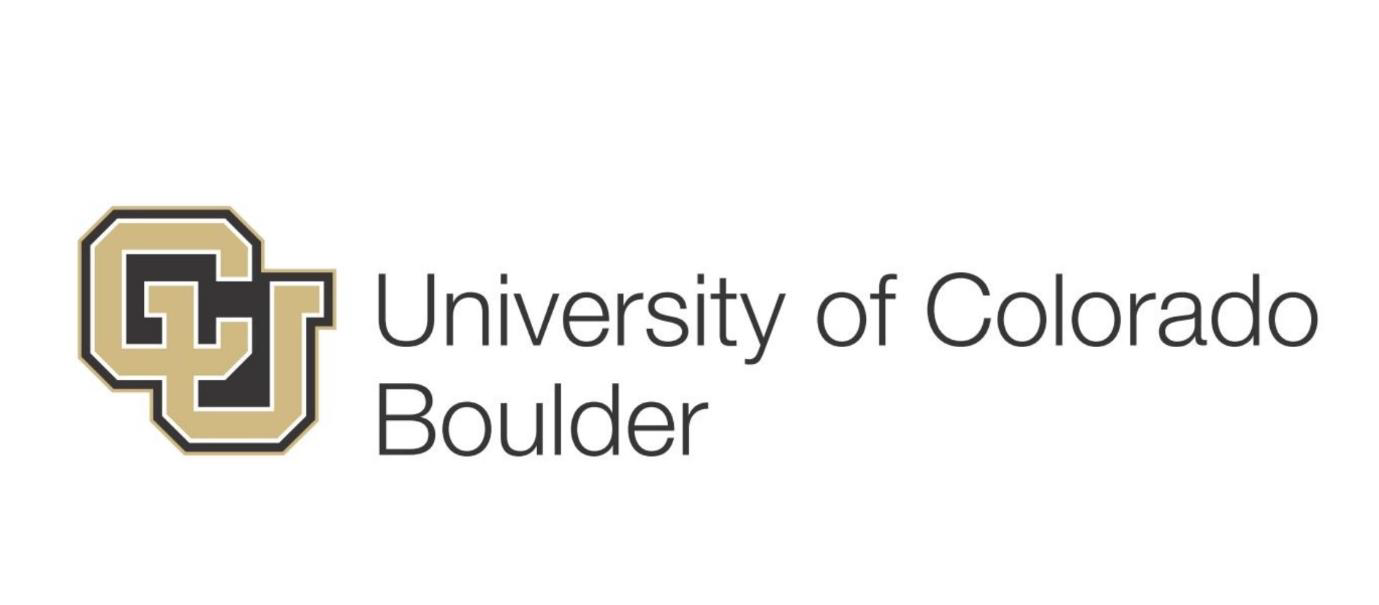
****

**Embedded Machine Vision and Intelligent Automation**

**Project**

**Autonomous Car Prototype**

**Github-** [**https://github.com/deep6000/Autonomous-Car-Prototype**](https://github.com/deep6000/Autonomous-Car-Prototype)

* **Deepesh Mahendra Sonigra**

**Autonomous Car Prototype**

**Aim -**

To design an Autonomous car prototype which could sense the environment variables like the road lanes, vehicles, road signs and traffic signals.

**Introduction-**

* An Autonomous car is defined as a robotic vehicle, which without any human interaction could fulfil all transportation capabilities.
* The autonomous car has a camera installed in the front side of the car. The camera is installed in such a way that it captures the lanes, vehicles, pedestrians, traffic signals, and road signs.
* This camera can be used to capture frames, and environmental variables like the lane, vehicles and traffic signs can be detected.
* A human driver senses all these parameters and then take a decision accordingly on what could be the next step.
* This is what an autonomous car does. It senses the variable, analyse it’s value and then take decision accordingly.
* An autonomous car consists of various sensors integrated with camera to give cameras inputs it cannot calculate or takes a long time to calculate.
* Sensors like objection detector to detect objects very close or lightness sensor to give light values could be very important parameters for camera calibration.
* The prototype takes an input video as an argument and starts processing it.
* All the processed frames are written into an output video.
* The autonomous car consists of the 4 modules, namely, Lane detection module, vehicle detection module, traffic sign detection module, and pedestrian detection module.
* All these modules are integrated together on a Jetson Nano using multi-threading and scheduling to achieve desirable frame rate.
* The lane detection module detects the lane, vehicle detection module detects the vehicle, traffic signal detection module detects the signal and pedestrian module detects the pedestrians
* This prototype is limited to detection only and does not involve processing of that data.
* Also the prototype limits itself with one jetson nano. With resources being limited the frame per second cannot be achieved to exactly 30 FPS but is achieved as close as possible.
* Adding more modules will definitely decrease the processing speed thus affecting the real time capabilities of the system.

