze and pupil dilation, to detect drowsiness and drunkenness



Figure 4.3: Different device orientations/positions and with varying amounts of vibration, on India's roads

CHAPTER 5

Object Detection Algorithms on Embedded Devices

In computer vision, there are a few types of tasks that require to be distinguished for further discussion.

- **Image Classification:** Given an image containing a single object or scene, identify the class to which it belongs. This generally involves training of a classifier to differentiate between two or more classes of images in a multi-dimensional space. Before 2012, this was done by extracting hand-picked features and passing them to the classifier. Since 2012, excellent results have been obtained by different types of neural network architectures, starting from AlexNet (Krizhevsky *et al.* (2012)) in 2012, and going all the way up to ResNet-152 (He *et al.* (2016)) in 2015.
- **Object Detection:** Given an image, identify all the different objects and also classify them into different categories. Before 2013, this involved splitting an image into many different regions and passing them all through a classifier. Since 2013, there have been various methods that improved the region proposal, feature extraction and classifier stages of the pipeline. Finally in 2016, a paper titled "You Only Look Once (YOLO)" (Redmon *et al.* (2016)) replaced the entire pipeline with a single end-to-end convolutional neural network. Since then, various papers have improved significantly using deep learning.

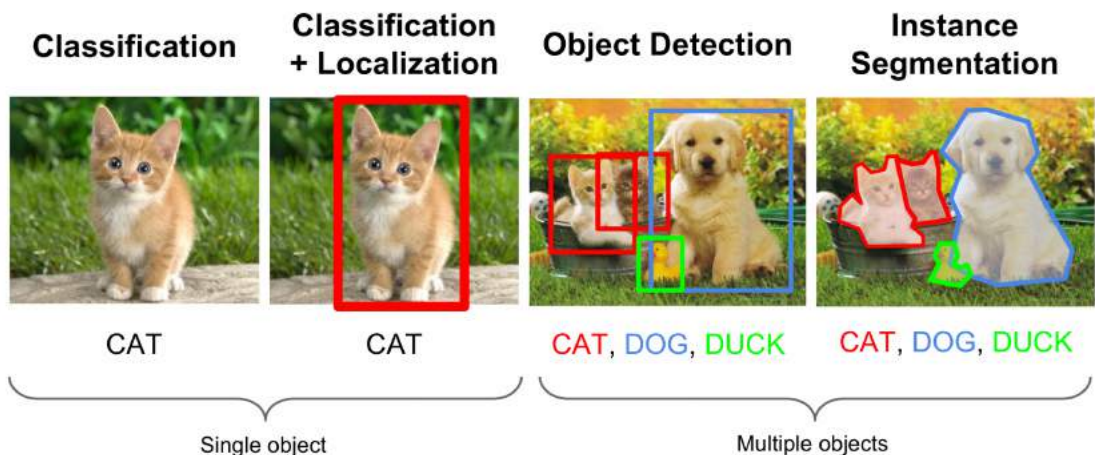


Figure 5.1: Image Classification, Object Detection (single object), Object Detection (multiple objects) and Instance Segmentation

- **Image Segmentation:** Given an image, classify each and every pixel of the image into one of two or more classes. Once again, before the advent of deep learning, it was common to use different classical computer vision methods, including K-Means clustering (Lloyd (1982)), region growing (Adams and Bischof (1994)),

and the famous normalized cuts method (Shi and Malik (2000)). Now with deep learning, we use convolutional neural networks to classify every pixel into the respective class. This would be called semantic segmentation as there is no difference between instances of the same class. However, there are also techniques for instance segmentation, which distinguish between different instances of the same class.

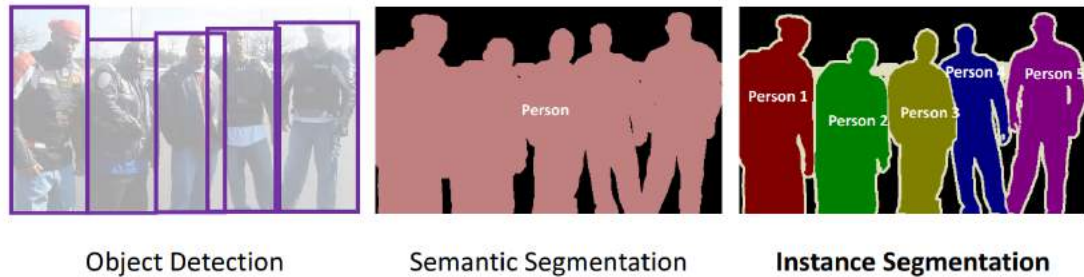


Figure 5.2: Object Detection, Semantic Segmentation and Instance Segmentation

The latest methods for object detection involved the use of deep convolutional neural networks (CNNs), and resulted in an increase of mean Average Precision (mAP) scores from 40% to over 80%. The algorithms of Girshick et al from Microsoft - RCNN (Girshick *et al.* (2014)), Fast RCNN (Girshick (2015)) and Faster RCNN (Ren *et al.* (2015)), took 20 seconds (0.05 FPS), 2 seconds (0.5 FPS) and 150 ms (7 FPS) per image on powerful NVIDIA GPU hardware. They had separate stages for region proposals and for classification, making them slower. YOLO by Redmon *et al.* (2016) described an end-to-end approach and was able to run an image within 22 ms (or 45 FPS) on similar GPU hardware.

GPUs were well suited to these initial CNN approaches to solve the problem. They had high speed interconnects and high memory bandwidths of over 300 GBps. The weights file for the first version of YOLO was over 750MB. Transferring any part of this for processing on an FPGA like the PYNQ-Z1 (2019) was not at all feasible. It had just 630KB of Block RAM (BRAM) and just 512 MB of system RAM. The memory bandwidth of the PYNQ-Z1 was below 120MBps. It would take more time to transmit the weights per image than to actually perform the calculations.

Fortunately, NVIDIA also released embedded GPU boards, under the name Jetson. The Jetson-TX2 (2019) series boards were powerful hexa-core ARM systems, with connected 256-core GPU and 8GB of shared RAM. It was possible to use the GPU

with small modifications of the code and a custom compiler and set of libraries called CUDA (Compute Unified Device Architecture - NVIDIA (2019)) released by NVIDIA.

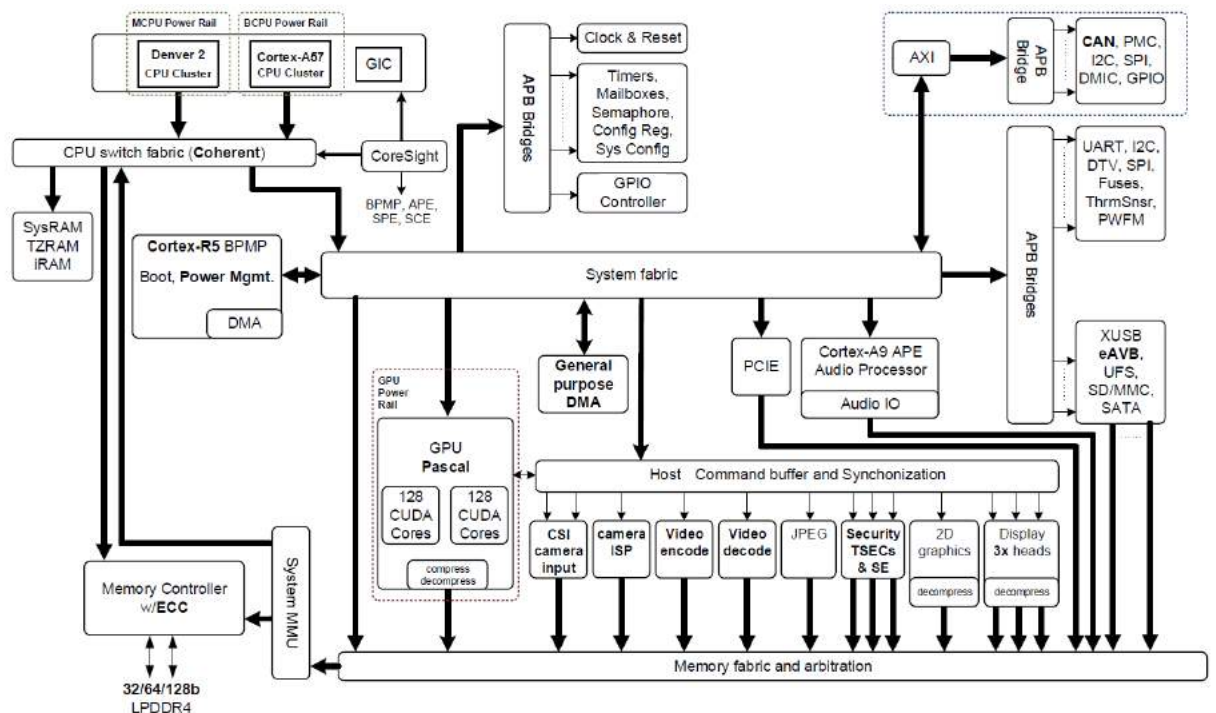


Figure 5.3: The Jetson Tegra SoC block diagram with integrated NVIDIA Pascal GPU, NVIDIA Denver 2 + ARM Cortex-A57 CPU clusters, and multimedia acceleration engines

The AI pipeline (NVIDIA (2018)), from sensor acquisition to control looks as follows. For the YOLO "darknet" object detection algorithm, the input was either via MIPI CSI camera (included with the developer kit), or via a USB-3.0 camera. It was then processed using CUDA (without cuDNN) and displayed on screen using OpenCV bindings for C++.

