EEE 312 | Project Report

YOLO Based Traffic Speed Detection System

L-3, T-1

Section: A2

Group No: 03

Submission Date: 25th July, 2021

Submitted to,

Dr. S. M. Mahbubur Rahman

Professor, Dept of EEE, BUET

Shahed Ahmed

Lecturer, Dept of EEE, BUET

Submitted by,

Name: Monirul Islam
ID: 1706042

Name: Fariza Siddiqua
ID: 1706043

Name: A F M Mahfuzul Kabir
ID: 1706045

Name: Shuvro Chowdhury
ID: 1706044

Table of Contents:

(1) Introduction:

Vehicle speed detection has always been a necessity in the fields of traffic and road accident control. As the number of people using vehicles such as car, motor-bikes etc. is increasing almost exponentially, number of road accidents is also increasing. In such case, limiting vehicle speed in roads can prevent potential accident. Among the various methods used in todays technology, there are some intrusive and some non-intrusive systems [1]. Intrusive sensors are usually based on inductive loop detectors, which can be very complex to install and use as well as very high maintenance. On the other hand, non-intrusive sensors mostly include laser meters and Doppler radars. These non-intrusive systems are comparatively easier to use. However, the easy to use advantage comes with a trade-off of being very costly as well as requirement of frequent maintenance. While these systems fail to be low cost, low maintenance, and easy to use at the same time, engineers across the world has been working on speed detection from a live video feed using various machine learning and deep learning models. As IP cameras are usually low in cost and you only need a central computer to analyze the video feed and measure speed, this software-based detection system can be very much efficient, easy to use and low cost at the same time.

In our project, we've tried to build such a model using YOLO V2 network. YOLO V2 is basically a real time object detection system [2]. It processes images and detects various objects from that image. Once trained, the YOLO model can achieve high accuracy for object detection, which in our case is vehicles.

In this project, we tried to achieve the following objectives.

1. To build a model and train it using a dataset that consists of almost 2400 images of cars in roads.
2. To detect our desired object, a car, from a video feed collected from a traffic signal CCTV footage. And then, to calculate the accuracy of the vehicle detection.
3. Measure speed from the movement of our detected object in the video feed as frames change.
4. Compare our measured speed with actual speed of the vehicles in the traffic signal and calculate the error.
5. Analyze the error and search for room for improvements in future work.

(2) Methodology:

Our workflow for this project can be divided in two major parts. Detection of the vehicle and calculating speed or speed measurement. For the detection part, YOLO V2 vehicle detection network, a pre-trained model for vehicle detection was trained using a dataset of almost 2400 labeled images of cars. The video feed was loaded in the matlab program and frame by frame cars were detected. As for the speed measurement, it was done by calculating movement of the detected cars. The whole process can be represented by the following block diagram.

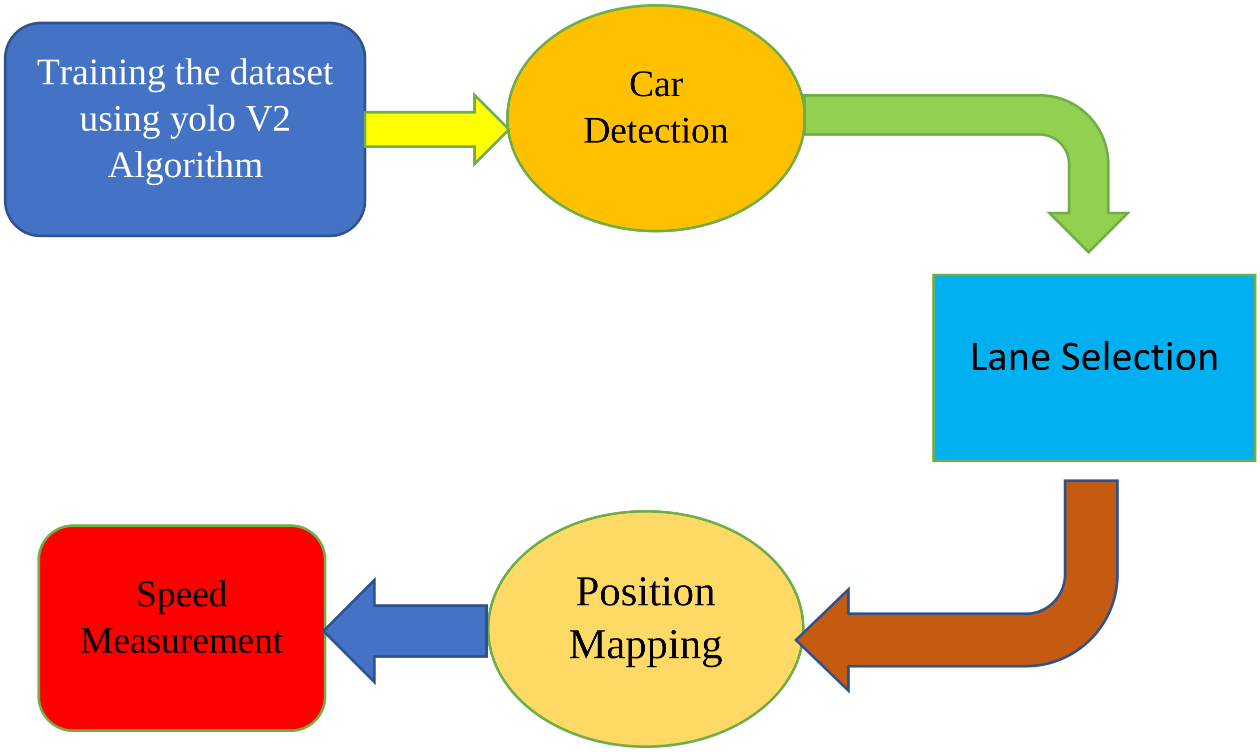


Figure: The block diagram for vehicle speed detection system.

(2.1) Training the Dataset using YOLO-V2 Vehicle Detection Network:

For our project we used YOLO-V2 (you only look once) network for object detection. We collected our dataset from a source and trained the network with almost 2400 dataset of positive images (where the car is present). We used "**Image labeler**" app from matlab to label our dataset images. The image labeler app let us define the bounding box of each cars in their respective image. The YOLO network then took those images as input, processed them to train the network for future detection. The trained output "**yolodetector3.mat**" was saved and used for further detection of cars.

Train the YOLO v2 network.

```
[detector,info] = trainYOLOv2ObjectDetector(ds,lgraph,options);
```

We also verified our training accuracy by inspecting the training loss for each iteration:

```
figure
plot(info.TrainingLoss)
grid on
xlabel('Number of Iterations')
ylabel('Training Loss for Each Iteration')
```

(2.2) Car Detection:

Once done with the training, the detector file was used to detect cars in each frame of the video feed. The detector processed each image, detected the cars and confined them in a bounding box. The bounding box has 4 values in output. The x and y coordinates of the upper left corner of the bounding box, the height and width of the bounding box. Thus, the car and its position of the respective bounding box is determined.

