A PROJECT REPORT
ON

”SMART TRAFFIC CONTROL SYSTEM”

Submitted to
NIRMA UNIVERSITY

In Partial Fulfilment of the Requirement for the Award of

IDEA LAB FISCAL YEAR 2019-2020

BY

ANANT RAINA	17BEC010
ABHISHEK RAKHOLIYA	17BEC078
MOSAM PATEL	17BEC067
SHIVANSHU UPADHYAY	17BEC121

UNDER THE GUIDANCE OF
DR. TANISH ZAVERI

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
INSTITUTE OF TECHNOLOGY, NIRMA UNIVERSITY
AHMEDABAD, GUJARAT - 384281

2019-2020

Acknowledgements

We are profoundly grateful to **Dr. Tanish Zaveri** for his expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion.

We would like to express deepest appreciation towards **Dr. RN Patel**, Director, ITNU, **Dr. Dhaval Pujara**, Head of Department of Electronics and Communication Engineering and **Dr. Akash Mechwan**, Project Coordinator, Idea Labs whose invaluable guidance supported us in completing this project.

At last we must express our sincere heartfelt gratitude to all the staff members of Electronics and Communication Engineering Department who helped me directly or indirectly during this course of work.

MOSAM PATEL
ANANT RAINA
ABHISHEK RAKHOLIYA
SHIVANSHU UPADHYAY

ABSTRACT

Traffic in India is one of the premier causes of tardiness for the working class society today. With increasing population this problem is only expected to increase. Systems, especially electronic in nature are quick to get obsolete when certain parameters overflow and this is a perfect case for such a scenario. Not only is the current traffic management system obsolete, it lacks any feedback loop: a parameter that controls the output in such a way that both the computation and the latency is optimized.

We propose to solve this very issue with a SMART TRAFFIC CONTROL SYSTEM, one that assigns the density of cars on a particular junction to allocate time dynamically for the lights to switch. This proposal seemingly solves all the inherent issues that come with the current system and hopes to add feedback loops to it with minimum possible latency. We also realize other issues such as blocked ambulances and implement priority queues and interrupts to solve this issue. As of the validation of this report, we used the NVIDIA Jetson Nano for computation and the YOLO Object Detection Algorithm ported to support TensorFlow Lite for ease of execution on the Nano. More details about the hardware and software has been documented in detail in this report. **Keywords:** YOLO, Control Systems, Algorithms, Camera, OpenCV, Jetson Nano, GSM

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

As stated before, the current system has started to become inconvenient in incapable of handling the sheer influx of population and vehicles entering the ecosystem. A new system needs to replace the current time-based triggering in traffic control system. Traffic in many countries becomes unmanageable due to congestion and high density. We wish to propose a new system that leverages this very fact and create map out the density to act as the feedback loop, instead of the conventional time-based loops.

Another major issue with the current system materializes itself during emergencies such as ambulances passing or law enforcement agencies wanting a clear path. The current system doesn't take into account its surroundings and how they may affect the timing algorithms and hence we can often see poor management of accidents or passing ambulances. The system we propose will make an attempt to offer solutions for this very purpose.

1.2 Proposed Hardware and Required Budget

This project was thought-up with a certain type of hardware and software affects in mind. The budget for its inception was accordingly provided by the institute for which we are grateful. Here is an approximate mapping of how the budget was used.

Hardware/Software Package	**Particulars**	**Costing**
Camera Module	iBALL CHD20.0	1499 INR
Embedded Processor	NVIDIA Jetson Nano	9999 INR
GSM Module	SIM800A	1000 INR
Timer Module	TM1637	150 INR

Table 1.1: Costing Dynamic for Smart Traffic Control System

1.3 Hardware in-Depth

This section delves deeper into the hardware used for the purpose of this project and also the requirements for their efficient working. The purpose of this section is to open dialogue over the use of alternatives to our propositions and also to make more efficient models as the project progresses into the hands of more capable and well-versed groups.