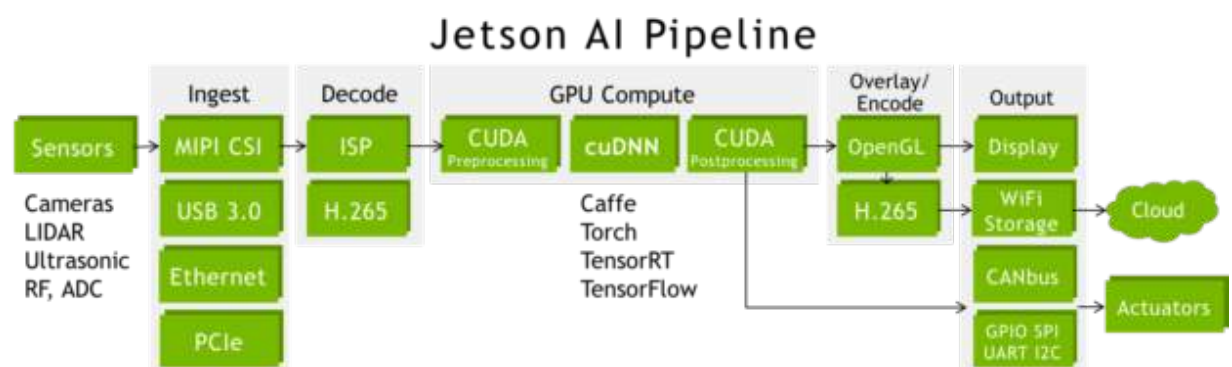


Figure 5.4: End-to-end AI Pipeline including sensor acquisition, processing, command and control

The shared RAM of 8GB was used by the CPU as well as the GPU. The final inferencing speed for the algorithm was around 7 FPS. There was also a less-accurate, higher framerate version of YOLO, called tinyYOLO. On the Jetson, it ran at 16 FPS.

The FPGAs which were being used to demonstrate such object detection tasks were the Zynq Ultrascale FPGA boards, costing over \$3000 and would only run these algorithms at around 10 FPS. On the other hand, the Jetson cost "only" \$600 for the developer kit.

We demonstrated the YOLO algorithm on the Jetson TX2 (with object tracking for increasing speed to 10 FPS) and the FPGA based software-defined radio FM processing at the KPIT Sparkle (2018) 2018 Grand Finale, and we were placed in the top 10 out of 1500 national teams. However, we were also told by many experts that even the \$600 cost is prohibitively expensive for deployment in Indian vehicles, and we need to reduce the costs. Also, software defined radio may be too futuristic/expensive and will not sell in India.

We began researching methods to optimize the object detection algorithms for FPGAs and other embedded systems. A PhD student at Stanford, now Professor at MIT, Song Han, took a lecture as part of the CS231 course, titled "Efficient Methods and Hardware for Deep Learning". This described four types of efforts - algorithms for efficient training and inference, and hardware for efficient training and inference. We were particularly interested in optimizing inference latencies, and so the algorithms/architectures for efficient inference were most interesting. The algorithms for efficient inference are listed below:

- Pruning
- Weight Sharing
- Quantization
- Low Rank Approximation
- Binary/Ternary Net
- Winograd Transformation

Of these, we tried quantization with 8-bit and 16-bit weights, especially useful for fixed point math operations on the FPGA, but not as useful without software support

on the GPU. In fact, with the GPU, the quantized operations did not lend to any difference in speed, though the weight files were of a reduced size (hence reduced memory access time, but with such high bandwidth memory, it didn't translate to any visible performance gains). We also tried binary and ternary nets, using the Xilinx library for PYNQ called BNN-PYNQ (Umuroglu *et al.* (2017)) (binarized neural networks). This supported simple networks for classification, but none for object detection.

The authors of the YOLO paper came up with a new method (Rastegari *et al.* (2016)) that utilized the advantages of binarized networks and also a technique to improve the speed of inferencing, even on highly constrained embedded systems, described below:

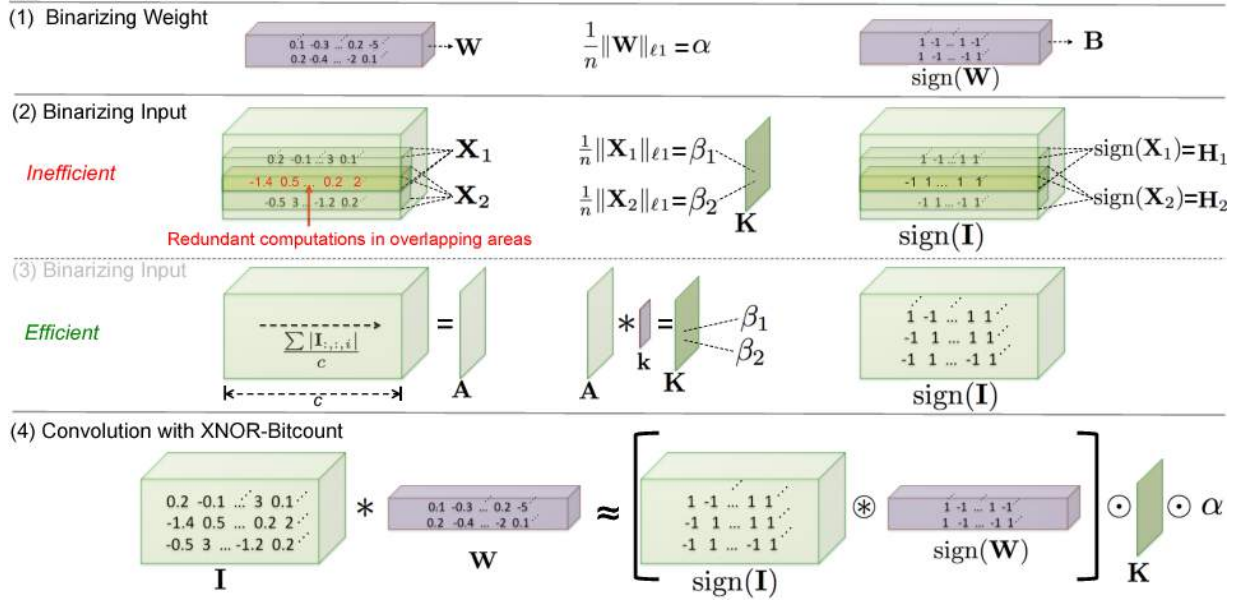


Figure 5.5: Performing a convolution of binary input and binary weights using an XNOR-bitcount operation

The figure above describes how a binarized input and binarized weight tensor can be efficiently convolved by using XNOR bitcount operations. The first method described, when performed with a binary network algorithm, results in a binary weight network, with similar accuracy, 32x less memory required, and 2x speedup of computations. However, when both the weights as well as the inputs are binarized as in the final method, 32x less memory is required, and there is a 58x speedup on average. This enables us to use a vector XNOR operation for doing the convolution of 32-bit or 64-bit vectors at a time, and then a popcount operation to find the number of set bits.

Using the method described above, it should have sped up the operations significantly on even an ARM CPU. However, while the size of the model file reduced as

expected to 32x smaller, the operations did not speed up, and the reason for this was due to how it was being stored and retrieved from memory. Once this was written in the required fashion, it still did not give acceptable performance (accuracy and output), and so we had to try something else.

CHAPTER 6

Building Perception Systems for Fully Autonomous Cars

6.1 Move-It Hackathon, March 2018

In March 2018, I was fortunate to be one of the 20 engineers selected globally for an autonomous car hackathon in China, called the Move It Hackathon (PIX-Moving (2018)), organized by PIX Moving, a startup based in Guiyang.

During this hackathon, PIX gave us a Honda Civic sedan, and a basic golf kart chassis, and gave us all the sensors, computers and equipment required to hack it to drive autonomously within a week. There were experts from Nagoya University in Japan, who built the open source software stack called Autoware (CPFL (2018)), experts from Velodyne (which makes LIDAR systems for self-driving cars), and experts from Baidu as well. We had engineers experienced in vehicle electronics, path planning and mapping, sensor fusion, software, and in optimization on hardware.

The Autoware software stack is as described below:

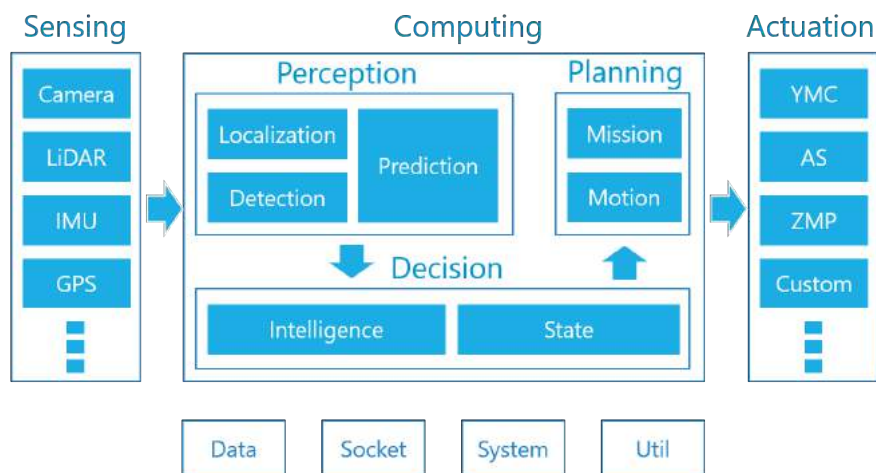


Figure 6.1: Autoware Software Stack: Sensing, Computing and Actuation

Autoware is a flexible and powerful open-source platform for autonomous vehicles. The software stack runs on the Robotic Operating System (ROS) (Quigley *et al.*

(2009)) middleware that is commonly used for robotics, which runs as a node-based inter-process communication system on top of a Linux-based operating system. Processing all the data from the sensors and running different algorithms requires a powerful computing system. In our case, we used a system with an NVIDIA GTX 1080 graphics card, an Intel Core i7 processor and 16GB of RAM. We also installed the NVIDIA drivers for CUDA, cuDNN and cuBLAS.

Using Autoware required the installation of the drivers for the LIDAR, and then the ros-velodyne driver as well. It will then send LIDAR point cloud data over Ethernet. Based on this, it was possible to use ROS to record a "ROSBAG" file, which is a recording of the timestamped data for simulation later. This can also be used to generate a map of the environment, something that is necessary for Autoware to perform all localization and mapping tasks. Finally, the vehicle can be setup to follow the waypoints generated from the map trajectory, and also to dynamically avoid any obstacles that may come in the way.