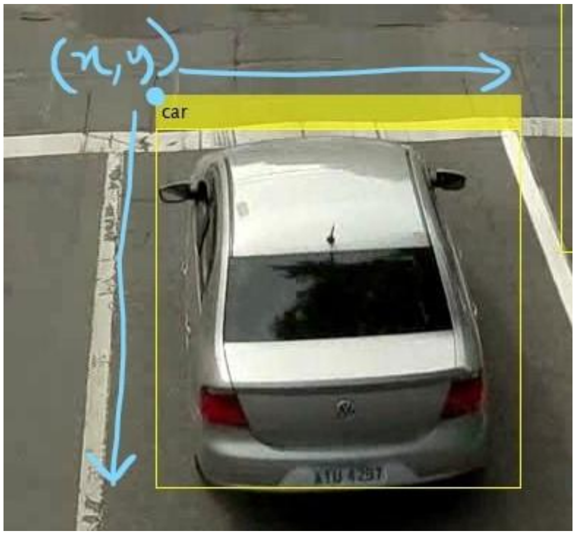


Figure: Bounding box.

```
[bboxes,scores] = detect(detector,img);
```

Display the detection results.

```
if(~isempty(bboxes))
    img = insertObjectAnnotation(img,'rectangle',bboxes,scores);
end
figure
imshow(img)
```

As for now, our system is trained only by images of cars. So other vehicles such as motor-bikes or trucks or any type of vehicle except cars are not detectable. However, detection of cars were almost 90% accurate.

(2.3) Lane Selection:

The road in the video feed had three lanes. Our speed measurement system is built to work on a single lane at a time. The roads were divided into three lanes as following. Values of x represent the pixel values. The lane 4 was avoided as the pixel width is too low in that lane and impossible to measure using our system.

1. Lane 1 (x<300) 2. Lane 2 (330<x<580) 3. Lane 3 (630<x<830)

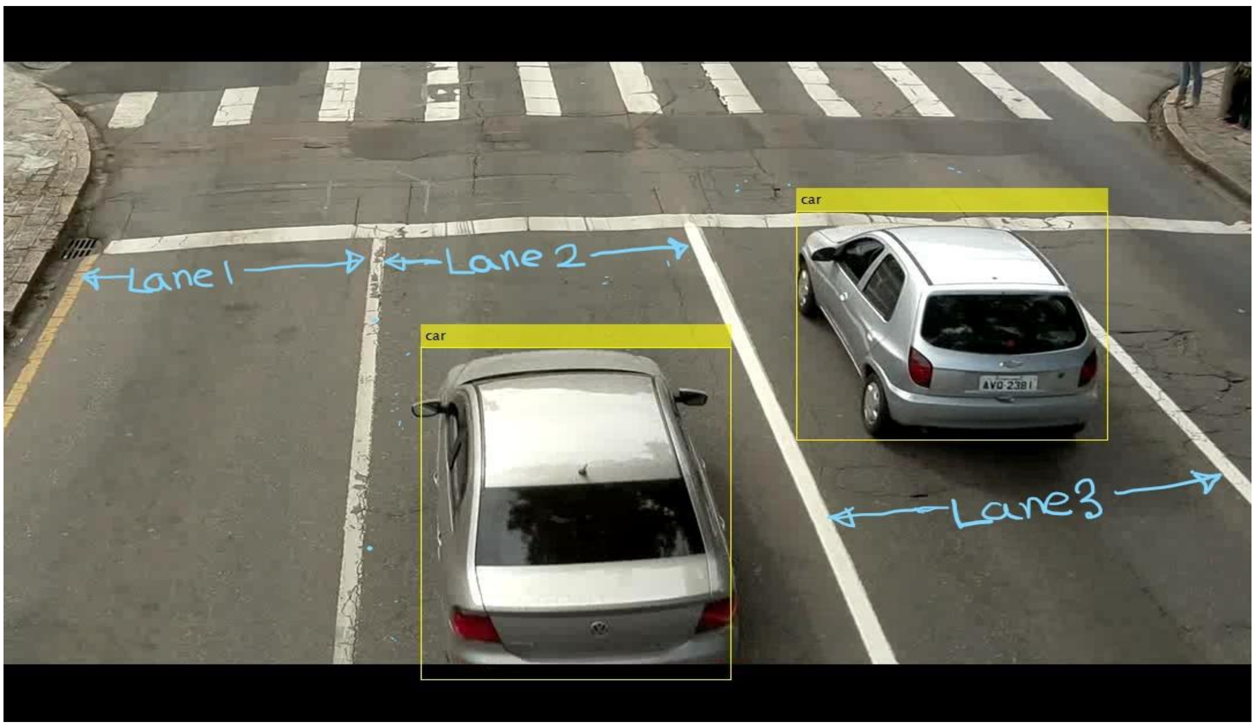


Figure: The road divided into three lanes.

The selected lanes had a specific position in their respective frame where the speed measurement would occur. The reason behind this is the fact that there are some cars that stop at the traffic signal in the video when the traffic light is red. For perfect measurement of the speed, the following range in the video frames were used. Please keep note that the cars were detected in the whole frame, but only the cars in the range shown in the figure were displayed.

