Image-Based Situation Awareness Audit 8.5.2018

Sakari Lampola

Previous Audit 28.2.2018

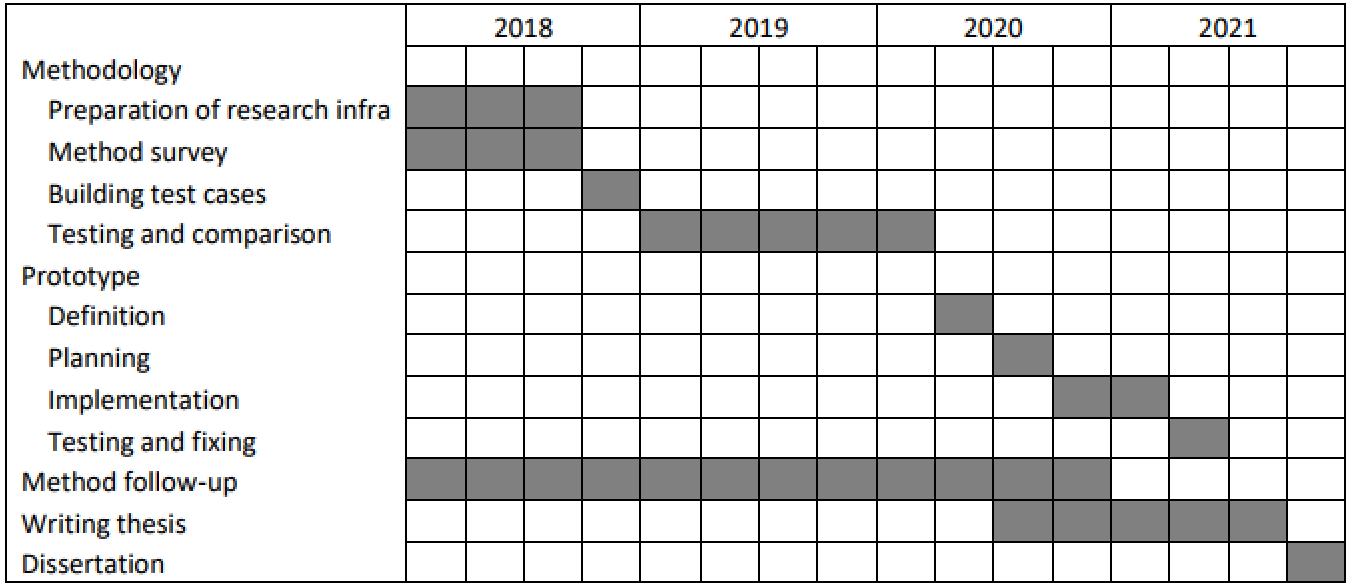
Next steps

- Kalman filter parameter adjustments (Q1)
- Dataset selection (Q1)
- Stereo vision (Q2)
- Camera yaw, pitch, roll estimation (Q2)
- Speech recognition (Q2)
- Semantic segmentation (Q2)
- Experiments in the wild (Q2)
- Paper (Q3)
- Speech analysis (Q3)
- Speech generation (Q3)
- Use cases (Q4)

Other

- Body forecast
  - kinetic
  - based on class history
  - based on swarm history
- R matrix estimation
- Monograph or papers

Project Plan



	2018				2019				2020				2021			
Methodology																
Preparation of research infra	■	■	■													
Method survey	■	■	■													
Building test cases				■												
Testing and comparison					■	■	■	■	■							
Prototype																
Definition										■						
Planning											■					
Implementation												■	■			
Testing and fixing														■		
Method follow-up	■	■	■	■	■	■	■	■	■	■	■	■				
Writing thesis											■	■	■	■	■	
Dissertation																■