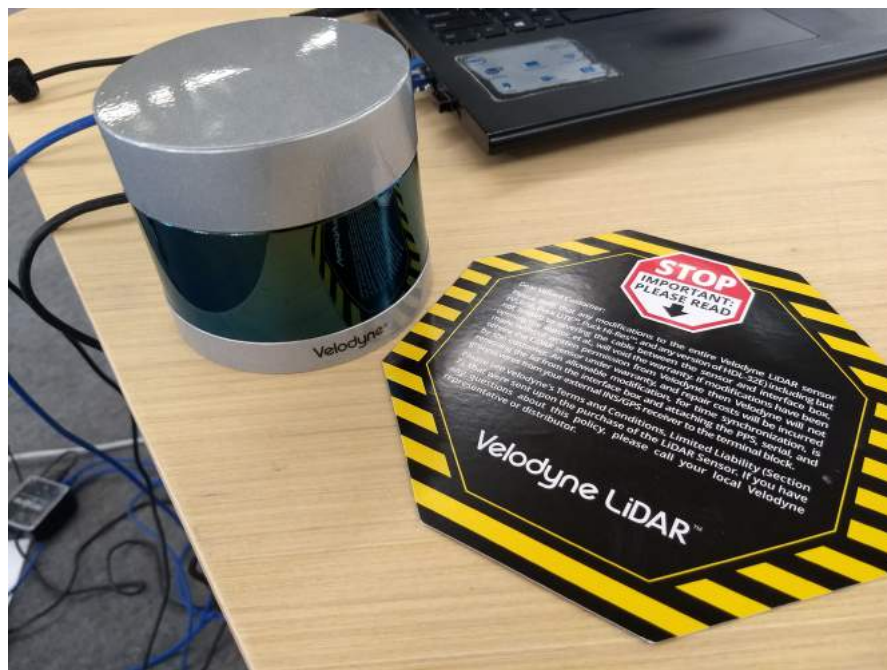


Figure 6.2: The Velodyne LIDAR Ultra Puck (VLP-32C), with 32 channels

Setting up actuation on the "chassis/cafe car" was easier, since there was direct drive-by-wire (DBW) access to the acceleration, brake and steering. However, doing the same on the Civic required a CAN "Panda" module from a company called comma.ai. It was not as easy to control as the other car, but it worked.



Figure 6.3: Data collection for the Chinese Police Gesture Dataset

Finally, we also built the world’s first open source Chinese police gesture dataset. We all spent some time participating in a data collection activity, by installing 6 cameras+LIDAR at different angles in front of a green screen, and performing the different gestures. A model was then trained using a pre-trained Inception architecture (Szegedy *et al.* (2015)) and transfer learning, to predict the gestures.

6.2 DIY Robocars KuaiKai, May 2018

In May 2018, the same company, PIX Moving, had another event, but this time it was a meetup+competition, with two categories - a small sized car and a full sized car category. Once again, I was selected to be one of the top 13 engineers worldwide participating in this event. We all teamed up for the ultimate challenge, that of a human driver vs an AI driver, to compete for the Grand Prize of \$100,000.

The advanced challenges were: GPS outage, Under Construction Sign Perception, Queuing, Automatic Parking, Slope Driving, Refueling Simulation, Accident Avoidance, and Stop Line. The basic challenges included: Checkered flag start, Pedestrian Avoidance, Bus Stop Sign Perception and S-Bend driving. There were a total of 16 challenges, for 160 points. We were provided 2 vehicles, the BAIC Motor EU4000 electric

cars. We were also given the PIX Moving Vehicle Control Unit (VCU) hardware to control them easily.

In terms of hardware and sensors, we were provided two 32-channel LIDARs and two 16-channel LIDARs, 4 Mindvision Ethernet cameras, and the PIX Moving factory and engineering team support. The VLP-32C was installed on top of the vehicle, while one camera and a VLP-16 were installed in front of it on the grille. The second camera was mounted inside the vehicle, on the windshield.

In terms of software, we went with Autoware once again. The other option we had was to use Baidu's Apollo Auto, but it supported only 64-channel LIDAR at the time, and we had more experience with Autoware, so we chose to go with it. Mapping and localization were done with LIDAR, path planning was also done once we created a map from point cloud data, and visual perception was done with custom ROS nodes for the start grid, sign detection and pedestrian detection.

Vision tasks – start grid

- Start moving only when checkerboard **not visible**
- Checkerboard detection with pure CV approach
 1. Thresholding
 2. Blob detection
 3. Output – ROI from blob grid
 4. Detection result

↓

[Don't start yet] to path planning




Figure 6.4: Start grid detection algorithm

The processed output of the grid detection is as shown below. It identifies the blobs on the image using various shape and convexity constraints and then finds the grid RoI.

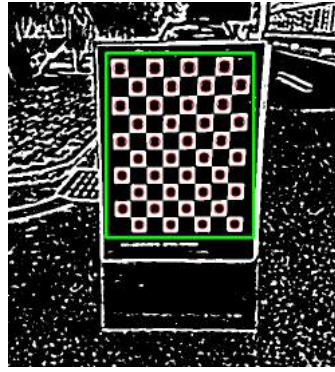


Figure 6.5: Processed grid detection output

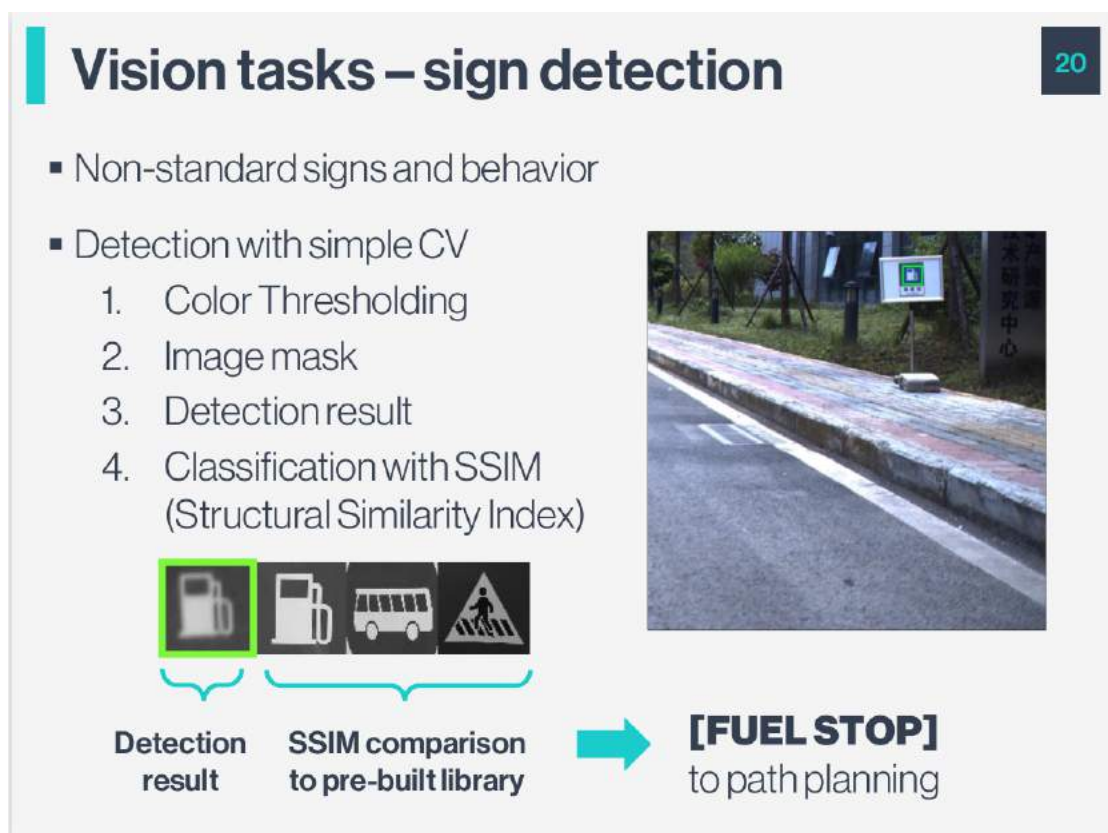


Figure 6.6: Sign detection algorithm

In the above algorithm, instead of using deep learning to train a sign detection model, we decided to use a simple computer vision approach (since all the signs were blue in colour), by thresholding the image, then finding the mask for blue, and then the region of interest for matching. Finally, we ran the match with the pre-built sign library using the Structural Similarity Index (SSIM), and this gave good results.

Vision tasks – object detection

21

- Pre-trained SSD (Single Shot Multibox Detector)
 - Trained on COCO dataset
 - Modified classification layers for three classes
 - Pedestrians, cars, background
- Real-time on GPU

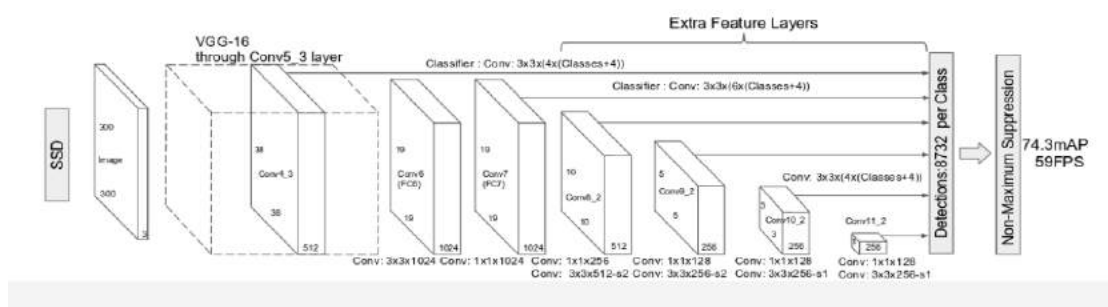


Figure 6.7: Object detection for pedestrians and cars

In the object detection algorithm, we used an off-the-shelf SSD model (Liu *et al.* (2015)) that was trained on the Microsoft Common Objects in COntext (COCO) dataset (Lin *et al.* (2014)), and then modified the output layer to predict whether an object was a pedestrian, a car, or the background. This ran in real-time on the GPU, along with all the other Autoware processes, as a separate ROS node.