**Functional Requirements**

1. **Lane Detection**

* Lane Detection forms an integral part of autonomous car as it plays an important role in preventing accidents. The system should detect the white and yellow lanes on the road real-time and mark it on the frame.
* The coordinates can then be used to compare the location of the vehicle with the location of the lane. If the vehicle intersects the lane we could warn the system about it.

1. **Vehicle Detection**

* The vehicles are the moving objects on the road and most prone to accidents. The system should be able to detect the vehicles in its region of interest and should be able to denote it on the frame by drawing a rectangle.
* These locations of the vehicle could be further used to make decisions of keeping considerable distance between those cars.

1. **Traffic Signal Detection**

* Traffic Signs provide assistance to driving by giving a feedback of when it is safe to go ahead. The system should be able to detect the traffic signs in the frame which are facing the vehicle.

1. **Pedestrian Detection**

* The pedestrians form the most important and critical environment variable when it comes to autonomous cars. The system should be able to detect pedestrians in its region of interest.

1. **Real-Time Processing**

* Autonomous Cars are critical systems and they should produce results real- time.
* The system should be able to run all modules in parallel and detect all the characteristics required and to plot it on the frame. As this system takes an input video, should be able to achieve fps close to the input video.

1. **Output Video**

* The system takes input video as an argument as doesn’t have a live camera. The system should be able to write a video with all the sensing parameters on the frame

1. **Machine Vision and Machine Learning Algorithm**

* The system should be able to use the machine vision and machine learning algorithm to detect objects so that it’s more accurate and real-time

**Machine Vision and Machine Learning**

**Machine Vision Algorithm**

The system must be able to use all the machine vision algorithms required for the prototype to reduce processing time, increase efficiency and accuracy.

The following are some of the Machine Vision algorithms which can be used for prototyping an autonomous car.

1. Hough Line transform - This transform can be used to detect particular shapes and could be used to detect straight lines which form the lanes on the roads. Hough Lines depends on the parameters passed to it and can be manipulated accordingly.
2. Image Cropping. It is similar to the region of interest but in this input image is cropped based on the requirement. For example while detecting lanes we could crop the sky and while detecting the signal we could crop the road.
3. Extracting Pixel values - These are basically inRange APIs used to extract a specific range of pixel values(colors).
4. ROI- Region of interest is the part of the frame, only where the expensive process should be applied
5. Resizing the image - Resizing the image involves increasing and reducing the resolution. Some cameras have very high resolution and resolution can be downgraded to improve processing speed.
6. Bitwise Operators - These are very useful machine vision algorithms as these allow to manipulate the images the way you want, adding images, defining ROI, and etc.

**Machine Learning Algorithm**

Machine Learning algorithm means either training your system or using data from a trained system. This is very helpful as these are able to detect objects, extract various features from the frame.

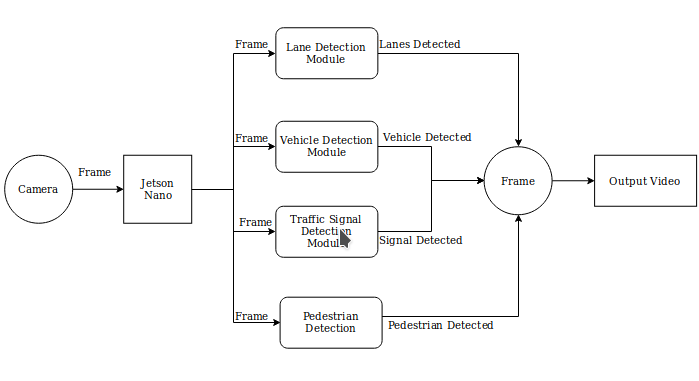
The system should be able to make use of these machine learning algorithms to detect the required features from the frame

1. Haar Cascade- Haar Cascade for object detection is a very effective machine learning algorithm. Haar Cascade algorithms can detect various objects based on how they are trained. Haar Cascade Algorithms are trained using a large database of positive and negative images. Haar Cascade can be used for face detection, vehicle detection, Traffic Signal detection etc.
2. HOG descriptor - HOG descriptor is very effective machine learning algorithm for human detection.This machine learning algorithm can be used to detect pedestrians on the road.

