Traffic Light changed based on the density of project

- Divya Pattisapu (160040084)

- Naman Yadav (160100012)

- Umang Chhwachharia (160100014)

- Kartavya Singh (16D100007)

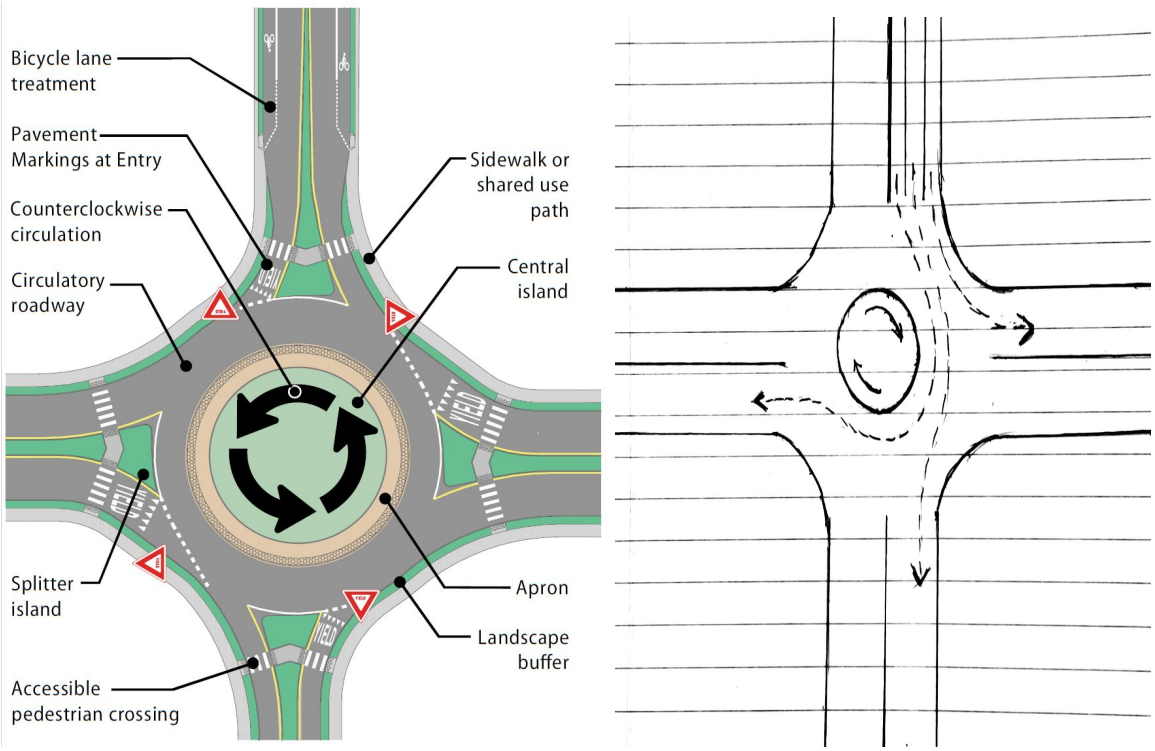
Aim

The aim of this project is to implement a dynamic traffic signal based on vehicle flow rate for simultaneous optimization of traffic signal parameters. For this, we have written a python script which takes as input real-time videos of four sides of the road in a four way traffic signals and outputs the processed video containing information regarding vehicle flow rate in each lane.

Motivation

Most traffic signals are configured using past data. However, it is difficult to make traffic predictions in crowded cities like Mumbai. A dynamic traffic signal which works based on current traffic density can save a lot of time and monitor the traffic as well.

Approach



The solution is attempted for a 4 way traffic signal so as to optimise the total waiting time of the vehicles.
The possible paths from a given lane is into the 3 other lanes as shown above.

The algorithm that we use is as follows

$$C_{min} = \frac{LX_c}{X_c - \left(\frac{\sum v_i}{s}\right)}$$

where
C_{min} = The signal cycle time
L = Total time lost per cycle
X_c = Intersection desired value to Capacity ratio
V_i = critical flow rate in i^{th} phase

$v_i(\text{demand flow rate for lane group } i) = \frac{V_i}{PHF}$

PHF = Peak Hour Factor = 0.95 for standard conditions
s = Saturation flow rate

X_c = 0.9 for standard conditions

L = N l_t where

l_t = total late time for one lane group
N = Number of phases = 4

$l_t = l_1 + l_2$ where

l_1 = Startup Loss Time(like Driver Reaction Time)
l_2 = Clearance Loss Time(for clearing vehicles currently moving at the intersection)
l_t is estimated to be around 3 s for Standard conditions.
Therefore L = 4*3 s = 12 s

For saturation flow s,

s = 3600/h vehicles/hr where
h = saturation head-way (time between 2 consecutive cars) = 2.3 s
Therefore s = 1565.2 vehicles/hr

$V_c = \Sigma V_i$ for all paths going into the intersectioni as shown below

Effective Total Green light time:

$g_t = C - Nl_t$

Effective Green light time:

$$g_i = g_t \frac{v_i}{v_c}$$

Actual Green light signal time for phase i:

$$G_i = g_i - Y_i + l_t$$

We estimate $Y_i = 3\ s$ for Standard conditions..