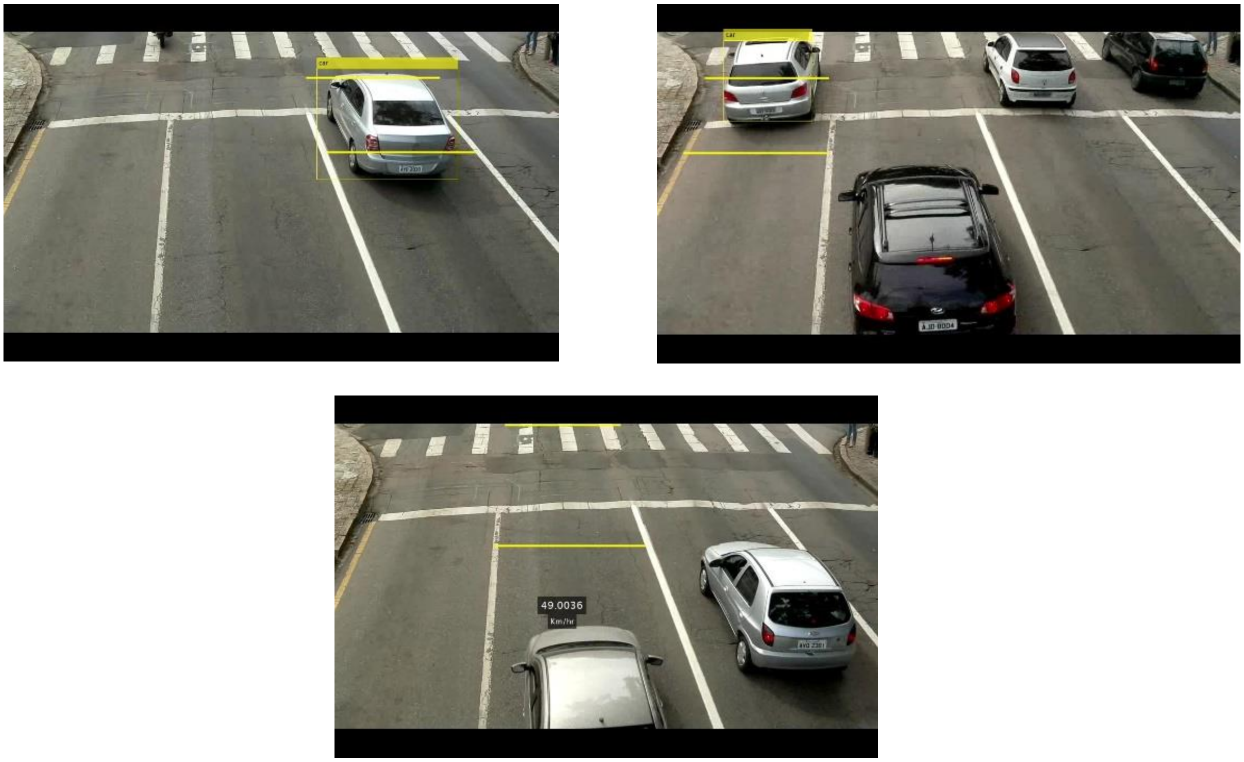


Figure: Lane 1, 2, 3 and their respective zone for vehicle speed measurement.

(2.4) Position Mapping:

For measuring the speed of each car, the procedure was to track the change in position of bounding box of each car as frames go by. The change in pixel was then multiplied by a factor to get km/h unit from change in pixels. As for position mapping, we carefully kept tract of the movement of a pixel in bounding box. In this case, we tracked the point of exactly the middle of the bounding box, as shown in the figure below.

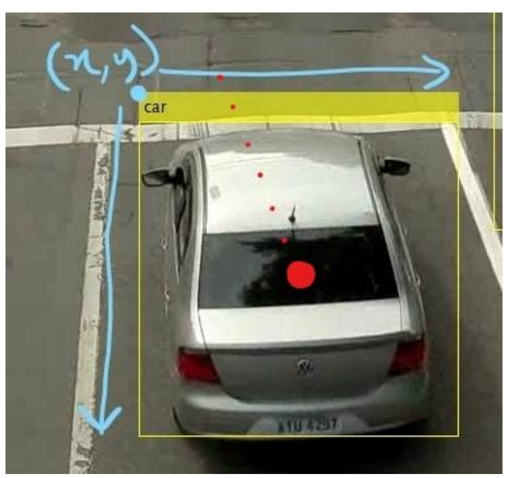


Figure: The position mapping.

As shown in the figure, the middle point of the bounding box changes its position as the vehicle goes straight. We tracked the change in its coordinates to calculate the distance covered by the car.

(3) Result:

As mentioned in the previous sections, our first task was to detect the cars in the given video feed. The feed was collected from reference [3], which was actually used in the research paper of reference [4]. Thus, we got both the traffic video feed as well as the actual speed of vehicles measured by the method mentioned in reference [4]. To detect the vehicles in the video feed, the pre-trained method YOLO V2 Vehicle Detector was used. The pre-trained model was trained using our dataset which contained almost 2400 images of cars and empty roads. The result of the detector was pretty good. However, at this moment, our system is trained only to detect cars. Vehicles of other sort such as motor-bikes, trucks etc. are not included.

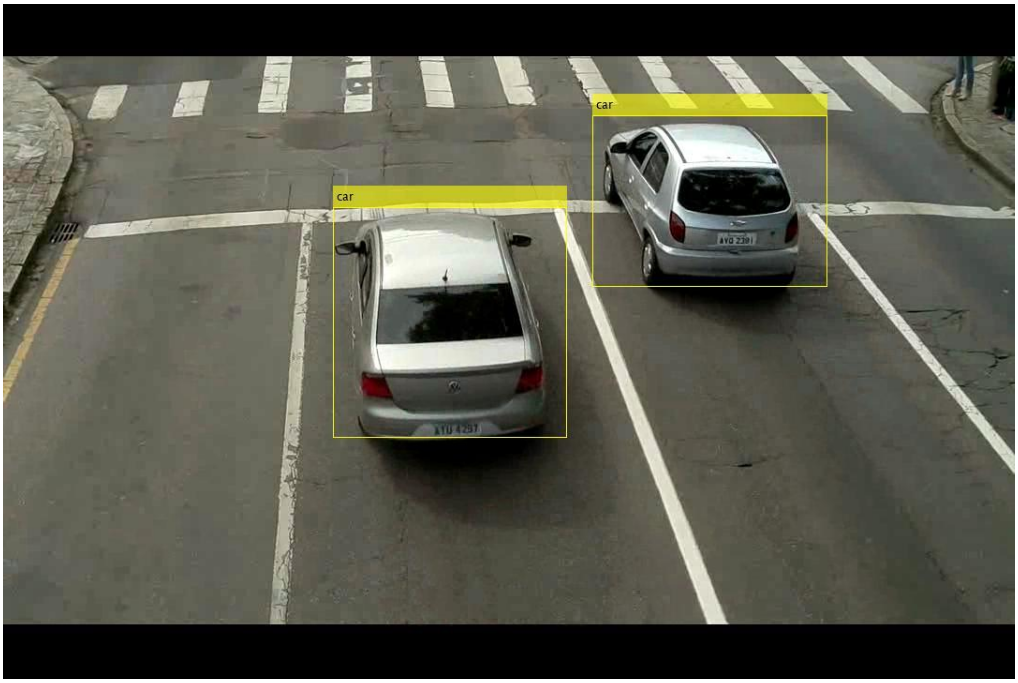


Figure: Detection of cars in the video feed.

As for the second and most vital part of the system, the detected bounding box movement was measured and thus the speed of the vehicles. As mention earlier, the speed was measured for each car in their respective lane. The speed was measured and showed in the video using matlab inserttext() function. The measured speeds were stored in an excel file which were compared to the actual values later on.

To compare the result with actual measured values, the difference between actual and calculated values were measured. This difference is the deviation of our calculated values. The RMS deviation was calculated later on to gather insight about the deviation. A percentage error was also calculated and average percentage error was calculated using the error for each case.

(3.1) Lane 1:

	Calculated			Actual						
frame start	frame end	speed	iframe	frame_start	frame_end	speed	difference	Difference squared		Error (%)
287	294	61.00709923	281	281	321	59.57	-1.437099234	2.065254208		2.412454648
443	451	49.11879878	436	436	476	51.92	2.801201221	7.846728281		5.395225773
764	771	56.34029081	757	757	797	50.59	-5.750290814	33.06584445		11.36645743
812	820	53.84686533	806	806	846	50.06	-3.78686533	14.34034903		7.564653076
2184	2206	19.21203707	2162	2162	2202	23.19	3.977962926	15.82418904		17.15378579
2282	2307	19.15036437	2271	2271	2311	21.02	1.869635631	3.495537393		8.894555809
2352	2372	22.77016155	2337	2337	2377	23.36	0.5898384477	0.3479093944		2.524993355
2413	2431	26.11666792	2404	2404	2444	29.68	3.563332079	12.69733551		12.00583585
2476	2494	24.41599851	2463	2463	2503	26.54	2.124001488	4.51138232		8.003019924
2550	2569	24.61326057	2539	2539	2579	25.78	1.16673943	1.361280899		4.525754191
2608	2622	30.81585244	2597	2597	2637	29.12	-1.695852435	2.875915482		5.823669077
2806	2814	48.87538031	2798	2798	2838	46.94	-1.935380313	3.745696956		4.123093977
2914	2924	41.10167911	2906	2906	2946	40.72	-0.3816791106	0.1456789435		0.9373259101
3486	3495	47.59816325	3479	3479	3519	49.07	1.471836754	2.166303431		2.99946353
4602	4630	15.82873143	4594	4594	4634	21.59	5.761268567	33.1922155		26.68489378
4679	4698	22.39182763	4667	4667	4707	26.63	4.238172373	17.96210506		15.91502956
4809	4818	44.23799106	4802	4802	4842	44.74	0.5020089445	0.2520129804		1.122058437
4897	5189	1.462139918	4894	4894	4934	48.18	46.71786008	2182.558451		96.96525546
5308	5316	50.06953957	5302	5302	5342	53.02	2.950460433	8.705216766		5.56480655
5368	5374	54.75548788	5361	5361	5401	52.97	-1.785487878	3.187966961		3.370753025
5527	5535	54.73714214	5519	5519	5559	49.24	-5.497142135	30.21857165		11.16397672
5653	5661	54.87131724	5647	5647	5687	52.97	-1.901317238	3.615007241		3.589422764
5717	5725	53.06343662	5711	5711	5751	49.97	-3.093436619	9.569350114		6.19058759
5768	5776	56.53750117	5760	5760	5800	48.42	-8.117501166	65.89382518		16.76476903
							RMS Avg	2.066449975	Avg	11.71091005

Here, the RMS average of deviation is **2.066 km/h**, meaning the values differ from actual values by an average of 2.066 km/h. The average error was **11.71%.**

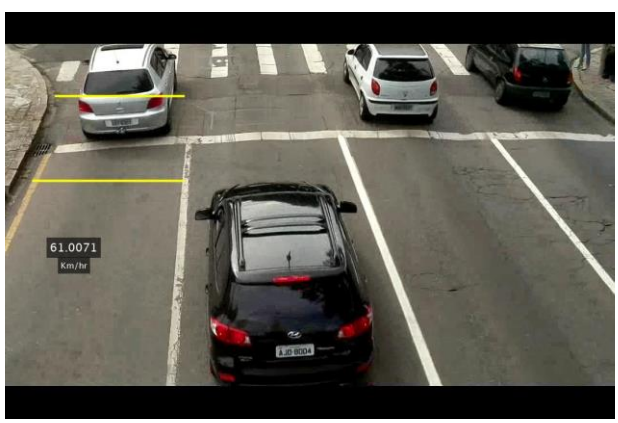


Figure: Speed measurement algorithm used in lane 1 and showed in the video.

(3.2) Lane 2:

Actual			Calculated						
start	end	speed	frame start	frame end	speed	difference	Squared Difference		Error %
187	227	52.13	195	209	49.00362755	3.126372454	9.774204724		5.997261566
253	293	50.51	261	277	45.04406178	5.465938218	29.8764806		10.82149716
295	335	48.83	304	316	48.07959394	0.7504060571	0.5631092505		1.536772593
384	424	49.16	392	399	68.03542447	-18.87542447	356.281649		38.39590007
478	518	58.33	485	498	56.40460965	1.925390346	3.707127985		3.300857785
575	615	53.55	582	596	50.96875782	2.581242181	6.662811196		4.820246836
869	909	50.84	877	890	51.26310623	-0.4231062335	0.1790188849		0.8322309865
2167	2207	19.33	2196	2233	17.80955322	1.520446779	2.311758407		7.865736051
2265	2305	24.87	2281	2303	29.41131568	-4.541315684	20.62354814		18.26021586
2319	2359	30.53	2333	2356	29.64845688	0.8815431231	0.777118278		2.887465192
2505	2545	57.07	2512	2523	56.54641839	0.5235816109	0.2741377033		0.9174375519
2569	2609	50.59	2576	2590	45.27740077	5.312599234	28.22371062		10.50128332
2612	2652	51.02	2621	2634	54.71633339	-3.696333395	13.66288056		7.244871413
2665	2705	48.42	2672	2689	43.29550706	5.12449294	26.26042789		10.58342202
2727	2767	51.11	2734	2747	51.52517849	-0.4151784865	0.1723731757		0.8123233937
2760	2800	44.33	2769	2781	46.95540168	-2.625401679	6.892733978		5.922403969
2860	2900	50.03	2868	2875	59.06848959	-9.038489595	81.69429415		18.06613951
3034	3074	42.23	3041	3058	40.61275031	1.617249694	2.615496574		3.829622766
3101	3141	52.97	3109	3121	52.87684481	0.09315519404	0.008677890176		0.1758640627
3225	3265	50.06	3234	3249	47.14411299	2.915887007	8.502397039		5.824784273
3500	3540	54.43	3507	3517	62.79398315	-8.363983147	69.95621409		15.36649485
4640	4680	24.78	4656	4687	22.83771672	1.942283277	3.77246433		7.838108464
4714	4754	30.1	4727	4744	34.36829566	-4.268295659	18.21834784		14.18038425
4765	4805	34.74	4776	4797	34.2498188	0.4901811992	0.2402776081		1.410999422
4827	4867	38.09	4838	4857	36.8600359	1.2299641	1.512811687		3.229099764
4873	4913	40.95	4882	4900	38.11528016	2.834719839	8.035636563		6.922392768
4913	4953	42.39	4920	4937	41.48253439	0.9074656116	0.8234938363		2.140753979
4990	5030	40.4	5000	5017	40.86106891	-0.4610689136	0.2125845431		1.141259687
5080	5120	45.87	5090	5106	41.81120532	4.058794675	16.47381421		8.84847324
5110	5150	37.65	5121	5139	39.25711482	-1.607114825	2.58281806		4.268565271
5162	5202	48.45	5170	5184	51.00761372	-2.55761372	6.541387941		5.278872487
5223	5263	49.76	5231	5245	47.53719808	2.222801922	4.940848387		4.467045664
5271	5311	50.71	5279	5293	45.84908868	4.860911323	23.62845889		9.585705626
5375	5415	51.07	5382	5396	48.31280439	2.75719561	7.602127634		5.398855709
5430	5470	54.07	5436	5443	61.95161545	-7.881615449	62.11986208		14.57668846
5615	5655	48.9	5624	5637	46.0408646	2.859135403	8.174655252		5.846902665
5528	5568	55.71	5534	5602	9.794965655	45.91503434	2108.190379		82.41793995
5738	5778	54.33	5624	5637	49.87760331	4.452396687	19.82383625		8.195097895
						RMS Avg	1.432196552	Avg	9.466052014

Here, the RMS average of deviation is **1.432 km/h**, meaning the values differ from actual values by an average of 1.432 km/h in lane 2. The average error was **9.466%.**

Figure: Speed measurement algorithm used in lane 2 and showed in the video.