**Function Design Overview**

The function design overview consists of different modules processing the frames for sensing the road environment variables and integrating them for further decision making.

**Block Diagram**

****

The modules Include as follows

1. Lane Detection
2. Vehicle Detection
3. Traffic Signal Detection
4. Pedestrian Detection

The camera captures the frame and the frame is passed through each module to process it.

The frame is cloned in each module and is processed individually. Coordinate points for line, vehicle, signal and pedestrian are calculated.

These coordinates are then used on the original frame to plot the lanes, vehicles, traffic signals, and pedestrians.

**Lane Detection**

The lane detection module detects the lane ahead of the vehicle camera. This is achieved by using Hough Lines which detect the straight lines in the frame.

**Block Diagram**



**Algorithm**

**Cropping Frames**

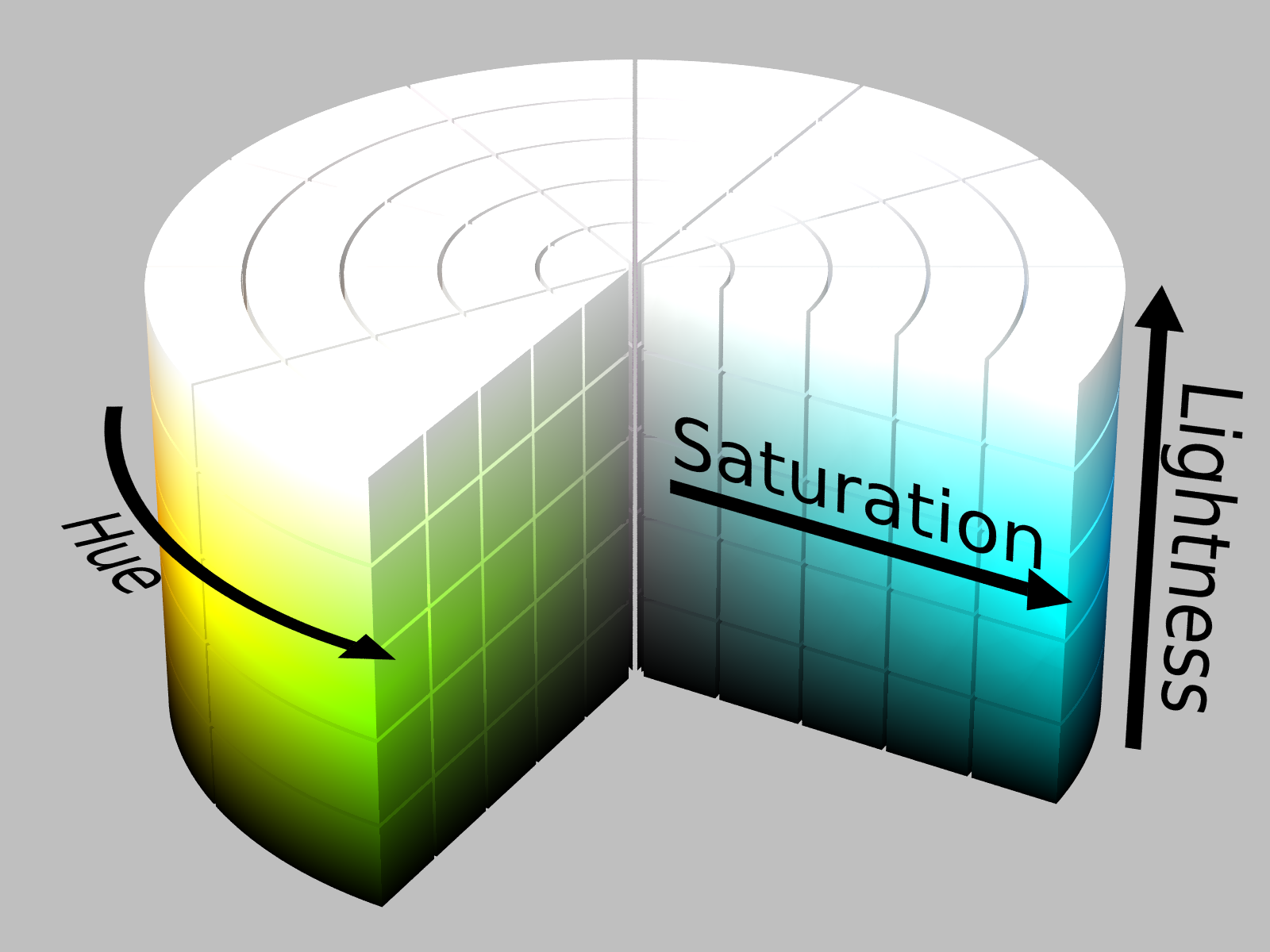
Firstly the frame is read from the video. The camera is placed in such a way that it covers roads, vehicles and traffic signs and pedestrians walking on the road.