We also used the different sensors we had for extra redundancy and for improving the detection accuracy. Only if different cameras gave the same results, we would consider it for path planning, as shown in the figure below. The same algorithms would run on both video streams, but the results were compared before passing it on to the path planner.

Similarly, for the car following task, we used the algorithm described below. It used the SSD object detection model, and if a car was detected within the ROI, it would track it and try to maintain the size of bounding box using speed control of the vehicle actuation. All this was performed within ROS.

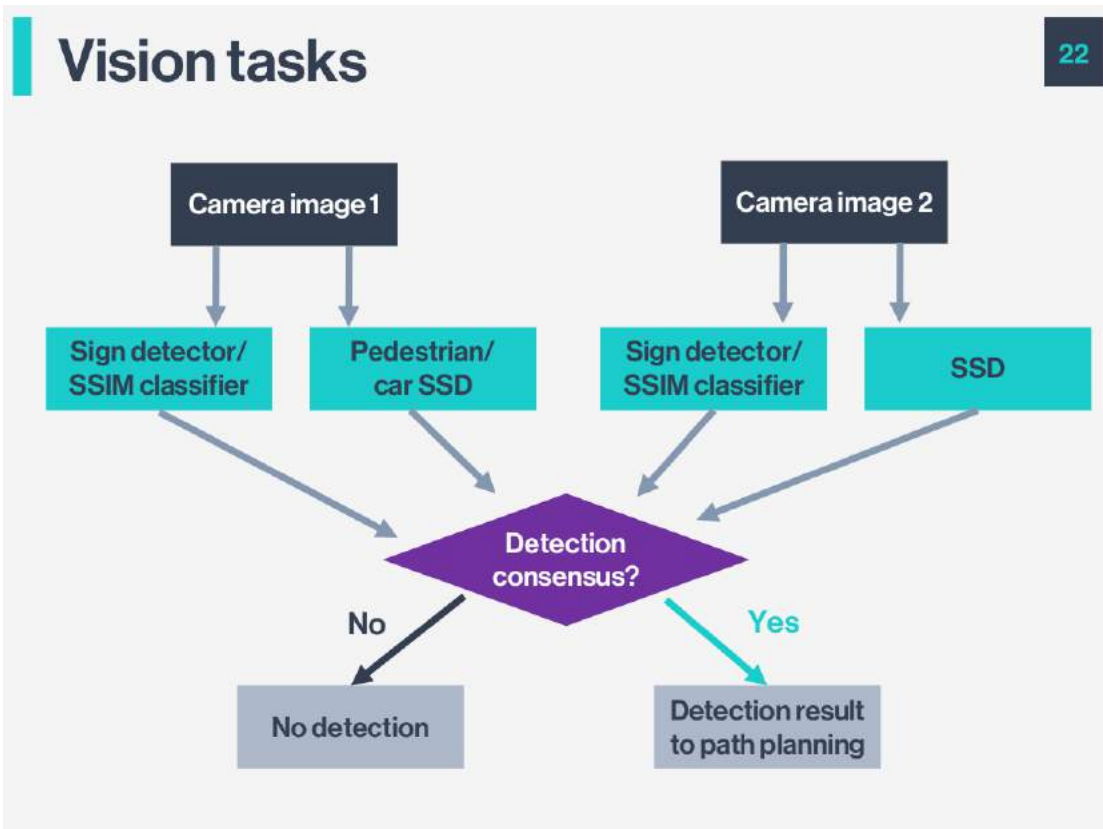


Figure 6.8: Redundancy for the different vision tasks

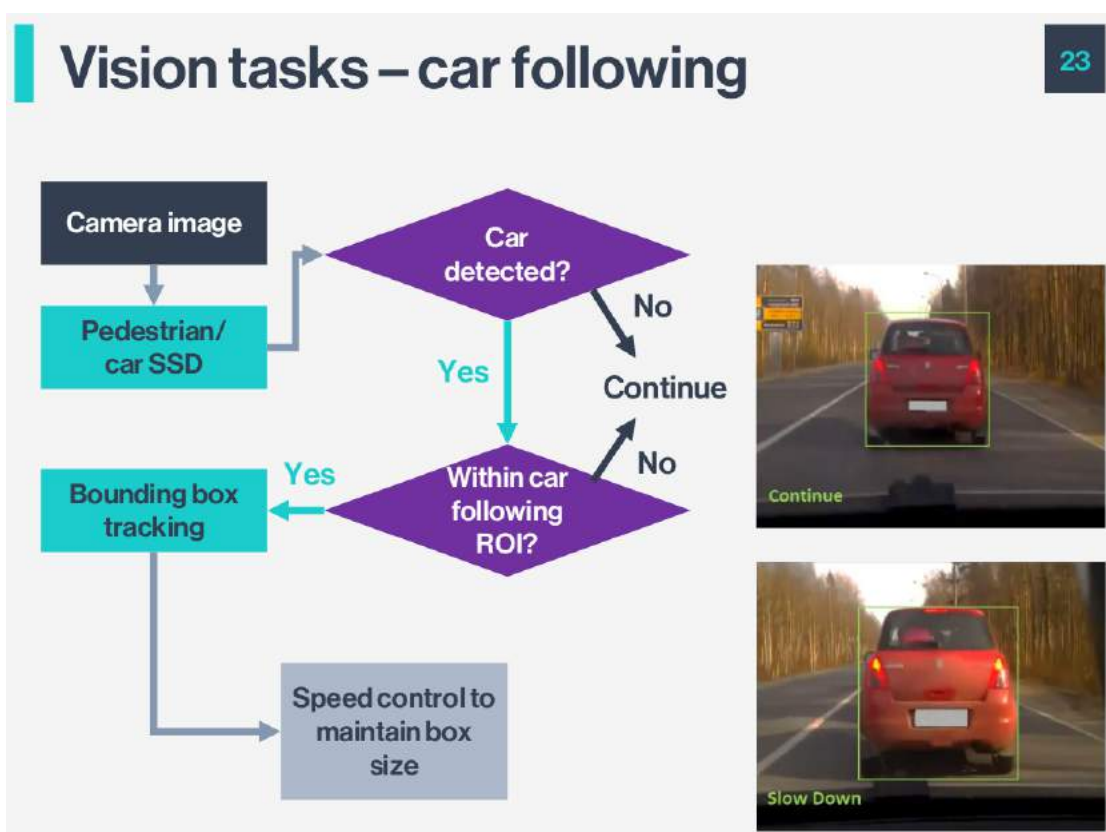


Figure 6.9: Car following algorithm using size of bounding box and tracking

Trajectory information was stored using Autoware's `waypoint_saver` and loaded using `waypoint_loader`. The lane planner was used for global planning, and A* for local planning. For control, the pure pursuit algorithm was used for getting the trajectory and speed information to navigate to the subsequent waypoints, and a twist filter was used to smooth the velocity. Finally, the `vehicle_gateway` was used to send the actuation information to the PIX VCU.

The final result was that our team scored 105/160 on the race day. The best human driver scored 150/160. After the race, we continued to optimize the ROS nodes and the vehicle performance, and we were able to reach a score of 120/160. We placed first, but didn't beat the human driver, so won an award of \$40,000.



Figure 6.10: Camera+LIDAR on the front



Figure 6.11: The insider view



Figure 6.12: Navigating the S-Bend

We also presented our work titled "The PIX Moving Kuaikai: Building a Self-Driving Car in 7 Days" at the DSP in Vehicles conference at Nagoya University, Japan, in October 2018.

CHAPTER 7

Image Enhancement methods for Camera Systems

In the CS6350: Computer Vision course I took in the Jan-May 2018 semester, my term project assignment was on Panorama Generation from Videos with Significant Blur. We used a single-image architecture for deblurring images, which was trained using a Generative Adversarial Network (GAN).

The paper on GANs by Goodfellow *et al.* (2014) introduces an adversarial framework with two models - a generator G and a discriminator D , where the generator attempts to generate samples that mimic the training data, while the discriminator attempts to identify if the observed sample is real or fake. This corresponds to a minimax two player game. When D and G are multi-layer perceptron networks, they are end-to-end trainable using backpropagation and the minimax objective function.

The above framework was further improved by Isola *et al.* (2017) for Image to Image Translation using Conditional Adversarial Networks (also known as pix2pix), where the objective function was modified to condition the discriminator on the input image. There is also an additional L1 loss component for the generator to closely resemble the ground truth output. Although pix2pix gives excellent results for a variety of image translation tasks (colorization, in-painting, edges to handbags/shoes, segmentation to photos, etc), it doesn't work well for deblurring.

The DeblurGAN paper by Kupyn *et al.* (2017) improved on pix2pix using a new Wasserstein GAN objective function that is more stable and also includes a perceptual content loss. Recent papers also used residual blocks for deeper networks and larger receptive fields. Finally, there were papers attempting to use multiple images in sequence to deblur the final image using spatio-temporal architectures with 3D convolutions and deep residual learning methods. We were able to identify a unique method of performing multi-image convolution on a video stream and a method to implement the same in hardware, and filed a patent on it. However, it was too slow for use both on an ARM CPU as well as on an NVIDIA GPU, and hence we could not use it further.