Effective Red light time:

$$r_i = C - g_i$$

Actual Red light time:

$$R_i = r_i - l_t$$

V_i is estimated from the videos from the traffic light cameras by counting the number of vehicles that are in a certain rectangle over all the lanes in lane group i.

Image Detection Algorithm

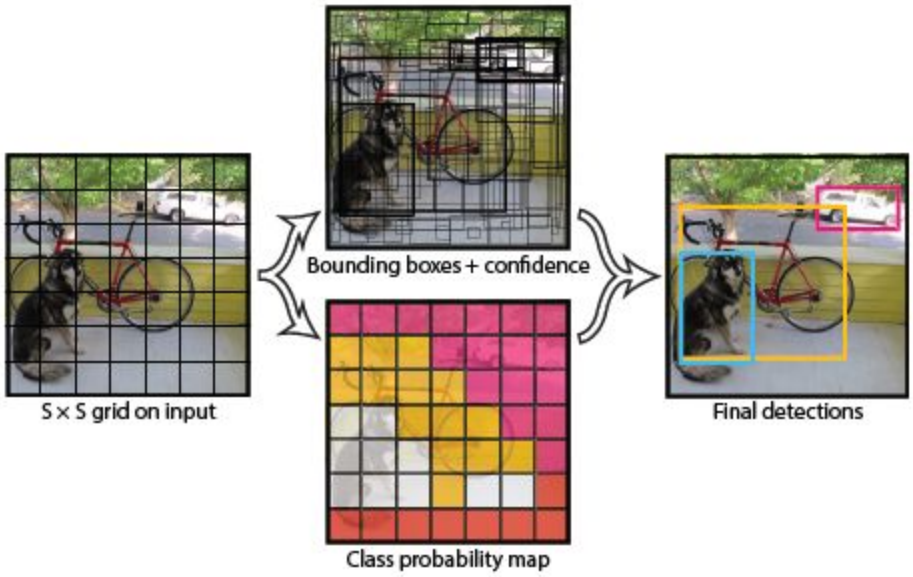
YOLO (You Only Look Once)

Yolo v1

YOLO stands for You Only Look Once. It's an object detector.

Idea of the algo

The system divides the input image into an S xS grid.



Our system models detection is expressed as a regression problem. It divides the image into an S S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S xS x(B *5 + C) tensor.

Each bounding box consists of 5 predictions: x, y, w, h, and confidence. x and y are scaled to relative coordinates with a scale of [0, 1] inside grid.

Yolo v2

After doing some clustering studies on ground truth labels, it turns out that most bounding boxes have certain height-width ratios. So instead of directly predicting a bounding box, YOLOv2 (and v3) predict off-sets from a predetermined set of boxes with particular height-width ratios - those predetermined set of boxes are the anchor boxes.

Yolo v3

The most salient feature of v3 is that it makes detections at three different scales. Detections at different layers helps address the issue of detecting small objects, a frequent complaint with YOLO v2. The upsampled layers concatenated with the

previous layers help preserve the fine grained features which help in detecting small objects.

The 13 x 13 layer is responsible for detecting large objects, whereas the 52 x 52 layer detects the smaller objects, with the 26 x 26 layer detecting medium objects.

YOLO is a fully convolutional network and its eventual output is generated by applying a 1 x 1 kernel on a feature map. In YOLO v3, the detection is done by applying 1 x 1 detection kernels on feature maps of three different sizes at three different places in the network. It has 75 convolutional layers, with skip connections and upsampling layers.
No form of pooling is used, and a convolutional layer with stride 2 is used to downsample the feature maps. This helps in preventing loss of low-level features often attributed to pooling.

Each box predicts the classes the bounding box may contain using multilabel classification. We do not use a softmax as we have found it is unnecessary for good performance, instead we simply use independent logistic classifiers. This formulation helps when we move to more complex domains like the Open Images Dataset. In this dataset there are many overlapping labels (i.e. Woman and Person). Using a softmax imposes the assumption that each box has exactly one class which is often not the case. A multilabel approach better models the data.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