1.3.1 NVIDIA Jetson Nano

NVIDIA Jetson Nano is an embedded system-on-module (SoM) and developer kit from the NVIDIA Jetson family, including an integrated 128-core Maxwell GPU, quad-core ARM A57 64-bit CPU, 4GB LPDDR4 memory, along with support for MIPI CSI-2 and PCIe Gen2 high-speed I/O. Jetson Nano runs Linux and provides 472 GFLOPS of FP16 compute performance. It is a powerful computer that lets you run multiple neural networks in parallel for applications like image classification, object detection, segmentation, and speech processing. All in an easy-to-use platform that runs in as little as 5 watts.

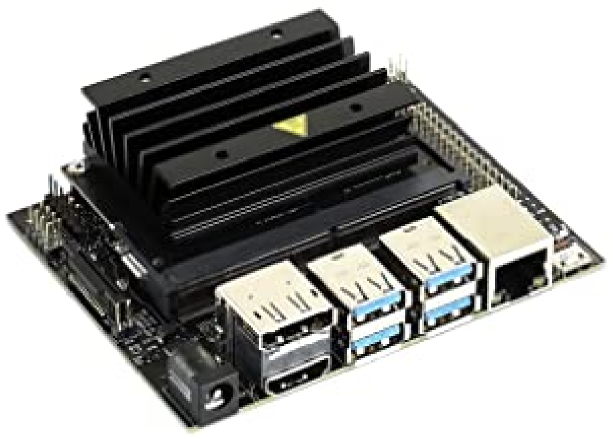


Figure 1.1: The NVIDIA Jetson Nano

It's simpler than ever to get started! Just insert a microSD card with the system image, boot the developer kit, and begin using the same NVIDIA JetPack SDK used across the entire NVIDIA Jetson^{TM} family of products. JetPack is compatible with NVIDIA's world-leading AI platform for training and deploying AI software, reducing complexity and effort for developers.

Ports & Interfaces

1. 4x USB 3.0 A (Host)
2. USB 2.0 Micro B (Device)
3. MIPI CSI-2 x2 (15-position Camera Flex Connector)
4. HDMI 2.0
5. DisplayPort
6. Gigabit Ethernet (RJ45)
7. M.2 Key-E with PCIe x1
8. MicroSD card slot
9. (3x) I2C, (2x) SPI, UART, I2S, GPIOs

Specifications

GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (not included)
Video Encode	4K @ 30 — 4x 1080p @ 30 — 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 — 2x 4K @ 30 — 8x 1080p @ 30 — 18x 720p @ 30 (H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI and display port
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I2C, I2S, SPI, UART
Mechanical	69 mm x 45 mm, 260-pin edge connector

Table 1.2: Specifications

1.3.2 GSM Module SIM800A

SIM800A is a complete Quad-band GSM/GPRS solution in a SMT type which can be embedded in the customer applications. SIM800A can transmit Voice, SMS and data information with low power consumption. With tiny size of 24*24*3mm, it can fit into slim and compact smart traffic control system design. Featuring Bluetooth and Embedded AT, it also allows total cost savings.

Interfaces

1. 68 SMT pads
2. Analog audio interface
3. PCM interface(optional)

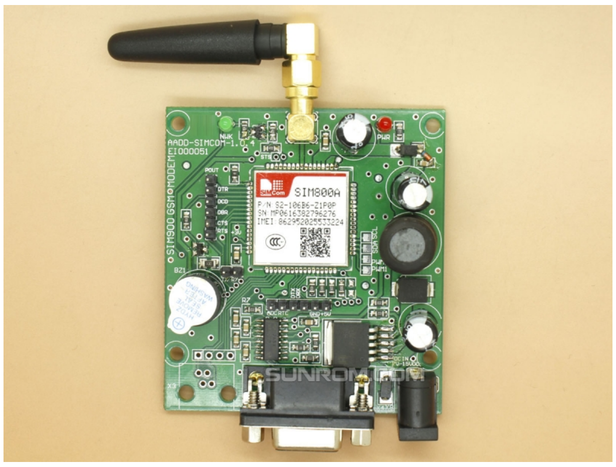


Figure 1.2: GSM Module SIM800A

4. SPI interface (optional)
5. RTC backup
6. Serial interface
7. USB interface
8. Interface to external SIM 3V/1.8V
9. Keypad interface
10. GPIO
11. ADC
12. GSM Antenna pad
13. Bluetooth Antenna pad