(3.3) Lane 3:

frame end	frame start	speed					actual			Difference	Squared Diff		Error %
85	78	61.24353445				111	71	56.65		-4.593534453	21.10055877		8.108622159
122	114	58.85086868				147	107	53.74		-5.110868684	26.12097871		9.510362271
262	253	50.81447986				285	245	49.06		-1.754479856	3.078199566		3.576192124
292	283	53.44847737				315	275	48.89		-4.558477368	20.77971591		9.323946344
329	321	59.61879326				354	314	56.01		-3.608793264	13.02338882		6.443123127
389	381	52.30314161				413	373	51.12		-1.183141612	1.399824073		2.314439772
722	716	51.89875457				748	708	52.86		0.9612454328	0.9239927821		1.818474145
763	756	57.64285624				790	750	54.23		-3.41285624	11.64758771		6.293299354
883	875	56.6502202				907	867	52.39		-4.260220202	18.14947617		8.131743085
950	943	59.08365934				976	936	54.76		-4.323659336	18.69403005		7.895652549
2193	2147	10.69807037				2197	2157	16.69		5.991929635	35.90322075		35.90131597
2258	2238	22.29956072				2262	2222	22.13		-0.1695607168	0.02875083669		0.766202968
2318	2301	29.08158				2329	2289	27.55		-1.53158	2.345737298		5.559274049
2359	2345	34.38911764				2372	2332	31.31		-3.079117638	9.480965428		9.834294596
2411	2400	36.6190013				2429	2389	35.56		-1.059001304	1.121483763		2.97806891
2595	2587	55.17365905				2619	2579	48.86		-6.313659051	39.86229061		12.92193829
2639	2631	49.34681771				2665	2625	50.03		0.6831822873	0.4667380377		1.365545248
2818	2810	48.55566971				2842	2802	51.53		2.974330288	8.846640659		5.772036265
2894	2886	51.20549337				2918	2878	48.87		-2.335493368	5.45452927		4.778991953
2935	2926	51.91082922				2959	2919	46.56		-5.350829217	28.63137331		11.49233079
2971	2961	46.90285227				2993	2953	43.53		-3.37285227	11.37613243		7.748339696
3191	3187	51.77093385				3219	3179	56.6		4.829066149	23.31987987		8.531918991
3237	3231	54.88856987				3263	3223	53.87		-1.018569867	1.037484574		1.890792402
3271	3264	59.64283532				3297	3257	51.47		-8.172835321	66.79523718		15.87883295
3353	3346	60.83276041				3379	3339	55.25		-5.582760415	31.16721385		10.10454374
3398	3391	61.99279828				3425	3385	56.96		-5.032798281	25.32905854		8.83567114
3483	3476	61.51495723				3509	3469	56.86		-4.654957226	21.66862678		8.186699307
3523	3516	63.79958729				3548	3508	56.86		-6.939587292	48.15787178		12.20469098
4690	4566	3.348969552				4686	4646	19.82		16.47103045	271.294844		83.10307996
4855	4840	30.60938157				4869	4829	31.19		0.5806184257	0.3371177562		1.861553144
4930	4918	36.90495392				4946	4906	34.92		-1.984953917	3.940042051		5.684289566
4984	4972	33.92021618				5003	4963	35.29		1.369783819	1.876307711		3.881506997
5047	5036	40.80671585				5066	5026	36.7		-4.106715853	16.8651151		11.18996145
5159	5150	52.96214367				5182	5142	42.86		-10.10214367	102.0533067		23.57009722
5192	5183	50.98006467				5214	5174	46.37		-4.610064674	21.2526963		9.941912173
5393	5385	51.77616907				5384	5344	47.7		-4.076169071	16.6151543		8.545427822
5547	5539	52.49514069				5572	5532	50.88		-1.615140689	2.608679444		3.174411731
5770	5591	2.280134817				5794	5754	49.79		47.50986518	2257.18729		95.42049645
5927	5920	56.4247886				5954	5914	53.69		-2.734788595	7.47906866		5.093664733
										RMS Avg	1.449890746	**Avg**	12.29830114

Here, the RMS average of deviation is **1.449 km/h**, meaning the values differ from actual values by an average of 1.449 km/h in lane 3. The average error was **12.298%.**

Figure: S peed measurement algorithm used in lane 3 and showed in the video.

(4) Analysis:

From the results, it's visible that in all of the three lanes, the error is almost about 11%. From our analysis, we can say the error is not groundbreaking or of that sort, but it's a start. If we look at the following figures of the calculated speed vs actual speed, we can see that an idea or 100% accurate system would've generated a straight line. However, as our program isn't flawless, the plot deviates from the ideal line. There are tons of room for improvement in our model which can result in better error percentage as well as less deviation and a better plot.

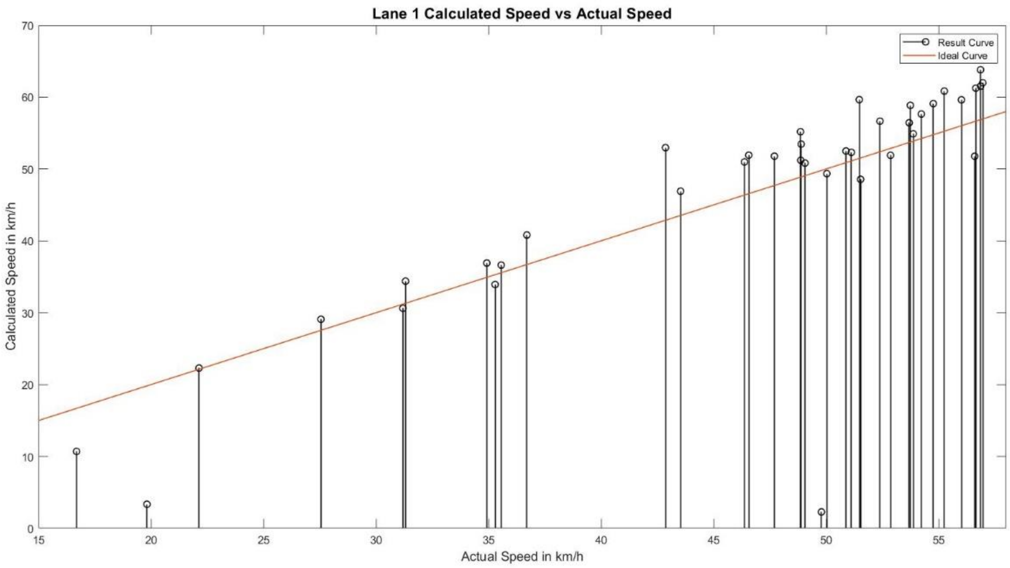


Figure: Calculated Speed vs Actual Speed Curve for Lane 1.