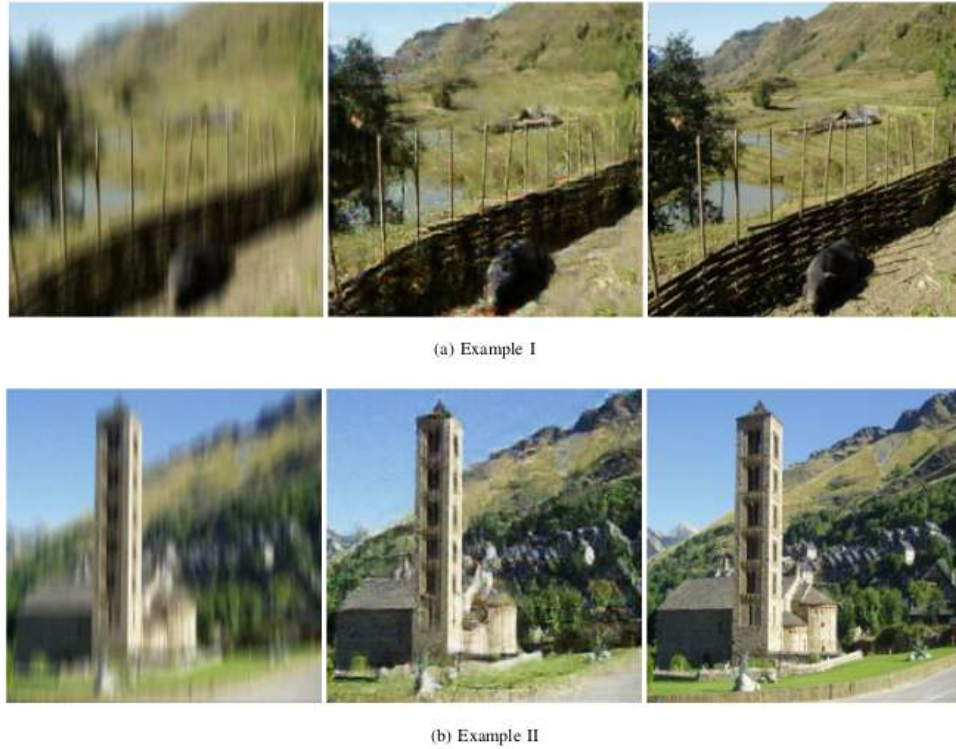


Figure 7.1: Sample Deblurred Images. Arranged from left to right as blurred, deblurred and ground truth.

We also identified a CVPR 2018 publication titled "Learning to See in the Dark" (Chen *et al.* (2018)). It trained a neural network to translate raw images from cameras with low ISO to highly enhanced and clear images. To do this, it trained the network by using images taken of the same scene using a camera with much higher ISO, like a DSLR. The network then learned how to translate between the low ISO image inputs to the high ISO image outputs. The disadvantage was that it required a significant amount of processing power and RAM to train this network, as well as for inferencing of the results (an NVIDIA Titan X graphics card, and 64-128 GB of RAM). It also required the raw image output of the camera, something that was not easily available from USB webcams or from MIPI.

Finally, the solution we used for image enhancement on our device was very simple and computationally inexpensive. We used an adaptive histogram equalization on all the three colour channels, and a gamma correction for dark scenes (like at night). This significantly improved the prediction accuracy as well as confidence scores for our object detection algorithms!

CHAPTER 8

Generating and training on Indian datasets

In August 2018, Intel and the International Institute of Information Technology Hyderabad (IIIT-H) released a unique Indian road dataset. It consists of 10,000 images, finely annotated with 34 classes collected from 182 drive sequences on Indian roads.

The dataset consists of images obtained from a front facing camera attached to a car. The car is driven around the Hyderabad, Bangalore cities and their outskirts. The images are mostly of 1080p resolution, but there is also some images with 720p and other resolutions. The figure bellow gives an illustration of the data acquisition setup:

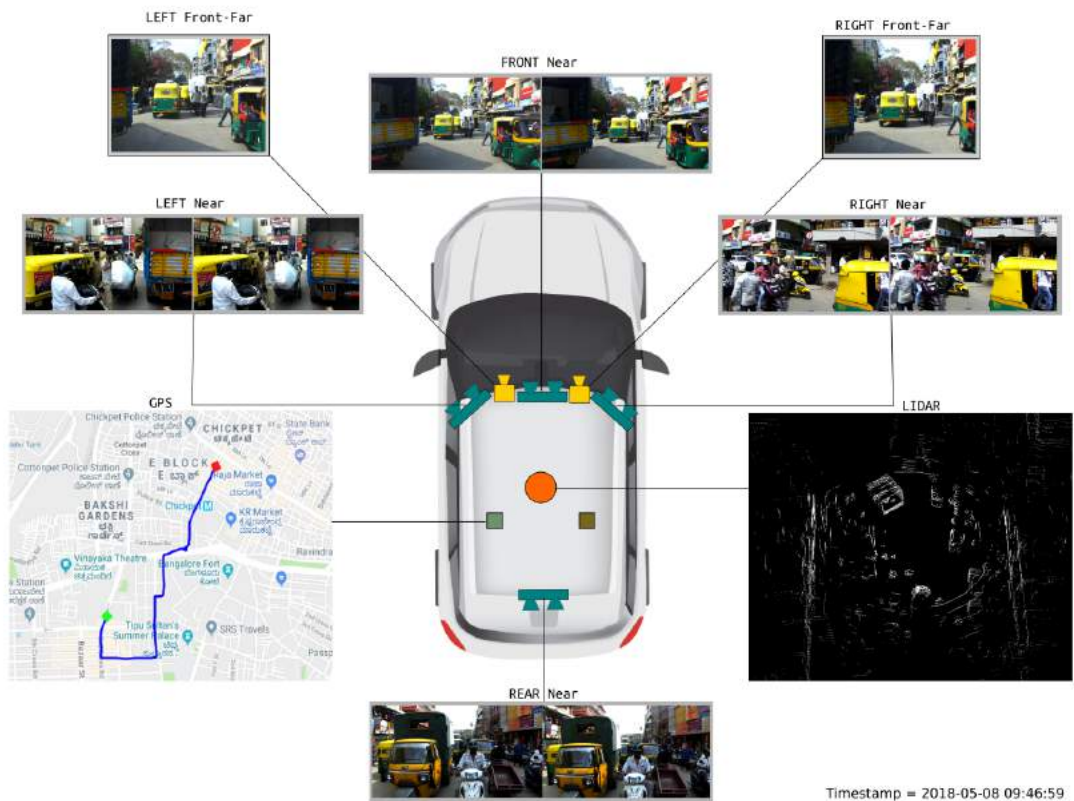


Figure 8.1: AutoNUE Data Acquisition Setup

Intel and IIIT-H also held a competition as part of the European Conference on Computer Vision (ECCV) 2018, titled "Scene Understanding for Autonomous Navigation in Unstructured Environments" or AutoNUE. They wanted teams to perform semantic and instance segmentation of different classes of Indian vehicles as provided in their dataset.

Although there were already datasets existing in a similar format from countries like USA and Europe, like KITTI (Geiger *et al.* (2013)), CityScapes (Cordts *et al.* (2016)) and BDD100K (Yu *et al.* (2018)), none of them had the unique vehicles and two-wheelers that we see only in India. As a result, a model that was trained on them would perform poorly on the AutoNUE dataset. It required retraining on the new object classes to get a decent output as shown below (semantic segmentation):

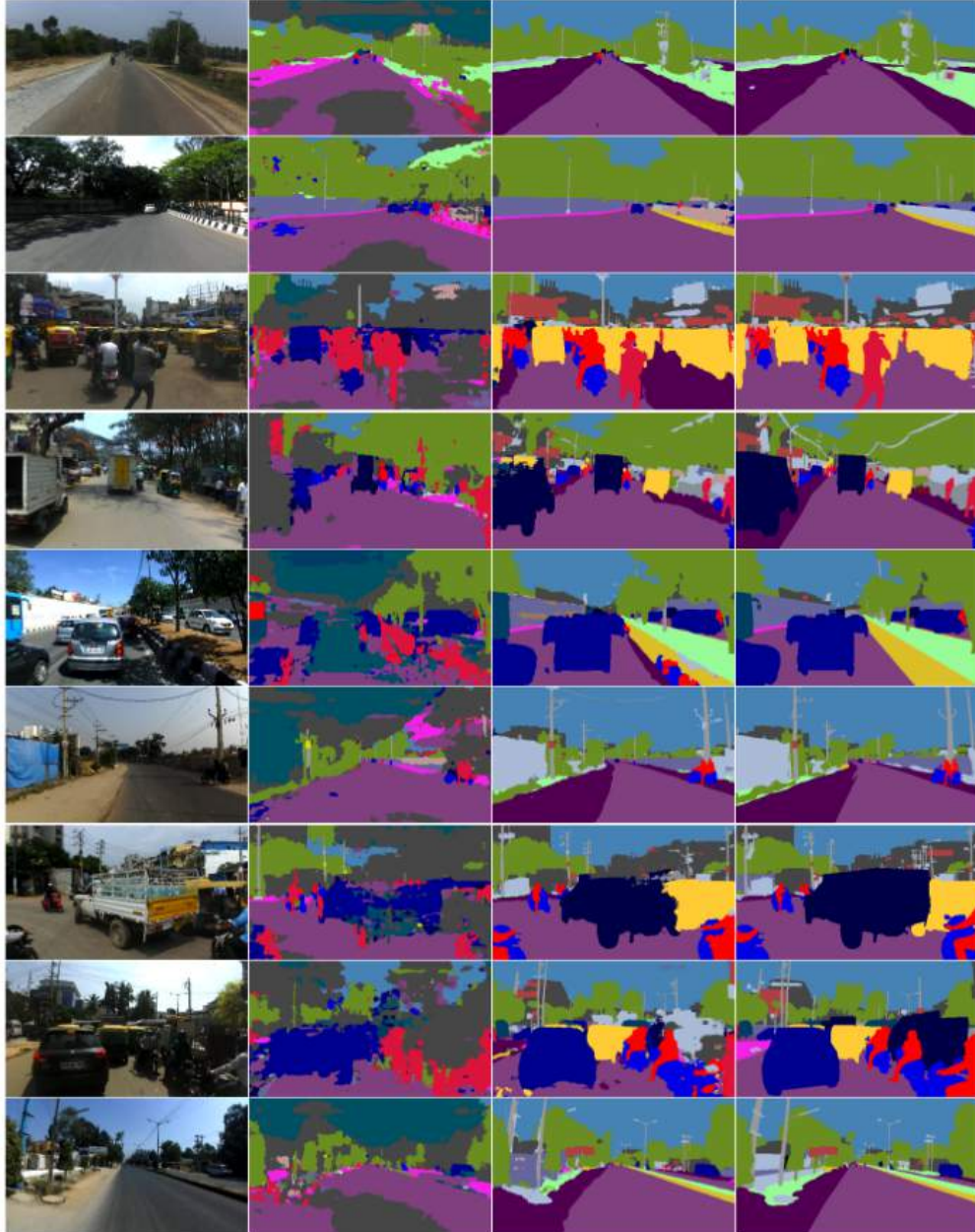


Figure 8.2: L to R: example image, prediction with pretrained model, prediction after training on this dataset, ground truth