Software Specifications

1. 0710 MUX protocol
2. Embedded TCP/UDP protocol
3. FTP/HTTP
4. MMS
5. E-MAIL
6. DTMF
7. Jamming Detection
8. Audio Record
9. TTS (optional)
10. Embedded AT (optional)

Specifications for SMS via GSM/GPRS

1. Point to point MO and MT

2. SMS cell broadcast
3. Text and PDU mode

Specifications for voice

1. Tricodec
2. Half rate (HR)
3. Full rate (FR)
4. Enhanced Full rate (EFR)
5. AMR
6. Half rate (HR)
7. Full rate (FR)
8. Hands-free operation (Echo suppression)

1.3.3 Timer Module TM1637

TM1637 DISPLAY module is used for displaying numbers. The module consists of four 7- segment displays working together. The module working is based on 'TM1637' IC present internally and hence the name 'TM1637 display'.

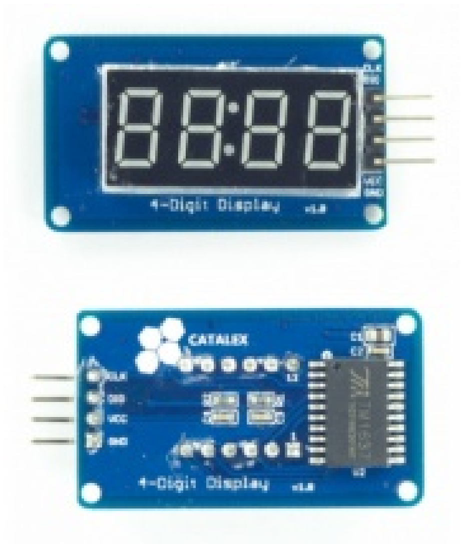


Figure 1.3: The TM1637 Timer Module

Pin Configuration

1. VCC: Connected to power source.
2. GND: Connected to ground.
3. DIO: Data Input/output pin
4. CLK: Clock pin

Features and Specifications

1. Two wire interface
2. Eight adjustable luminance levels
3. Grove compatible interface (3.3V/5V)
4. Four alpha-numeric digits
5. Potable size
6. Operating voltage: 3.3V – 5.5V
7. Operating current consumption: 80mA
8. Operating temperature: -10^{o}C to +80^{o}C

1.4 Software in-Depth

Now that we have covered the hardware used, it is time to understand all the software packages required to get the hardware setup and ready

1.4.1 Jetson Nano Setup

In order to install an operating system on the Jetson Nano, the following steps must be undertaken:

1. Download the Jetson Nano Developer Kit SD Card Image, and note where it was saved on the computer.

2. Download, install, and launch SD Memory Card Formatter for Windows. (Refer Fig 1.4).

3. Select card drive and Select “Quick format”. Leave “Volume label” blank.

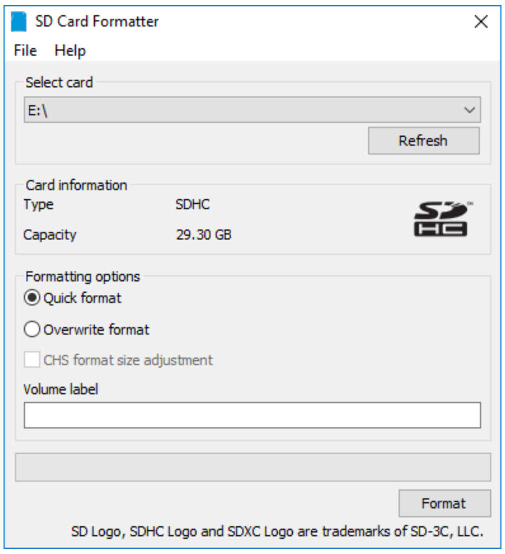
4. Click “Format” to start formatting, and “Yes” on the warning dialog

5. Download, install, and launch Balena Etcher. (Refer Fig 1.5)

6. Click “Select image” and choose the zipped image file downloaded earlier.

7. Click “Select drive” and choose the correct device. Click “Flash!” It will take Etcher about 10 minutes to write and validate the image if your microSD card is connected via USB3.