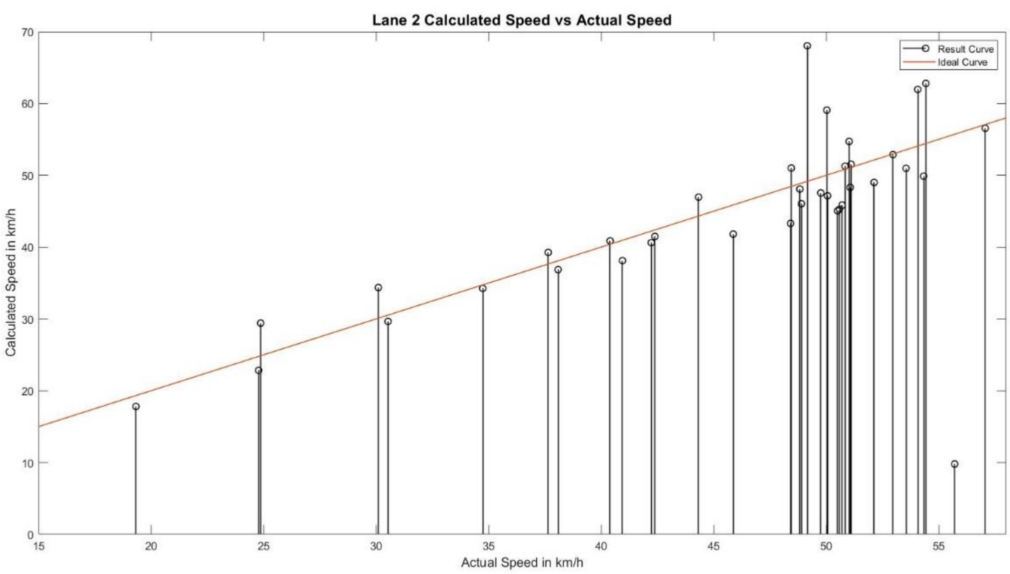


Figure: Calculated Speed vs Actual Speed Curve for Lane 2.

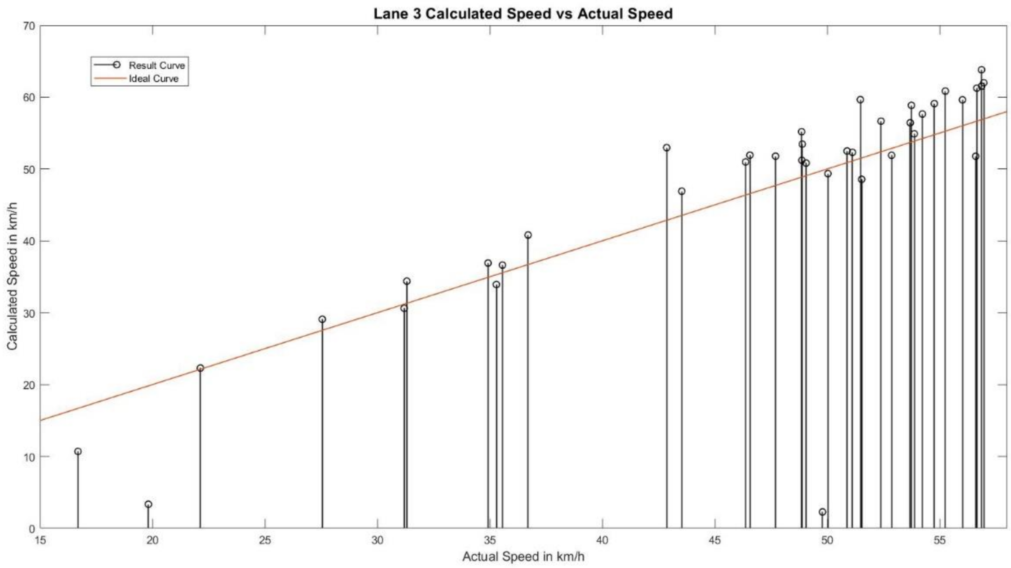


Figure: Calculated Speed vs Actual Speed Curve for Lane 3.

There are also some vehicles that went missing during our speed calculation but was present with their respective speed in the actual vehicle speed data. The reason for this missing data is simple, our method only works for cars, other vehicles such as motor-bikes or trucks are not yet detectable by our system. But the video feed contains such vehicles. Thus, the missing values occur.

(5) Room for Improvements:

As we submit our project for this time, we plan to improve our program with time. These are the objectives we'll be working on as we proceed for further improvements.

1. Decrease the error in measurement, drop it down at less than 5%.
2. Train our program for more types of vehicles.
3. Learn more insight about ML and DL along the way and try to build our own model for vehicle detection instead of the pre-trained model of YOLO V2.
4. Modify the program so that lane 4 can be included in speed measurement as well.

(6) Discussion:

As can be seen from our analysis, our system can efficiently measure speed from the given video file. However, the system is not flawless and there are some room for improvements. In future, we target to address those improvements and proceed. Right now, our system is at an initial stage which can be applied only for video feed similar to the one we used in our program. In future, we plan to build a more generalized system that will detect the grids and lanes by itself and measure speed more accurately. We plan to improve our methodology as well to improve our overall system result.

(7) Appendix:

Matlab codes:

File name: yolo.mlx **Objective:** Train the YOLO V2 vehicle detection network

Train YOLO v2 Network for Vehicle Detection

Load the training data for vehicle detection

```
data = load('yoloTrainingData3.mat');
trainingData = data.gTruth;
```

Randomly shuffle data

```
rng(0);
shuffledIdx = randperm(height(trainingData));
trainingData = trainingData(shuffledIdx,:);
```

```
imds = imageDatastore(trainingData.imageFilename);
```

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

```
ds = combine(imds, blds);
```

Load a YOLO v2 object detection network.

```
net = load('yolov2VehicleDetector.mat');
lgraph = net.lgraph
```

```
lgraph = 
  LayerGraph with properties:

         Layers: [25×1 nnet.cnn.layer.Layer]
    Connections: [24×2 table]
     InputNames: {'input'}
    OutputNames: {'yolov2OutputLayer'}
```

```
lgraph.Layers;
```

```
options = trainingOptions('sgdm',...
          'InitialLearnRate',0.001,...
          'Verbose',true,...
          'MiniBatchSize',16,...
          'MaxEpochs',50,...
          'Shuffle','never',...
```

```
            'VerboseFrequency',200,...
            'CheckpointPath',tempdir);
```