For lane detection we only need the road, so the frame is cropped according to the requirement, the sky is cropped out to avoid unnecessary processing of the frame.

**Extracting White and Yellow**

This module assumes that the lanes on the road are white and yellow. The idea is to extract these colors from the frame.

The white color is extracted from the HLS (hue, saturation, lightness) color model. The L is HLS is lightness which gives the white color in the frame. HLS is best to extract the white. The following image shows that.

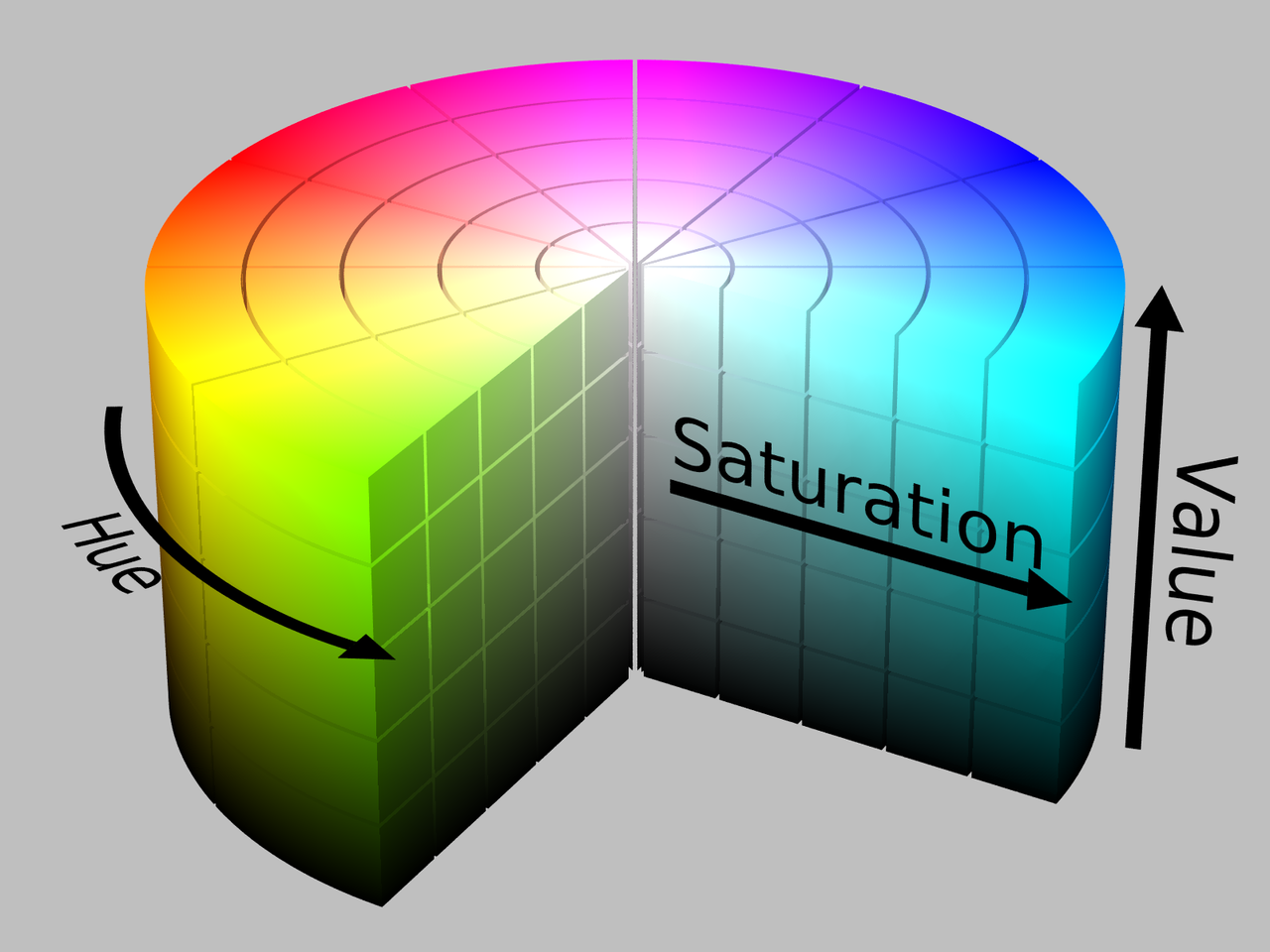
Input Image HSL Model HSL image

The output of the converting frame into HLS model and extracting white is shown below.



White Extracted Image

The yellow color is extracted from HSV(hue, saturation, value) color model. The Hue Saturation Value serves best for extracting individual colors.

Input Image HSV Model HSV image

The output of the converting frame into HSV model and extracting Yellow is shown below.

Yellow Extracted Image

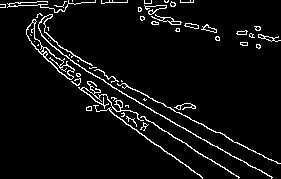
These two images are bitwise-ored so we get a frame with whit and yellow lanes.

This technique can also be used to differentiate the lanes as traffic rules for both the lanes are different.

**Noise Removal and Edge Detection**

The noise from the image is removed by passing it through a denoise filter that is the gaussian filter. This smoothens the image and removes all the unwanted noise from the image.

The Canny edge detector detects the edges which makes it easier to identify lines in an image.

Denoise (Gaussian Filter) Canny Edge Detector

**Line Detection**

Hough Lines, a machine vision algorithm is used to detect the straight lines in an image. Hough Lines detect all the straight lines in an image based on the parameters.