8. After Etcher finishes, Windows may let you know it doesn’t know how to read the SD Card. Just click Cancel and remove the microSD card.



SD Card Formatter

File Help

Select card

E:\

Refresh

Card information

Type SDHC

Capacity 29.30 GB

Formatting options

[✓ Quick format]

Overwrite format

CHS format size adjustment

Volume label

Format

SD Logo, SDHC Logo and SDXC Logo are trademarks of SD-3C, LLC.

Figure 1.4: Get SD card ready for a flash

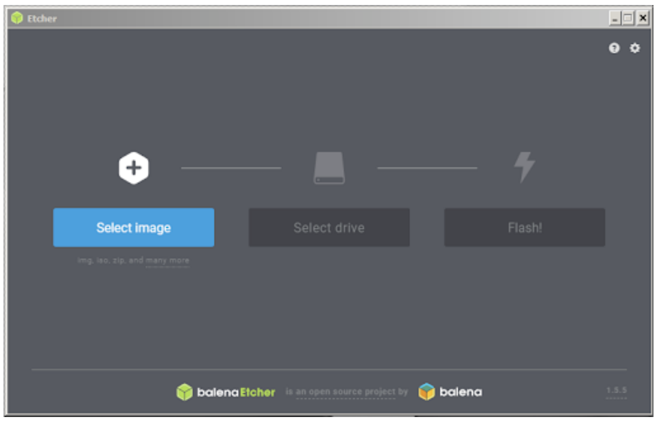

Etcher
Select image
Select drive
Flash!
img, iso, zip, and many more
balenaEtcher is an open source project by balena

Figure 1.5: Balena Etcher Interface

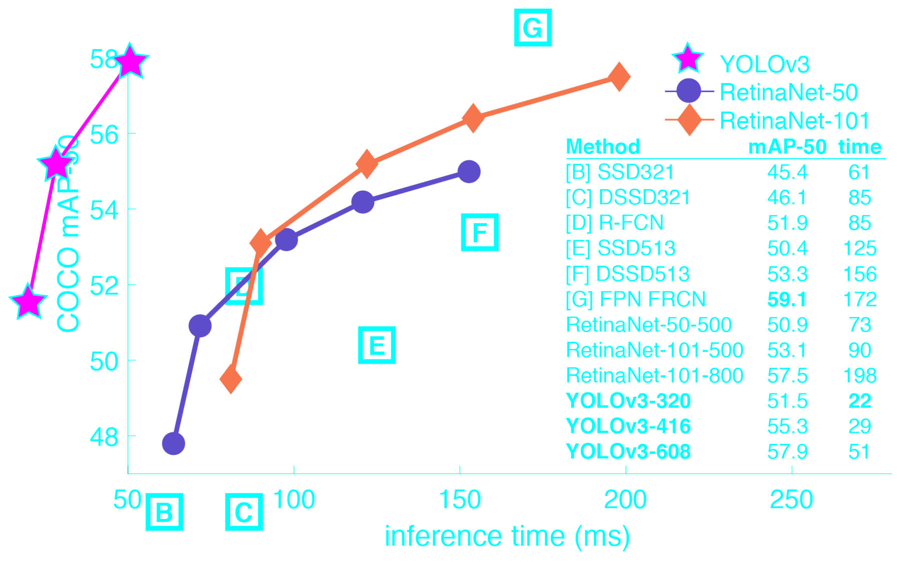
1.4.2 YOLO: You Only Look Once

You only look once (YOLO) is a state-of-the-art, real-time object detection system. On a Pascal Titan X it processes images at 30 FPS and has a mAP of 57.9% on COCO test-dev. Comparison to Other Detectors

YOLOv3 is extremely fast and accurate. In mAP measured at .5 IOU YOLOv3 is on par with Focal Loss but about 4x faster. Moreover, you can easily tradeoff between speed and accuracy simply by changing the size of the model, no retraining required!

Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections.

YOLO use a totally different approach. YOLO applies a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figure 1.6: YOLOv3 performance versus similar popular models

predicted probabilities.

YOLO has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN. YOLOv3 uses a few tricks to improve training and increase performance, including: multi-scale predictions, a better backbone classifier, and more.

1.4.3 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. We will be using OpenCV for receiving frames from a camera module and on the Python Linux platform. Jetson Nano comes pre-installed with opencv-python.

1.4.4 TensorFlow

YOLOv3, requires a tensorflow and darknet backend, as per its documentation. Largely, tensorflow is the penultimate library for deep learning and AI computation. TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy. If you need more flexibility, eager execution allows for immediate iteration and intuitive debugging. For large ML training tasks, use the Distribution Strategy API for distributed training on different hardware configurations without changing the model definition.

Installing Tensorflow on Jetson Nano

Installing Jetson Nano Tensorflow Binaries can be found here:

https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html

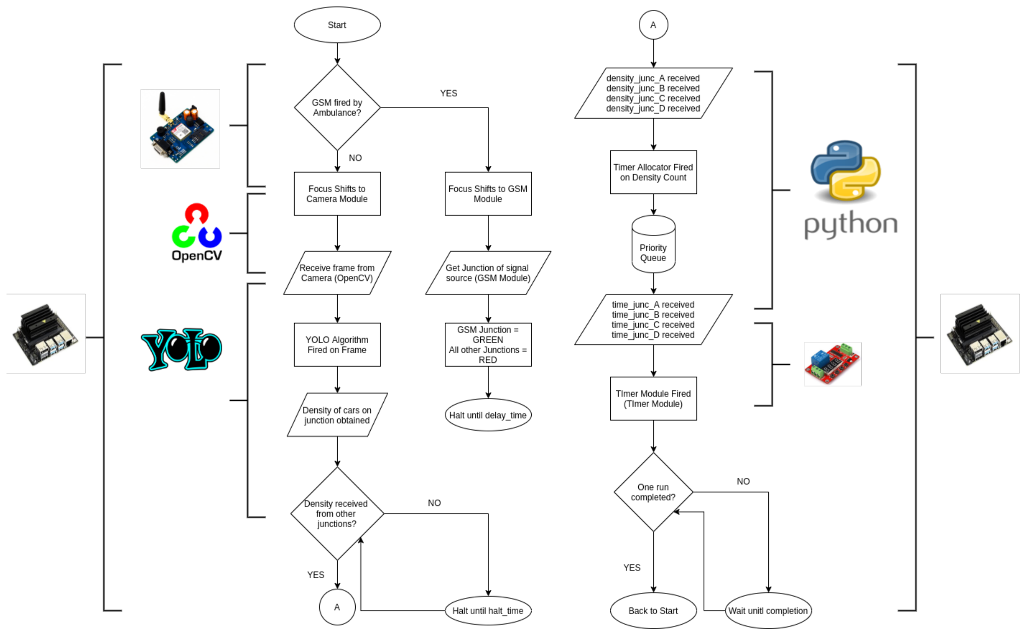
WARNING: There might be an issue installing the h5py binary when executing the commands for tensorflow dependencies. Should such an issue arise, simply remove h5py from the list and install it separately via

```
sudo apt-get install h5py
```