Train the network

```
[detector,info] = trainYOLOv2ObjectDetector(ds,lgraph,options);
```

```
*************************************************************************
Training a YOLO v2 Object Detector for the following object classes:

* car

Training on single CPU.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     RMSE     |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:00 |         7.16 |         51.2 |          0.0010 |
|       3 |         200 |       00:02:20 |         1.10 |          1.2 |          0.0010 |
|       5 |         400 |       00:04:33 |         0.73 |          0.5 |          0.0010 |
|       7 |         600 |       00:06:40 |         0.66 |          0.4 |          0.0010 |
|       9 |         800 |       00:08:55 |         0.51 |          0.3 |          0.0010 |
|      11 |        1000 |       00:11:04 |         0.54 |          0.3 |          0.0010 |
|      14 |        1200 |       00:13:12 |         0.47 |          0.2 |          0.0010 |
|      16 |        1400 |       00:15:34 |         0.45 |          0.2 |          0.0010 |
|      18 |        1600 |       00:17:36 |         0.37 |          0.1 |          0.0010 |
|      20 |        1800 |       00:19:35 |         0.33 |          0.1 |          0.0010 |
|      22 |        2000 |       00:21:32 |         0.28 |      8.0e-02 |          0.0010 |
|      25 |        2200 |       00:23:30 |         0.24 |      5.9e-02 |          0.0010 |
|      27 |        2400 |       00:25:29 |         0.27 |      7.1e-02 |          0.0010 |
|      29 |        2600 |       00:27:38 |         0.23 |      5.3e-02 |          0.0010 |
|      31 |        2800 |       00:29:27 |         0.20 |      4.1e-02 |          0.0010 |
|      33 |        3000 |       00:31:12 |         0.20 |      3.9e-02 |          0.0010 |
|      36 |        3200 |       00:32:58 |         0.18 |      3.2e-02 |          0.0010 |
|      38 |        3400 |       00:34:39 |         0.13 |      1.7e-02 |          0.0010 |
|      40 |        3600 |       00:36:16 |         0.15 |      2.4e-02 |          0.0010 |
|      42 |        3800 |       00:37:52 |         0.28 |      7.9e-02 |          0.0010 |
|      44 |        4000 |       00:39:28 |         0.23 |      5.4e-02 |          0.0010 |
|      47 |        4200 |       00:41:05 |         0.21 |      4.4e-02 |          0.0010 |
|      49 |        4400 |       00:42:41 |         0.16 |      2.5e-02 |          0.0010 |
|      50 |        4550 |       00:43:53 |         0.15 |      2.2e-02 |          0.0010 |
|=========================================================================================|
Detector training complete.
*************************************************************************
```

```
detector
```

```
detector = 
  yolov2ObjectDetector with properties:

            ModelName: 'car'
```

```
              Network: [1×1 DAGNetwork]
    TrainingImageSize: [128 128]
          AnchorBoxes: [4×2 double]
           ClassNames: car
```

Verification

```
figure
plot(info.TrainingLoss)
grid on
xlabel('Number of Iterations')
ylabel('Training Loss for Each Iteration')
```

Test image Read

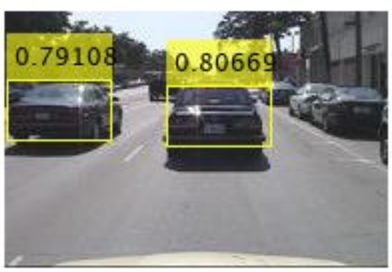
```
img = imread('detectcars.png');
```

Object detection

```
[bboxes,scores] = detect(detector,img);
```

Result

```
if(~isempty(bboxes))
    img = insertObjectAnnotation(img,'rectangle',bboxes,scores);
end
figure
imshow(img)
```



File name: lane1.mlx **Objective:** Calculate and show measured speed for Lane 1

```
clc
clear
```

```
load('yoloDetector3.mat')
```

```
video2 = vision.VideoFileReader('Set01_video01_xvid.avi');
videoPlayer2 = vision.VideoPlayer();
data=[0 0 0];
frame=0;
data_pre=[0 0 0];
```

```
while ~isDone(video2)
    frame = frame+1;
    img=step(video2);

    [bbox, score,label] = detect(detector,img);
    if(~isempty(bbox))

        pos=[bbox(:,1)+bbox(:,3)/2 , bbox(:,2)+bbox(:,4)/2];
        num=height(pos);
        for i=1:num
            if pos(i,1) < 300 && pos(i,2) < 300 && pos(i,2) > 150
                bbox2=bbox(i,:);
                label2=label(i,:);
                data = [data ; pos(i,:),frame];
                img = insertObjectAnnotation(img,'rectangle',bbox2,label2);

            elseif pos(i,1) < 300 && pos(i,2) < 150
                if height(data)>2

                    f_data = [data(2,:) ; data(end,:)];
                    data;

                    pix = sqrt((data(2,1) - data(end,1))^2 + (data(2,2) -
data(end,2))^2);
                    t= (data(end,3) - data(2,3))/25;
                    speed = pix * (1/8.25) /t;
                    data_pre=[data_pre ; data(end,3),data(2,3),speed];

img=insertText(img,[75,400],speed,'FontSize',20,'TextColor',"white","BoxColor","b
lack");

img=insertText(img,[95,435],'Km/hr','FontSize',15,'TextColor',"white","BoxColor",
"black");
                    videoPlayer2.step(img);
                    pause(1)
                end
                data=[0 0 0];


            end
        end
```

```
    end
    img = insertShape(img,'Line',[50 300 320 300; 90 150 320 150],'LineWidth',5);


    videoPlayer2.step(img);
    pause(0.0)
end


lane1_data=table(data_pre(:,1),data_pre(:,2),data_pre(:,3));
writetable(lane1_data,'lane1_data2.xlsx')
```

File name: lane2.mlx **Objective:** Calculate and show measured speed for Lane 2

```
clc
clear
```

```
load('yoloDetector3.mat')
```

```
video2 = vision.VideoFileReader('Set01_video01_xvid.avi');
videoPlayer2 = vision.VideoPlayer();
data=[0 290 0];
frame=0;
data_pre=[0 0 0];
```

```
while ~isDone(video2)
    frame = frame+1;
    img=step(video2);

    [bbox, score,label] = detect(detector,img);
    if(~isempty(bbox))

        pos=[bbox(:,1)+bbox(:,3)/2 , bbox(:,2)+bbox(:,4)/2];
        num=height(pos);
        for i=1:num
            if pos(i,1) > 330 && pos(i,1) < 580 && pos(i,2) < 290 && abs(pos(i,2)
- data(end,2)) < 100
                bbox2=bbox(i,:);
                label2=label(i,:);
                data = [data ; pos(i,:),frame];
                img = insertObjectAnnotation(img,'rectangle',bbox2,label2);
```

```
            elseif pos(i,1) > 330 && pos(i,1) < 580 && abs(pos(i,2) -
data(end,2)) > 100
                if height(data)>2

                    f_data = [data(2,:) ; data(end,:)];
                    data;

                    pix = sqrt((data(2,1) - data(end,1))^2 + (data(2,2) -
data(end,2))^2);
                    t= (data(end,3) - data(2,3))/25;
                    speed = pix * (1/6.5) /t;
                    if speed > 10
                        data_pre=[data_pre ; data(end,3),data(2,3),speed];

img=insertText(img,[405,400],speed,'FontSize',20,'TextColor',"white","BoxColor","
black");

img=insertText(img,[425,435],'Km/hr','FontSize',15,'TextColor',"white","BoxColor"
,"black");
                        videoPlayer2.step(img);
                        pause(1)
                    end
                end
                data=[0 290 0];


            end
        end



    end
    img = insertShape(img,'Line',[320 300 620 300; 340 60 570 60],'LineWidth',5);
    videoPlayer2.step(img);
    pause(0.0)
end
lane2_data=table(data_pre(:,1),data_pre(:,2),data_pre(:,3));
writetable(lane2_data,'lane2_data2.xlsx')
```