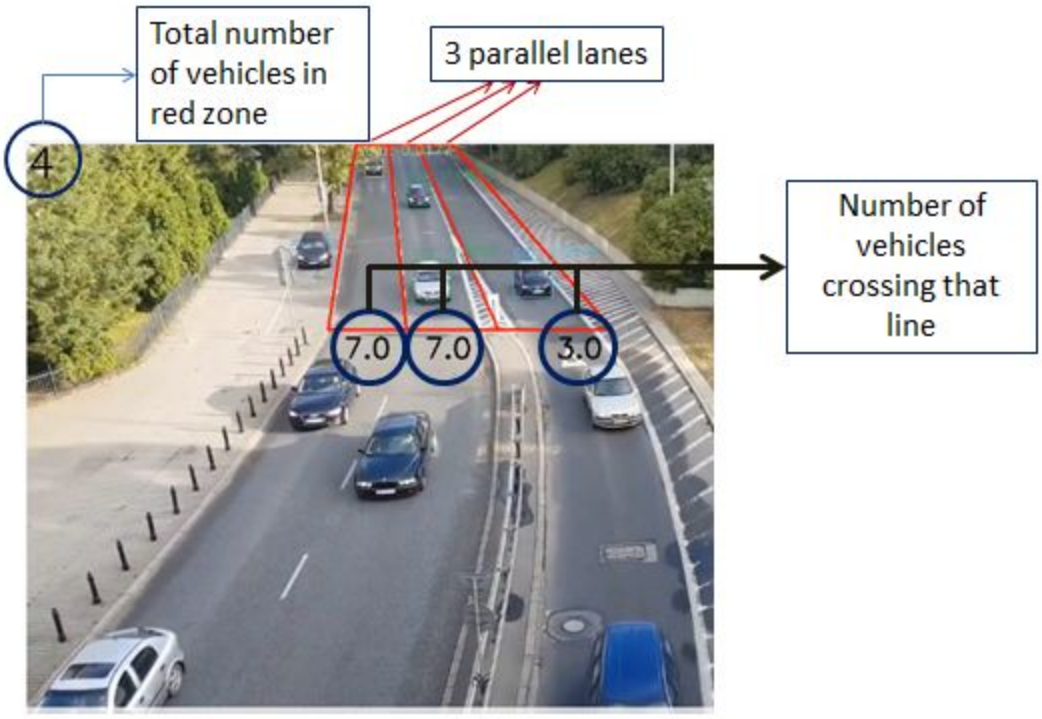
Table 1. **Darknet-53.**

We have used the YOLO v3 algorithm to identify all the vehicles in the videos from the traffic signal cameras.

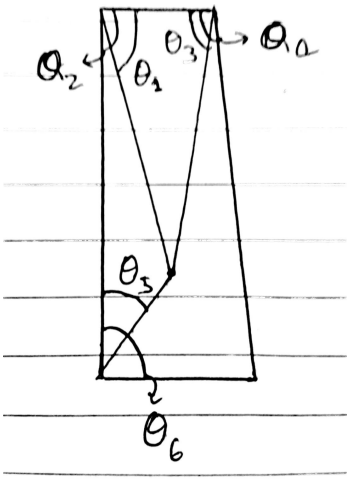
Final Output:

1. Only those cars were detected that were inside the red zone (showed in the figure below).
2. To identify the number of cars crossing the horizontal red line, the midpoint of the cars were sorted by their respective Y coordinates and stored in a list which was carried over to the next frame.
3. The Y coordinate of the bottommost cars in the current frame was compared with that of the previous frame. If a change of more than 5 in the negative direction was noticed, this means that the car has crossed the red line and gone out of bounds. Now the lowermost Y coordinate belongs to a different car.

4. This was done for every lane and the numbers of vehicles crossing red line were displayed below each lane.



To identify if a car is inside a lane or not following method was used:



1. Cosines of all the above given angles(in the figure shown above) were calculated using dot products of the adjoining lines.
2. A point is inside the area if:
    a. $\cos \Theta_1 > \cos \Theta_2$
    b. $\cos \Theta_3 > \cos \Theta_4$
    c. $\cos \Theta_5 > \cos \Theta_6$
3. All the centerpoints of cars that were in the lane were detected and the rest were were neglected.

Also, PCU (Passenger Car Unit) equivalent was calculated for each vehicle as following:

1. Trucks, buses = 3 PCU
2. Car = 1 PCU
3. Motorcycle = 0.5 PCU

The above shown output is recorded for each video of 75 s each. The numbers of all the 3 lanes summed up shows the number of vehicles passing the horizontal red line in 75 s. Multiplying this number by 48(=3600/75) would give the number of vehicles passing if an hour, which represents v_i .

Similar output for the videos of four way roads can be used to predict the heuristic for green light signal time using the algorithm aforementioned.
These are the number of vehicles that crossed each of the lanes in a total of 75 s, which have been obtained from the video.

Phase No	Lane 1	Lane 2	Lane 3
1	63	48	28
2	47	44	19
3	33	52	36
4	39	49	27

Calculating the flow as aforementioned:

Phase No	Lane 1	Lane 2	Lane 3
1	**3024**	2304	1344
2	**2256**	2112	912
3	1584	**2496**	1728
4	1872	**2352**	1296

Now calculating the V_i for each phase as the maximum of the flow among the lanes:

V_1	3024
V_2	2256
V_3	2496
V_4	2352

Now, based on these values of flow we can estimate cycle time and green time for each phase by using the equations mentioned in previous sections.
Snice, our video is very short and the flow is also continuous, which is not what actually happens in reality due to traffic signals, therefore values given in the above table are not accurate representatives of the vehicles passing in one hour.
Also, running the object detection and flow measurement algorithms over four one hour long videos will take a few days.

Conclusion

We have proposed an algorithm to detect vehicles in video and then calculate the number of vehicles crossing each lane. Calculated vehicle flow can be used to calculate cycle time of traffic signals using standard formula as mentioned in the above sections.