Chapter 2

System WorkFlow

2.1 Flow Diagram



Start
GSM fired by Ambulance?
YES
NO
Focus Shifts to Camera Module
Focus Shifts to GSM Module
Receive frame from Camera (OpenCV)
Get Junction of signal source (GSM Module)
YOLO Algorithm Fired on Frame
GSM Junction = GREEN All other Junctions = RED
Density of cars on junction obtained
Halt until delay_time
Density received from other junctions?
NO
YES
A
Halt until halt_time
A
density_junc_A received
density_junc_B received
density_junc_C received
density_junc_D received
Timer Allocator Fired on Density Count
Priority Queue
time_junc_A received
time_junc_B received
time_junc_C received
time_junc_D received
TImer Module Fired (TImer Module)
One run completed?
NO
YES
Back to Start
Wait unitl completion
OpenCV
python

Figure 2.1: Flow Diagram for Smart Traffic Control System

2.1.1 Main Pipeline

The flow diagram for the system can be seen in Fig 2.1. Let us focus for the main branch for now. First off, assuming no interrupt from GSM module is fired, OpenCV calls upon the camera module to render a frame and return it back to the Python interface. Once that is done, the frame received is input to the modified YOLOv3 model that has been modified to return the number of vehicles that are present on a junction. The beauty of this form of detection is that even if the number of cars that are really present on the junction is incorrectly identified, the timing allocation ahead in the pipeline is interested in the bigger picture: the density of cars in the area than than the actual number, any minor inaccuracies in the model are largely ignored. This operation is similarly performed on all three junctions. One master Jetson demands values from the three Jetson slave modules and waits until the values are received. The latency values and discussions are noted in Results and Conclusions section.

Once the densities are obtained, a timing allocator built solely in python, performs allocation of the times the green light will be on for each junction. To avoid a starvation problem, a minimum time limit has been allocated if the time calculated becomes too low for the vehicles to clear practically. Based on the density values, a priority queue is generated, that decides which whether an interrupt has to be executed or the normal execution continues. Once the time values are fixed, the timer module is fired that counts down the given time until completion. The master is also required to communicate the timers to the slaves. Due to lack of separate hardware, we simply for testing purposes, communicate the master nano with itself. This is an indication that the overhead values we cite must be taken with a grain of salt.

2.1.2 The Interrupt System

We can see in the Fig 2.1 that there exists a new interrupt system that can stall the main pipeline. It has the highest priority in timing allocation, irrespective of the values that are decided to be allocated for that run. An ambulance can now send a text message on a predefined number, that, if contains a very specific string of characters that will only be revealed to ambulance and law enforcement agencies, will trigger the Jetson to use the GSM module and find out which junction has an ambulance coming through it. Once this is determined, we may push a predefined time into the allocation queue, with the junction having the ambulance having first go and the highest time. Once this time passes, normal operation can be resumed. This solution may not be the most optimized solution, but takes into account multiple agencies and also leaves room for improvement.

Chapter 3

Results and Inferences

3.1 Results

3.2 Issues Encountered

There were a lot of issues that we faced when dealing with this system. Some were minor issues easily fixed like throughput and memory, but some were potent and needed attention. Potent issues are discussed here.

3.2.1 Jetson's Capabilities to the test

We first started off with a Raspberry Pi 3B+ and YOLOv3 without the tiny weights. We knew that the Pi would not be able to handle detecting all of the objects, but were still optimistic. As expected, the Pi crashed as soon as the model weights were loaded. We needed a new solution, so one that was proposed was to only detect vehicles and disable the bounding boxes around other objects. This obviously reduced a lot of processing, but was still not enough to sustain on a Pi. The next step was to use YOLOv3 tiny, which was still proving to be too tough for the Pi. A decision was made to switch to the Jetson Nano due to its fame in AI computation. The model still failed to exeucte and crashed upon loading weights. In order to reduce the stress on hardware and to concurrently reduce throughput, the following changes were made:

1. Porting the YOLO model from Tensorflow to TfLite, a sister library to tensorflow optimized for embedded devices.
2. Any form of bounding boxes rendered were removed: no frame can now be visualized. The model had garnered enough faith and we did not need to observe the detection output.
3. The model was made to return only the count of the number of vehicles on the

junction. This made it easier for data transfer within the pipeline.