File name: lane3.mlx **Objective:** Calculate and show measured speed for Lane 3

```
clc
clear
```

```
load('yoloDetector3.mat')
```

```
video2 = vision.VideoFileReader('Set01_video01_xvid.avi');
```

```
videoPlayer2 = vision.VideoPlayer();
data=[0 0 0];
frame=0;
data_pre=[0 0 0];
```

```
while ~isDone(video2)
    frame = frame+1;
    img=step(video2);

    [bbox, score,label] = detect(detector,img);
    if(~isempty(bbox))

        pos=[bbox(:,1)+bbox(:,3)/2 , bbox(:,2)+bbox(:,4)/2];
        num=height(pos);
        for i=1:num
            if pos(i,1) > 630 && pos(i,1) < 830 && pos(i,2) < 300 && pos(i,2) >
150
                bbox2=bbox(i,:);
                label2=label(i,:);
                data = [data ; pos(i,:),frame];
                img = insertObjectAnnotation(img,'rectangle',bbox2,label2);

            elseif pos(i,1) > 630 && pos(i,1) < 830 && pos(i,2) < 150
                if height(data)>2

                    f_data = [data(2,:) ; data(end,:)];

                    pix = sqrt((data(2,1) - data(end,1))^2 + (data(2,2) -
data(end,2))^2);
                    t= (data(end,3) - data(2,3))/25;
                    speed = pix * (1/8.75) /t;
                    data_pre=[data_pre ; data(end,3),data(2,3),speed];

img=insertText(img,[780,400],speed,'FontSize',20,'TextColor',"white","BoxColor","
black");

img=insertText(img,[800,435],'Km/hr','FontSize',15,'TextColor',"white","BoxColor"
,"black");
                    videoPlayer2.step(img);
                    pause(1)
                end
                data=[0 0 0];


            end
        end
```

```
    end
    img = insertShape(img,'Line',[630 300 920 300; 590 150 850 
150],'LineWidth',5);




    videoPlayer2.step(img);
    pause(0.0)
end


lane3_data=table(data_pre(:,1),data_pre(:,2),data_pre(:,3));
writetable(lane3_data,'lane3_data.xlsx')
```

File name: plots.m **Objective:** Plot and show the calculated speed vs actual speed curve

```
clc
clear all
close all

%% Lane 1
L1 = [61.24353445
    58.85086868
    50.81447986
    53.44847737
    59.61879326
    52.30314161
    51.89875457
    57.64285624
    56.6502202
    59.08365934
    10.69807037
    22.29956072
    29.08158
    34.38911764
    36.6190013
    55.17365905
    49.34681771
    48.55566971
    51.20549337
    51.91082922
    46.90285227
    51.77093385
    54.88856987
    59.64283532
    60.83276041
    61.99279828
    61.51495723
    63.79958729
```

```
    3.348969552
    30.60938157
    36.90495392
    33.92021618
    40.80671585
    52.96214367
    50.98006467
    51.77616907
    52.49514069
    2.280134817
    56.4247886];

L1actual=[56.65
    53.74
    49.06
    48.89
    56.01
    51.12
    52.86
    54.23
    52.39
    54.76
    16.69
    22.13
    27.55
    31.31
    35.56
    48.86
    50.03
    51.53
    48.87
    46.56
    43.53
    56.6
    53.87
    51.47
    55.25
    56.96
    56.86
    56.86
    19.82
    31.19
    34.92
    35.29
    36.7
    42.86
    46.37
    47.7
    50.88
    49.79
    53.69];
```

```
stem(L1actual, L1,'black', 'linewidth',1);
hold on
n=10:70;
plot(n,n,'linewidth',1);
xlim([15,58]);
title('Lane 1 Calculated Speed vs Actual Speed','fontsize',14);
xlabel('Actual Speed in km/h','fontsize',12);
ylabel('Calculated Speed in km/h','fontsize',12);
legend('Result Curve','Ideal Curve');
%% Lane 2
L1 = [49.00362755
45.04406178
48.07959394
68.03542447
56.40460965
50.96875782
51.26310623
17.80955322
29.41131568
29.64845688
56.54641839
45.27740077
54.71633339
43.29550706
51.52517849
46.95540168
59.06848959
40.61275031
52.87684481
47.14411299
62.79398315
22.83771672
34.36829566
34.2498188
36.8600359
38.11528016
41.48253439
40.86106891
41.81120532
39.25711482
51.00761372
47.53719808
45.84908868
48.31280439
61.95161545
46.0408646
9.794965655
49.87760331];

L1actual=[52.13
50.51
48.83
```

```
49.16
58.33
53.55
50.84
19.33
24.87
30.53
57.07
50.59
51.02
48.42
51.11
44.33
50.03
42.23
52.97
50.06
54.43
24.78
30.1
34.74
38.09
40.95
42.39
40.4
45.87
37.65
48.45
49.76
50.71
51.07
54.07
48.9
55.71
54.33];

figure
stem(L1actual, L1,'black', 'linewidth',1);
hold on
n=10:70;
plot(n,n,'linewidth',1);
xlim([15,58]);
title('Lane 2 Calculated Speed vs Actual Speed','fontsize',14);
xlabel('Actual Speed in km/h','fontsize',12);
ylabel('Calculated Speed in km/h','fontsize',12);
legend('Result Curve','Ideal Curve');
%% Lane 3
L1 = [61.24353445
58.85086868
50.81447986
53.44847737
59.61879326
```

```
52.30314161
51.89875457
57.64285624
56.6502202
59.08365934
10.69807037
22.29956072
29.08158
34.38911764
36.6190013
55.17365905
49.34681771
48.55566971
51.20549337
51.91082922
46.90285227
51.77093385
54.88856987
59.64283532
60.83276041
61.99279828
61.51495723
63.79958729
3.348969552
30.60938157
36.90495392
33.92021618
40.80671585
52.96214367
50.98006467
51.77616907
52.49514069
2.280134817
56.4247886];

L1actual=[56.65
53.74
49.06
48.89
56.01
51.12
52.86
54.23
52.39
54.76
16.69
22.13
27.55
31.31
35.56
48.86
50.03
```

```
51.53
48.87
46.56
43.53
56.6
53.87
51.47
55.25
56.96
56.86
56.86
19.82
31.19
34.92
35.29
36.7
42.86
46.37
47.7
50.88
49.79
53.69];

figure
stem(L1actual, L1,'black', 'linewidth',1);
hold on
n=10:70;
plot(n,n,'linewidth',1);
xlim([15,58]);
title('Lane 3 Calculated Speed vs Actual Speed','fontsize',14);
xlabel('Actual Speed in km/h','fontsize',12);
ylabel('Calculated Speed in km/h','fontsize',12);
legend('Result Curve','Ideal Curve');
```

(8) References:

[1] T. V. Mathew, “Intrusive and non-intrusive technologies,” Indian Institute of Technology Bombay, Tech. Rep., 2014.

[2] https://pjreddie.com/darknet/yolov2/

[3] https://pessoal.dainf.ct.utfpr.edu.br/rminetto/projects/vehicle-speed/

[4] “A Video-Based System for Vehicle Speed Measurement in Urban Roadways”, Diogo C. Luvizon, Bogdan T. Nassu and Rodrigo Minetto, 2016.