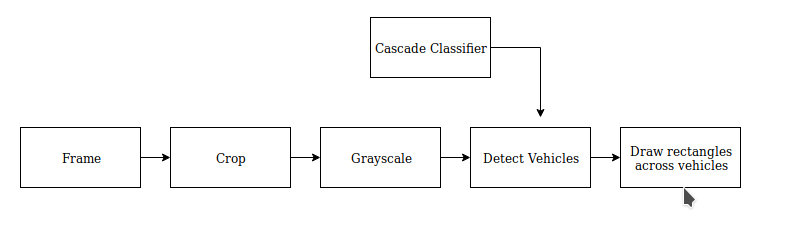
The problem with this is it detects all the lines and would detect multiple lines for the lanes. This could cause issues while calculating the coordinates of the lanes.

Thus averaging of the lanes is performed to get one line per lane, the right lane and the left lane. The averaging is done based on the x-y coordinates of the lines and one line is drawn on the frame which represents the lane of the road.

**Vehicle Detection**

The vehicle detection module detects the vehicles in the frame. Using detected coordinates we could know where the vehicles are and how far they are. This is done by using Haar cascade algorithm.

**Block Diagram**



**Algorithm**

**Cropping Frames**

Firstly the frame is read from the video. The camera is placed in such a way that it covers roads, vehicles and traffic signs and pedestrians walking on the road.

For Vehicle detection we need part of the frame just above the road and below the sky. So accordingly the sky and lower part of the frame is cropped.

**Grayscale**

The image is converted into grayscale to reduce the processing time as only 1 channel needs to be processed The Haar Cascade is trained to detect vehicles in gray scale as well.

**Detect Multiscale**

Based on the cascade classifier the multiscale API detects objects of that shape and returns the coordinates of those shapes. Here a car cascade classifier is passed as an input to this API. This detects all the cars in the frame.Based on the coordinates a rectangle is drawn across the detected objects.



**Sign Detection**

The traffic signal are detected so that the autonomous car could detect and follow the traffic signals.