And so for the challenge, I decided to use the Mask RCNN (He *et al.* (2017)) architecture that was recently published for semantic and instance segmentation. I used the

Matterport open source implementation (Abdulla (2017)), which uses a Feature Pyramid Network (FPN) and ResNet-101 backbone. They also provided a blog post with details of how to use transfer learning to train on a custom dataset. The results after training for over 100 hours, with multiple hyperparameters fine tuned, was as follows:

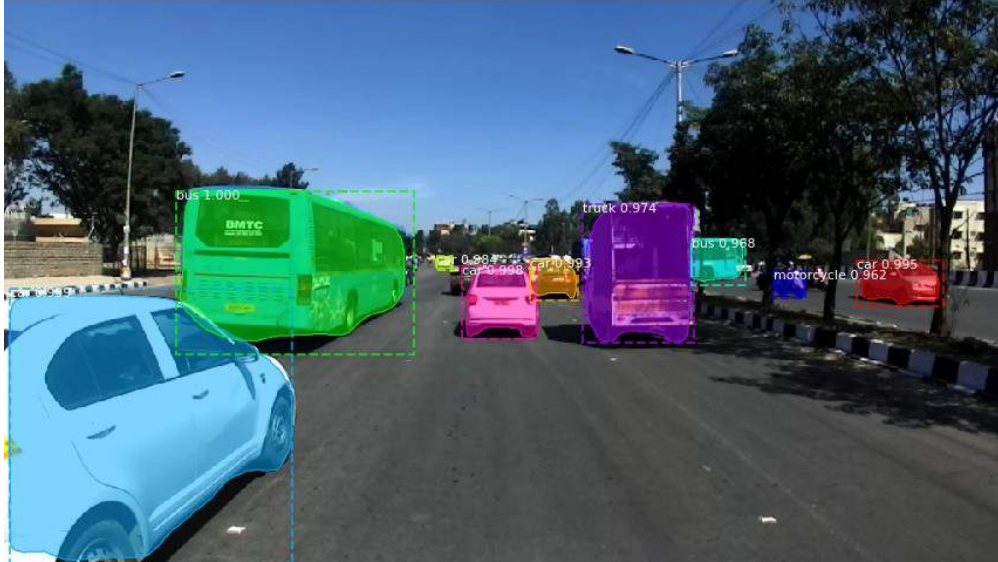


Figure 8.3: Output Image 1

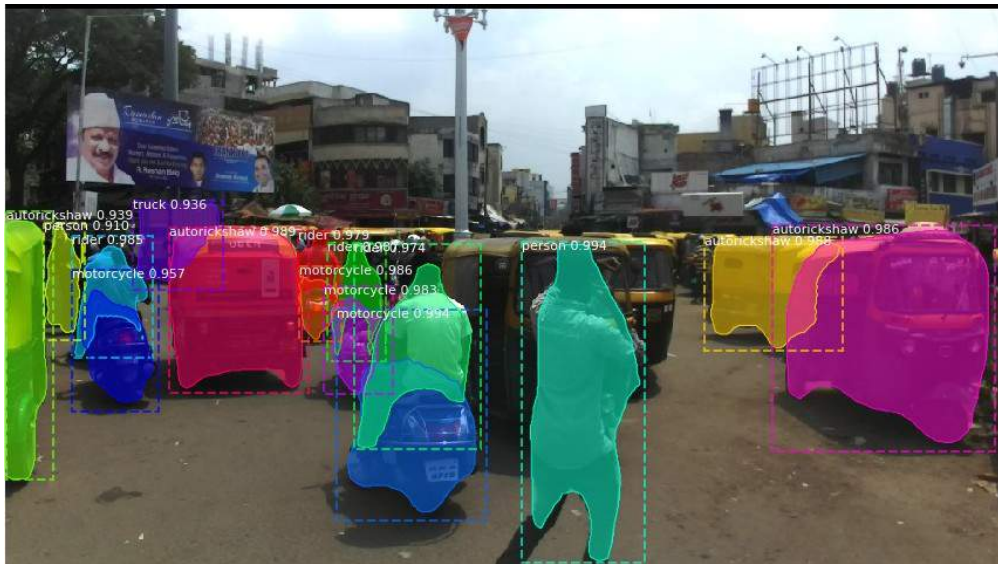


Figure 8.4: Output Image 2

As you can see, the model is able to identify vehicles like rickshaws, people, two-wheelers, cars, buses and trucks, and also segment each instance individually. We were declared as the best performing Indian team in this challenge, and awarded a travel grant to ECCV 2018 by Intel and IIIT-H.

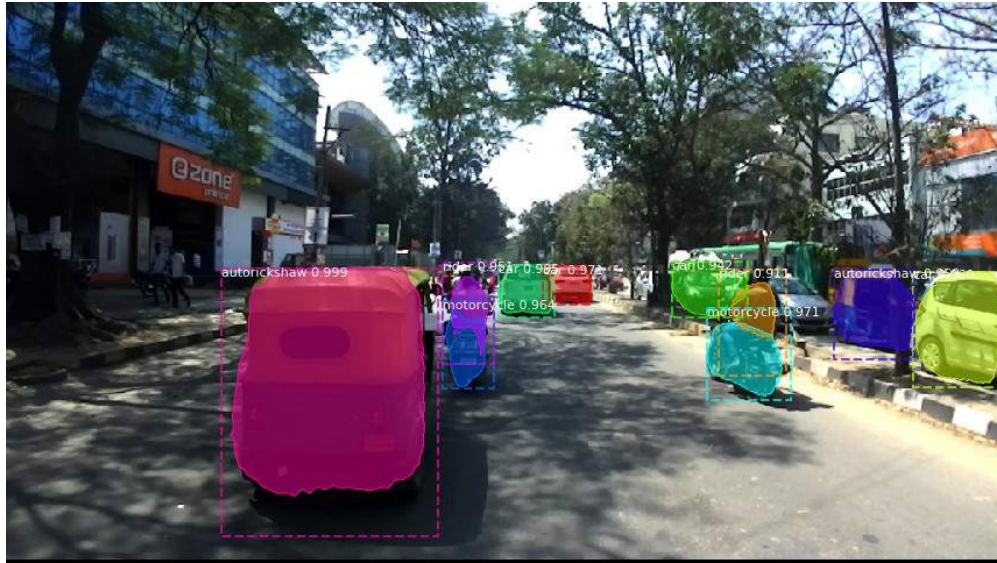


Figure 8.5: Output Image 3

We also trained a pix2pix (Image Translation using Conditional Adversarial Networks by Isola *et al.* (2017)) model for transforming US/European image data into Indian road data. After 300 hours of training, nearly 75 epochs, it did reasonably well, as shown below. This provided a quick method of getting the required data without actually putting devices on the road.



Figure 8.6: Example 1: Input US/Europe road -> AI Generated Indian Road

Here you can see that the smooth and perfect roads abroad are converted to pothole filled and confusing roads for India. This clearly highlights the "unstructured" nature of India's roads.



Figure 8.7: Example 2: Input US/Europe road -> AI Generated Indian Road

Here it makes the roads more dusty, and removes some of the foliage in the distance. The reason for this is because most of the Indian data we used for transfer learning was taken on highways, and that contributes to a clear sky ahead of the ego vehicle.



Figure 8.8: Example 3: Input US/Europe road -> AI Generated Indian Road

Once again, the houses are cleared to be replaced by clear skies. The roadside also becomes dustier, as it would be in India.



Figure 8.9: Example 4: Input US/Europe road -> AI Generated Indian Road

Finally, in this image, the generative adversarial network converts a car on the side of a road in US/Europe into an autorickshaw! This is definitely very useful, it has learned that rickshaws are vehicles in India and it should try to replicate them from the provided data.

CHAPTER 9

The final product: An Alerting and Monitoring System to Improve Driver Productivity

In September 2018, we participated in the MOVEHack Global Mobility Hackathon conducted by Niti Aayog, Govt. of India. This was in association with Intel, who provided the AutoNUE dataset and the challenge for all the teams. We reached the finals in Delhi, and ended up placing 2nd worldwide! We spoke to many people there, and refined our product as follows.

HOW IT WORKS

Driver-Facing Camera
Authenticates the driver and checks for **drowsiness, distraction, and drunkenness**. Equipped with IR LEDs for **night vision**, it also loop records up to 24 hours in HD.

GPS Tracking
Track the vehicle whereabouts, get real-time and reliable alerts about deviations and delays through **geofencing**.

4G + WiFi + Bluetooth
Our devices are always connected to you. **Access all the critical data from anywhere 24*7.**



OBD-II Compatible
Understand everything that goes on inside your vehicle to improve its performance. **Get speeding alerts, harsh incidents, and fuel and maintenance monitoring.**

Road-Facing Camera
Monitors the road in the day and the night. **AI-based collision avoidance, tailgating alerts, rule violation alerts etc** will be added soon. It also loop records up to 24 hours in HD.

Figure 9.1: The first prototype of the device

This is a plug-and-play device that mounts onto the windshield of a vehicle. It has two cameras facing in opposite directions - to face the driver and the road. It also comes with an On-Board Diagnostics (OBD) module, which connects to the in-vehicle sensors and provides information about speed/acceleration/braking/fuel levels and more. The device has GPS for location tracking, 4G to upload data to the cloud, and the cameras

use AI and computer vision to identify potentially dangerous incidents and flag them for review by uploading geotagged video and OBD snippets to the cloud. We filed 4 patents as part of this project, as described below.