4. A new library NNPACK was found that further reduced the computation. Since we were unable to port NNPACK to run with our model in-code and had to rely on the terminal to run NNPACK, this modification was removed in the final build.
5. A change was introduced in the input to the YOLO model. Instead of the model taking as input a video feed, we unanimously agreed to take an image of the junction at a specific time interval and use that as input for processing.

Taking the following steps greatly increased efficiency in the model and improved throughput tenfold.

3.2.2 Jetson operational voltage

As it so happens the Jetson Nano operates at 3.3V logic level to reduce the power consumption. However many sensors run at 5V logic level and so an interfacing level-shifter had to be introduced that managed this for us.

3.2.3 Jetson lacks wifi module

The Jetson nano lacks an implicit wifi module but does support a module externally. It also supports ethernet connections via the RJ45 port. So any internet connectivity required for installing packages needs to be handled either with an ethernet connection or downloaded externally, copied to the jetson and then built from source.

3.2.4 Lack of hardware for Junction testing

We did not have the hardware to test a master-slave directive for a junction but we managed to make the jetson communicate with itself to make demonstration possible. This however, does not portray the actual throughput of the system and hence should be taken with a grain of salt.

Chapter 4

Conclusions and Future Scope

4.1 Conclusion

We achieved most of what we set out to do: we built a dynamic feedback loop for a traffic control system that helps solve most of the issues that the current system has at the foundation. We can make the system more dynamic, which is a proof of its customizabiliy and cost-effectiveness. It also requires minimum maintenance, as most of the heavily lifting is done by software libraries that need only to be updated on a regular basis. There were many issues encountered along the way and we solved them as a team, working together for a common incentive. The throughput count can be stated as follows:

Particulars	Time
Time for normal run-through(One junction)	2.25s
Time for normal run-through(Four junctions)	10.31s
Interrupt processing time	1.54s
Pipeline processing and latency	0.18s
Timing Allocation	0.013s

Table 4.1: Throughput and Latency Information for System

4.2 Future Scope

A lot can be improved and a lot can still be learnt from the system currently proposed. A few have been listed for the convenience of the teams who want to carry the torch forward.

1. We can try and understand NNPACK and port it to support YOLO on the jetson in-code. This can reduce computation to 0.23s for the detection of density of vehicles.

2. We may attempt to come up with a better alternative to the GSM module that also involves image processing. We may detect the flashing lights from ambulances or law-enforcement vehicles to automatically fire an interrupt
3. We may try and reduce the load on the Jetson by making the JetPack OS run in CLI. This will stop all processes pertaining to GUI of the OS and help push our kernel up the OS priority queue.
4. We may remove embedded hardware from the equation and use it simply to get frames from the camera and push the frame to the cloud where the model resides.
5. Dynamic operation can be introduced to the interrupt system to support nearby topology such as IT sectors that can contribute heavily to the influx of traffic during a certain time of day. This can be done using Machine Learning
6. Interrupts can also be added to support accidents and mishaps on the road for easy management.

References

[1] Redmon, Joseph, and Ali Farhadi. *"Yolov3: An incremental improvement."* arXiv preprint arXiv:1804.02767 (2018).

[2] Bradski, Gary, and Adrian Kaehler. *"OpenCV."* Dr. Dobb's journal of software tools 3 (2000).

[3] Alsing, Oscar. "Mobile Object Detection using TensorFlow Lite and Transfer Learning." (2018). [Online], Available `http://www.diva-portal.org/smash/get/diva2%3A1242627/FULLTEXT01.pdf`

[4] Jetson Nano Developer Kit. (2020, February 25), [Online], Available, `https://developer.nvidia.com/embedded/jetson-nano-developer-kit`.

[5] SIM800. (n.d.), [Online], Available, `https://simcom.ee/modules/gsm-gprs/sim800/`

[6] TM1637- Grove 4 Digit Display Module Datasheet, Pinout, Features and Working. (n.d.), [Online], Available, `https://components101.com/displays/tm1637-grove-4-digit-display-module`.