**Block Diagram**

****

**Cropping Image**

The road part of frame is cropped as traffic signals are at a good height and road doesn’t need to be processed for this.

**Cascade Classifier**

The Haar cascade classifier is used to detect the traffic signals in a frame

****

**Machine Vision and Machine learning Analysis**

**Region of Interest**

The region of interest forms an important parameter in image processing because in most of the cases the processing can be expensive in terms of time and region of interest help us avoid that.

Region of Interest is region where mainly the characteristics needed to be extracted are located and the processing is performed only on those regions to save processing time.

For example in lane detection see the following image.



The white trapezium denotes the ROI for the lanes. As you can see all the lanes of interest are covered in the ROI for lanes and all expensive operations are performed in this region of interest. Doing so will definitely increase the processing speed and improve the frame rate.

On doing analysis on lane detection the frames per second with and without ROI are as follows

**With ROI : 27 fps**

**Without ROI : 16 fps**

The camera orientation is such that it covers the traffic signals as well. So most of the sky is also covered in the frame. This part of the frame is not needed for analysis of the vehicle. Hence the image containing the sky is cropped.



As you can see most of the sky is cropped and this is part of the frame where vehicle is appear. The frame size is reduced to almost ⅓ by doing so. This means that the processing time (detect Multiscale) is reduced to ⅓ of the actual time.

Similarly for Traffic Signal the road forms the unnecessary part of the frame. Thus the ROI for the signal would be part which includes some of the road and rest of the sky. In the image you can see the traffic signals are viewed and this reduces the frame to ⅓ as well.

****

**True and False Positive/Negatives**

The following figures were observed for car and traffic signal over a 50 images written randomly

**Vehicle Detection**

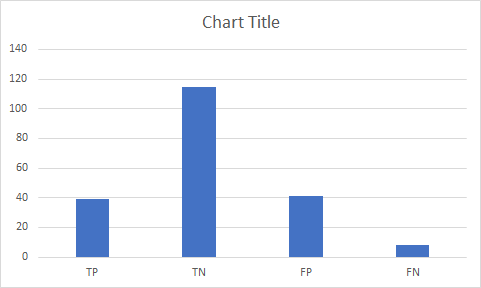
True Positive - Car was ahead and was detected successfully

True negative - Car was not there and was not detected

False positive - Car was not there but it was still detected

False negative - Car was there but not detected

The numbers are the number of objects in 50 frames.



According to the data TN>TP>FP>FN. This is desired result

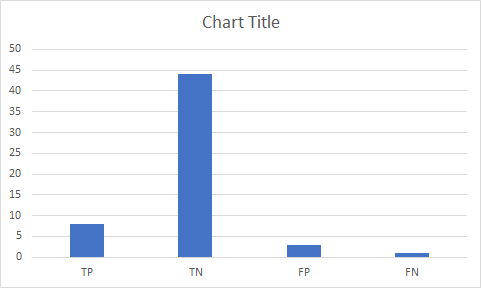
**Traffic Signal Detection**

True Positive - Signal was ahead and was detected successfully

True negative - Signal was not there and was not detected

False positive - Signal was not there but it was still detected

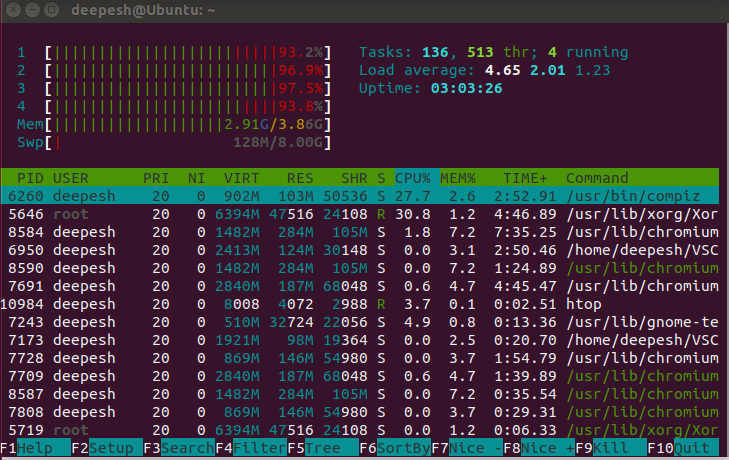
False negative - Signal was there but not detected

****

According to the data TN>TP>FP>FN . This is desired result

**Core Usage**

The jetson nano has 4 cores and idea is to make use of each core to the fullest so that we could get response real-time Following image shows the usage of core for running the executable.