9.1 Overall System Design Patent

This patent was titled "A modular multi-sensor hybrid architecture for driving environment monitoring and analysis". We described a broad and flexible design that could be extended to incorporate additional sensors in the future. A processing system would take in information from multiple sensors, which may be pre-processed in some way beforehand, and then it would generate alerts and insights depending on the use case. The design is shown below.

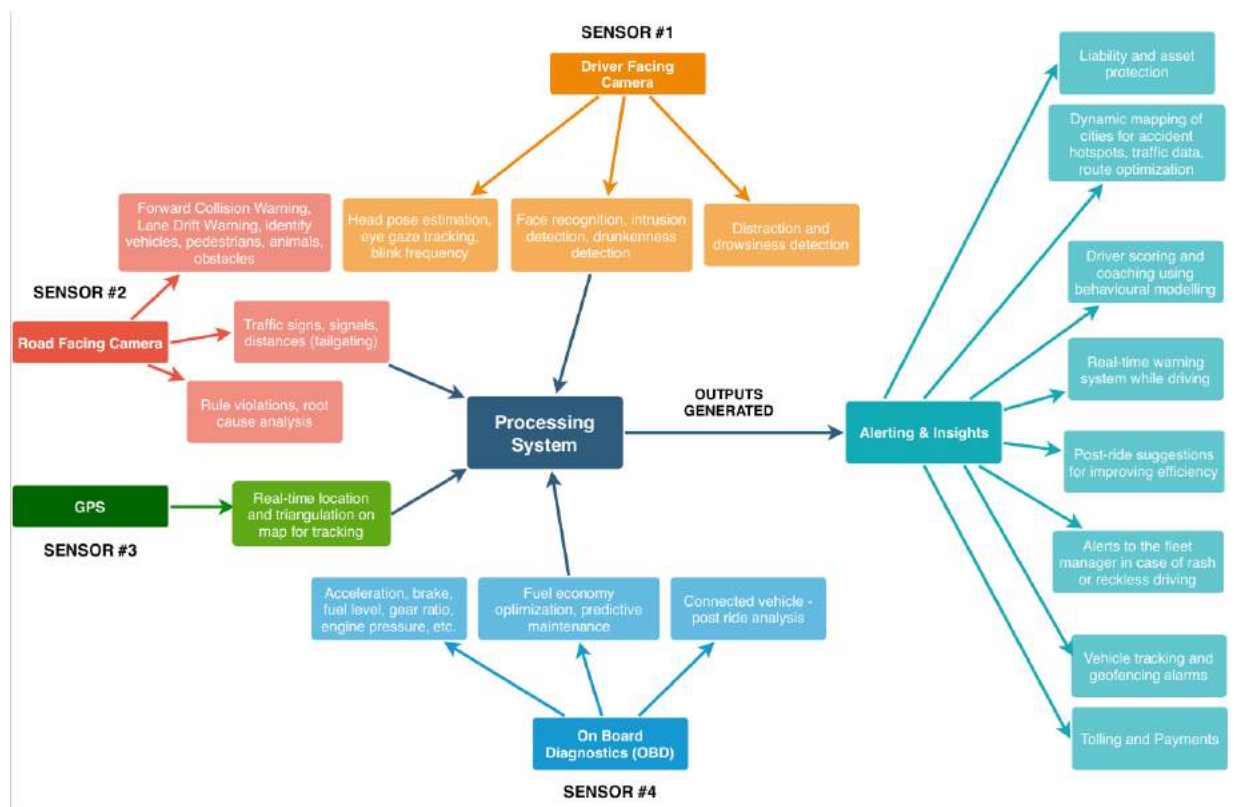


Figure 9.2: Overall System Design

9.2 Road Facing Camera Patent

This patent was titled "Systems and methods for road-facing camera based driver assistance and autonomous driving systems". We described a hybrid vision based telematics platform which captures information about the road and outside-vehicle environment, and in-vehicle sensors via on-board diagnostics, and processes it on the device in real-time, and also sends insights to the cloud for post-drive analytics. The design is shown below.

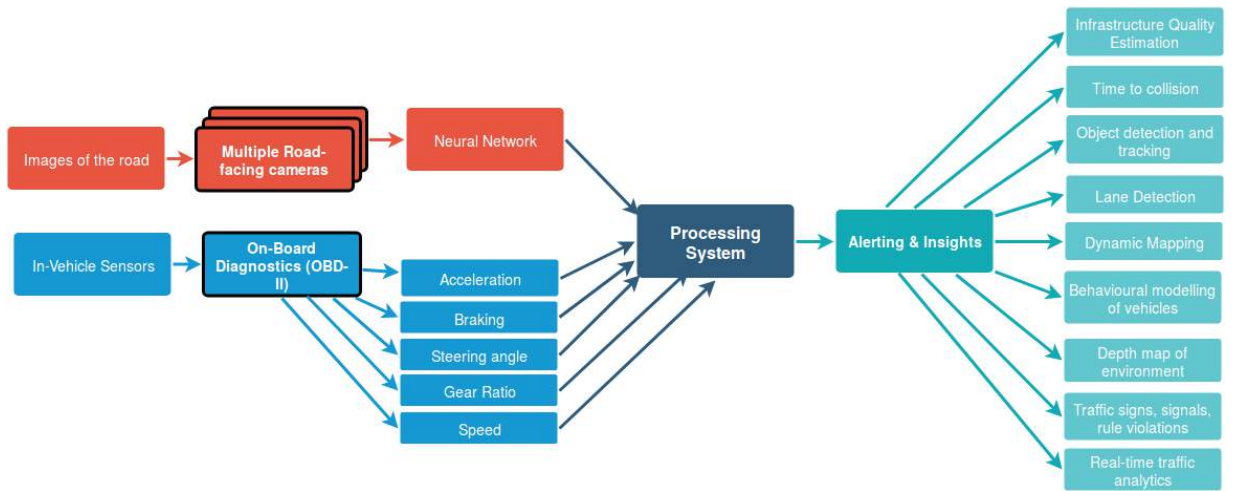


Figure 9.3: Road-Facing Camera ADAS Design

9.3 Spatiotemporal Deblurring Patent

This patent was titled "A Spatio-temporal Architecture for Real-time Blind Motion Deblurring on an FPGA". We described an implementation of a specific machine learning architecture in hardware by using specialized optimization techniques for deblurring of arbitrarily blurred video sequences. Unfortunately we were not able to get the published implementation to work on our PCs, and so could not port it to the FPGA either.

9.4 Driver Facing Camera Patent

This patent was titled "Multi sensor driver behavior monitoring system". We described an implementation of neural network architecture in hardware by using multiple sensor

data and specialized deep learning techniques for real-time driver behavior detection. The design is shown below.

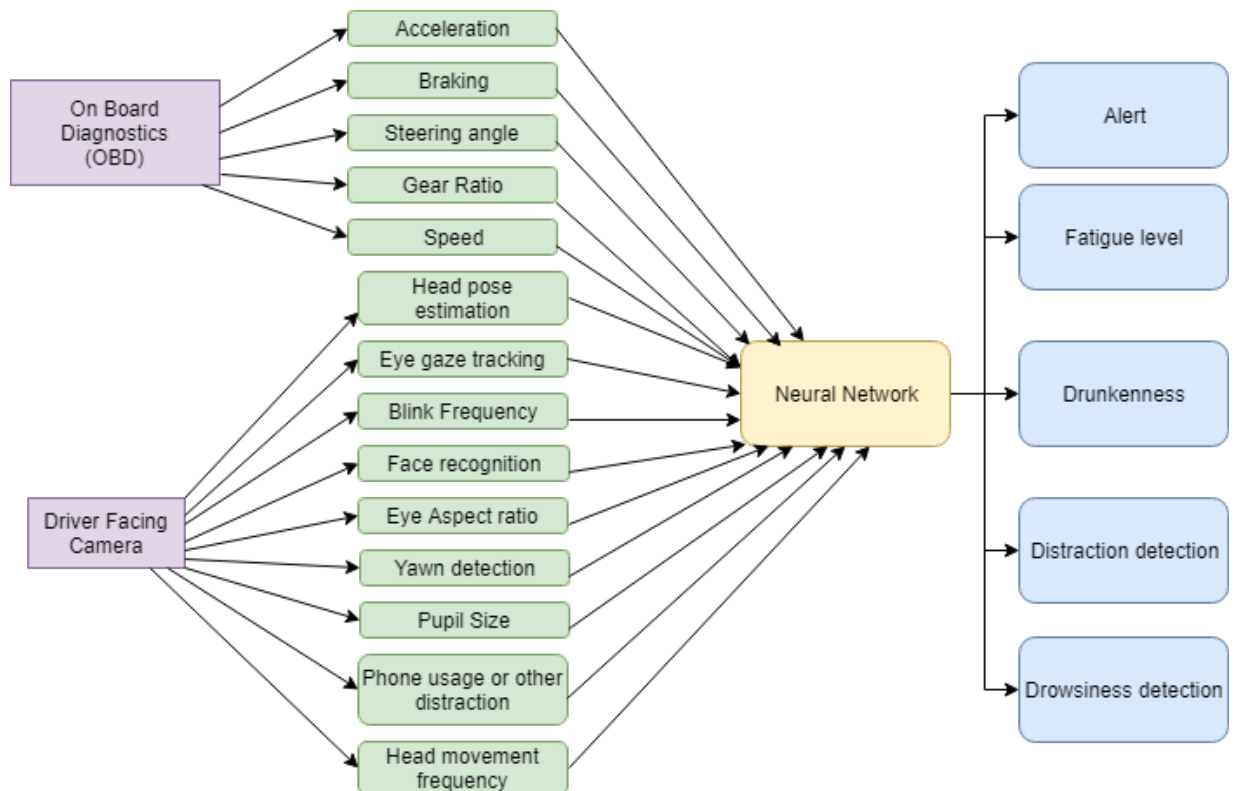


Figure 9.4: Driver Behaviour Monitoring System Design

9.5 Web Architecture

We asked one of the members from our team to build us a web-based dashboard. I implemented a storage system using the Amazon Web Services (AWS) Simple Storage System (S3) that would provide secure and encrypted storage for each driver, which could easily be accessed across a fleet. The dashboard image looks something like below:

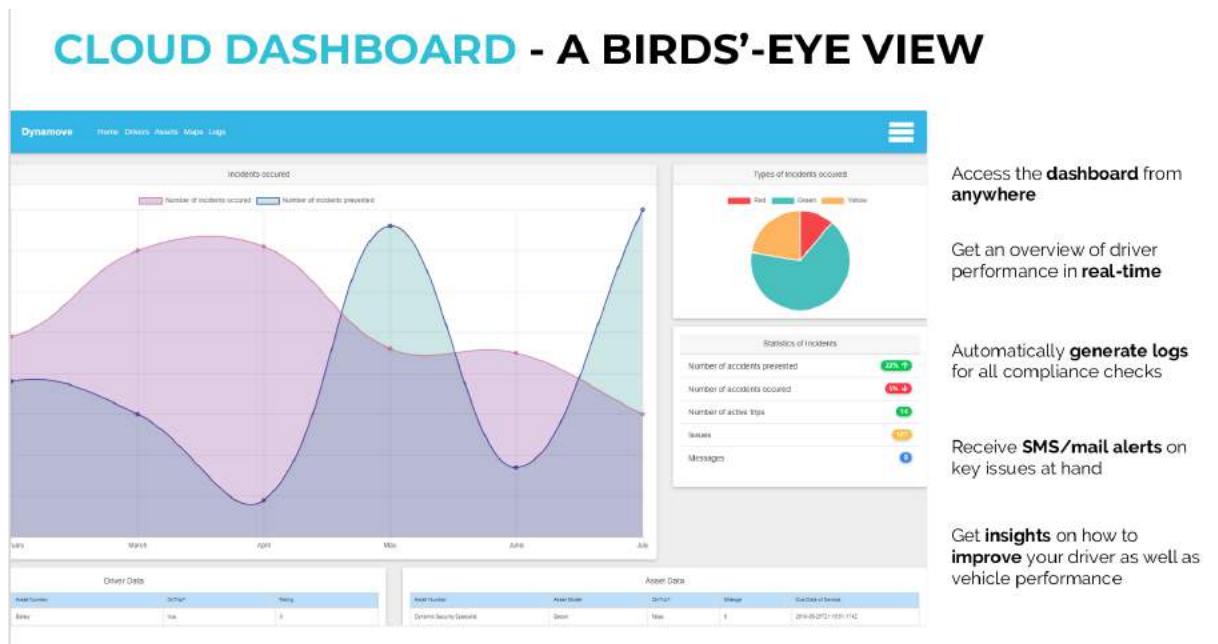


Figure 9.5: A Web-Based Fleet Dashboard

And so, having built this device, we have started deploying it into trucks from different companies, including BD Dhalla Transport in Mumbai, and Instavans in Bangalore. We are still ironing out minor issues, but the overall concept has been proven and requires time and effort to pursue the opportunity and scale.

REFERENCES

1. **Abdulla, W.** (2017). Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask_RCNN.
2. **AD9361** (2019). Ad9361. URL <https://www.analog.com/en/products/ad9361.html>.
3. **Adams, R.** and **L. Bischof** (1994). Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(6), 641–647. ISSN 0162-8828.
4. **Chen, C., Q. Chen, J. Xu,** and **V. Koltun** (2018). Learning to see in the dark. *CoRR*, **abs/1805.01934**. URL <http://arxiv.org/abs/1805.01934>.
5. **Cordts, M., M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth,** and **B. Schiele**, The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
6. **CPFL** (2018). Autoware. URL <https://www.autoware.org/>.
7. **DSP** (2018). Dsp. URL <https://www.dsp-workshop.org/>.
8. **Dynamove** (2019). Dynamove. URL <https://www.dynamove.in>.
9. **Geiger, A., P. Lenz, C. Stiller,** and **R. Urtasun** (2013). Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*.
10. **Girshick, R.**, Fast r-cnn. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15. IEEE Computer Society, Washington, DC, USA, 2015. ISBN 978-1-4673-8391-2. URL <http://dx.doi.org/10.1109/ICCV.2015.169>.
11. **Girshick, R., J. Donahue, T. Darrell,** and **J. Malik**, Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition*. 2014.
12. **Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville,** and **Y. Bengio**, Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14*. MIT Press, Cambridge, MA, USA, 2014. URL <http://dl.acm.org/citation.cfm?id=2969033.2969125>.
13. **Gridlock** (2019). Flipkart gridlock hackathon. URL <https://stories.flipkart.com/gridlock-hackathon-innovation/>.
14. **HackRF** (2019). Hackrf. URL <https://greatscottgadgets.com/hackrf/>.
15. **He, K., G. Gkioxari, P. DollÁar,** and **R. Girshick**, Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017. ISSN 2380-7504.

16. **He, K., X. Zhang, S. Ren, and J. Sun**, Deep residual learning for image recognition. *In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. ISSN 1063-6919.
17. **Isola, P., J.-Y. Zhu, T. Zhou, and A. A. Efros** (2017). Image-to-image translation with conditional adversarial networks. *CVPR*.
18. **Jetson-TX2** (2019). Jetson-tx2. URL <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/>.
19. **Kazemi, V. and J. Sullivan**, One millisecond face alignment with an ensemble of regression trees. *In 2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014. ISSN 1063-6919.
20. **King, D. E.** (2009). Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, **10**, 1755–1758.
21. **Krizhevsky, A., I. Sutskever, and G. E. Hinton**, Imagenet classification with deep convolutional neural networks. *In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12*. Curran Associates Inc., USA, 2012. URL <http://dl.acm.org/citation.cfm?id=2999134.2999257>.
22. **Kupyn, O., V. Budzan, M. Mykhailych, D. Mishkin, and J. Matas** (2017). Deblurgan: Blind motion deblurring using conditional adversarial networks. *ArXiv e-prints*.
23. **Lin, T.-Y., M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick**, Microsoft coco: Common objects in context. *In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars* (eds.), *Computer Vision – ECCV 2014*. Springer International Publishing, Cham, 2014. ISBN 978-3-319-10602-1.
24. **Liu, W., D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg** (2015). SSD: Single shot multibox detector. *arXiv preprint arXiv:1512.02325*.
25. **Lloyd, S.** (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, **28**(2), 129–137. ISSN 0018-9448.
26. **Mallick, S.** (2019). Head pose estimation using opencv and dlib. URL <https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/>.
27. **MOVEHack** (2018). Movehack. URL <http://pib.nic.in/newsite/PrintRelease.aspx?relid=181379>.
28. **NVIDIA** (2018). Nvidia ai. URL <https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/>.
29. **NVIDIA** (2019). Nvidia cuda. URL <https://www.nvidia.in/object/cuda-parallel-computing-in.html>.
30. **Osmocom** (2018). Osmocom. URL <https://osmocom.org/projects/rtl-sdr/wiki/Rtl-sdr>.
31. **Pioneer** (2018). Pioneer fellowship. URL <https://www.pioneer.app/>.

32. **PIX-Moving** (2018). Move it hackathon. URL <https://www.pixmoving.com/move-it>.
33. **PYNQ-Z1** (2019). Pynq-z1. URL <https://store.digilentinc.com/pynq-z1-python-productivity-for-zynq-7000-arm-fpga-soc/>.
34. **Quigley, M., K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng**, Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*. 2009.
35. **Rajasthan** (2017). Rajasthan hackathon 2.0. URL <https://www.hackerearth.com/challenges/hackathon/rajasthan-hackathon-20/>.
36. **Rastegari, M., V. Ordonez, J. Redmon, and A. Farhadi**, Xnor-net: Imagenet classification using binary convolutional neural networks. In **B. Leibe, J. Matas, N. Sebe, and M. Welling** (eds.), *Computer Vision – ECCV 2016*. Springer International Publishing, Cham, 2016. ISBN 978-3-319-46493-0.
37. **Redmon, J., S. Divvala, R. Girshick, and A. Farhadi**, You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. ISSN 1063-6919.
38. **Ren, S., K. He, R. Girshick, and J. Sun**, Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, NIPS’15*. MIT Press, Cambridge, MA, USA, 2015. URL <http://dl.acm.org/citation.cfm?id=2969239.2969250>.
39. **Rosebrock, A.** (2019). Drowsiness detection with opencv. URL <https://www.pyimagesearch.com/2017/05/08/drowsiness-detection-opencv/>.
40. **RTL-SDR** (2019). Rtl-sdr. URL <https://www.rtl-sdr.com>.
41. **Shi, J. and J. Malik** (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(8), 888–905. ISSN 0162-8828.
42. **Sparkle, K.** (2018). URL <https://www.sparkle.kpit.com>.
43. **Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich**, Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*. 2015. URL <http://arxiv.org/abs/1409.4842>.
44. **Umuroglu, Y., N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers**, Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA ’17*. ACM, 2017.
45. **Viola, P. and M. Jones**, Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1. 2001. ISSN 1063-6919.
46. **Yu, F., W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell** (2018). BDD100K: A diverse driving video database with scalable annotation tooling. *CoRR*, **abs/1805.04687**. URL <http://arxiv.org/abs/1805.04687>.

LIST OF PAPERS BASED ON THESIS

1. David Robert Wong, Alexander Carballo, Rohan Rao, Oscar Rovira, Chuan Yu
The PIX Moving Kuaikai: Building a Self-Driving Car in 7 Days *8th Biennial
DSP in Vehicles Conference*, (2018).
2. Rohan Rao Driver Safety in the Indian Context *8th Biennial DSP in Vehicles
Conference*, (2018).