****

All the 4 cores are more than 90 percent being used. This means that jetson is almost running at full power and faster response time is expected.

**Test Cases**

**Cores**

**Core-Split**

The core split technique where each core is assigned one thread and it performs that task.

This technique was least efficient as not all tasks took equal time and because of the semaphores the cores used to sit ideal until the semaphore was released.  
The FPS for 3 services came upto 9 frames per second

**Mutli-threading**

The multithreading technique with no core split was more efficient than the core split technique as all the cores were used all the time to their fullest. The core distribution was handled by the operating system and users had no control over the priorities of the thread execution.

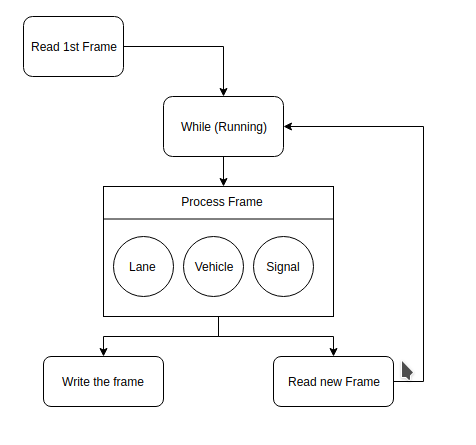
The FPS for 3 services came upto 15 frames per second

**FIFO-Scheduling**

The FIFO-Scheduling technique was found to be the most efficient technique as we could set the priority of the thread based on our requirement. The cores are utilized to their fullest and higher processing speed was achieved. As I was able to only get 2 services working with this technique comparing this with other techniques is quite difficult. But for 2 services the frame rate reached upto 22 frames per second.

**Scheduler**

Higher FPS can be achieved by designing a scheduler based on the frame processing criteria. The idea is that not much data changes between consecutive frames and processing each and every frame just adds overhead. Designing a scheduler properly could yield us FPS upto 27 FPS.

****

**Proof of Concept**

The system was able to detect the lanes, vehicles and traffic signals with a good efficiency of very less false negative percentage.

****

The white lane ahead of the car is perfectly detected and marked with red lines.

****

The car ahead of our car is detected and marked using a rectangle

****

The traffic signal is detected and marked with blue rectangle

****

The lanes are detected on slight curves as well

**To be performed before the demo**

* Pedestrian detection - Implemented individually. Needs to be integrated with all the other modules
* Designing a scheduler and making use of scheduler-fifo. Currently working for 2 threads with FPS= 27. Needs to be integrated for 4 threads.

**Conclusion**

* The system could effectively detect lanes, vehicles and signals in the video.
* The frames per second was close to what was desired.
* The system was able to react in real-time by making use of multi-threading and making all 4 cpu cores to the fullest.
* Machine Vision algorithms like the extracting colors, image masking , defining ROI, image cropping helped achieve higher frame rates.
* Multi-threading and assigning tasks priority increased processing speed and made use of full core capabilities.
* Overall the system worked according to the expectations but there is so much scope for improvement.
* Though this system only involves a camera as a sensing parameter, I believe adding more sensors would definitely increase processing speed, yield better results, and would be more robust.
* For example the lightness sensor could help calibrate the camera parameters and detect objects more accurately.
* Adding more cameras definitely would yield more accurate results but the processing time can be increased to a greater extent if more sensors were integrated along with the camera.

**References**

[1] Car Detection - <https://github.com/anhydrous99/CarDetection>

[2] Scheduling Policy <https://pdfs.semanticscholar.org/f3e4/7327c8ae8238d7539439c3cbc67b856edaeb.pdf>

Classmates Siddhant Jajoo and Sarthak Jain - They shared their knowledge they learnt in Real-Time Embedded Systems class.

[3] Matt Compton (Colleague)

He shared his views on the number of frames needed to be processed.

[4] Pedestrian Detection

<http://mccormickml.com/2013/05/09/hog-person-detector-tutorial/>

[5] Lane Detection

<https://github.com/MichiMaestre/Lane-Detection-for-Autonomous-Cars/blob/master/LaneDetector/demo.cpp>