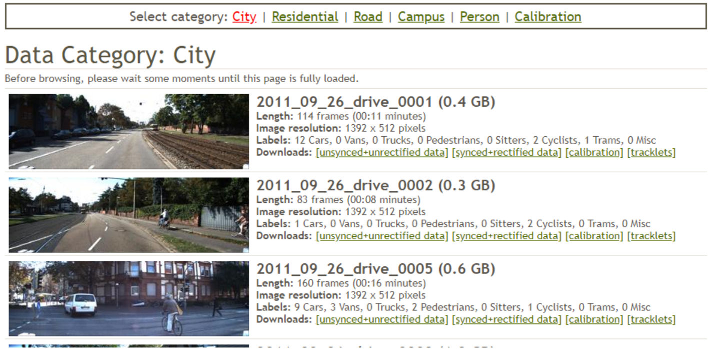
1. Methodology / Preparation of research infra
    a. Software platforms are constructed and tested
    b. Off-the-shelf models are acquired and tested
    c. Necessary skills on platforms are learned
2. Methodology / Method survey
    a. Current state-of-art methods are studied
    b. Methods are constructed and tested on the software platforms
3. Method follow-up
    a. Screening of conference papers related to the subject
    b. Possibly integrating new methods to the project

Work Done

Dataset Selection

Specification:

- Video
- Stereo
- Distance information
- Outdoor + indoor
- Odometry



Select category: City | Residential | Road | Campus | Person | Calibration

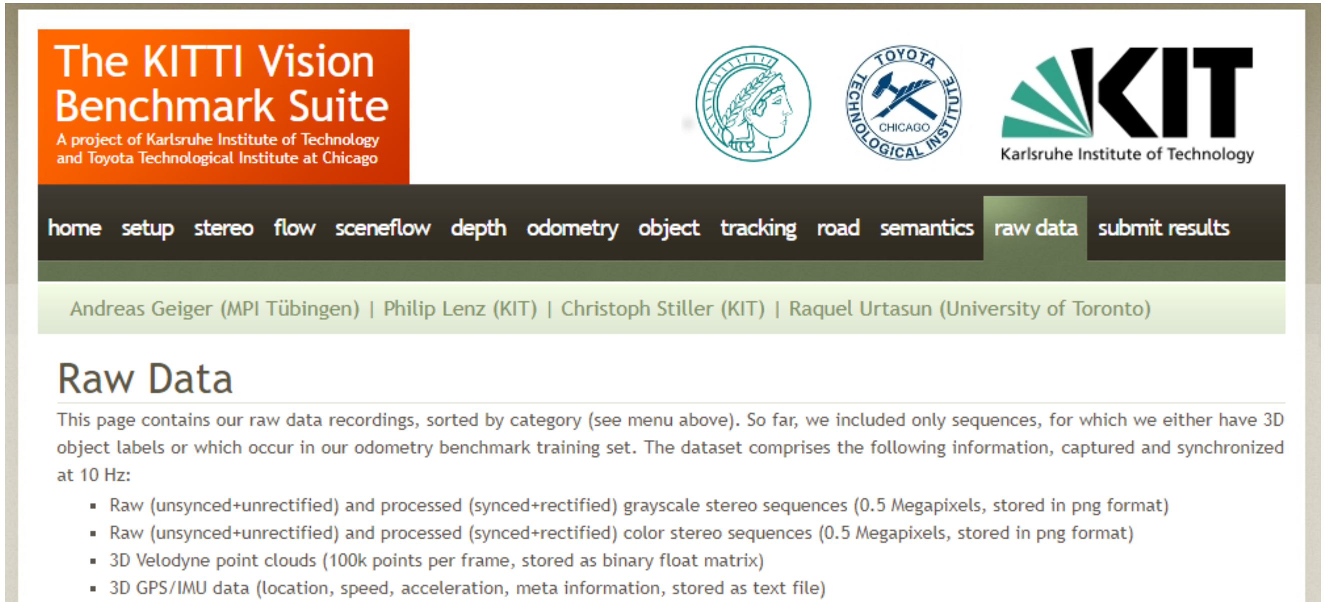
Data Category: City

Before browsing, please wait some moments until this page is fully loaded.

2011_09_26_drive_0001 (0.4 GB)
Length: 114 frames (00:11 minutes)
Image resolution: 1392 x 512 pixels
Labels: 12 Cars, 0 Vans, 0 Trucks, 0 Pedestrians, 0 Sitters, 2 Cyclists, 1 Trams, 0 Misc
Downloads: [unsynced+unrectified data] [synced+rectified data] [calibration] [tracklets]

2011_09_26_drive_0002 (0.3 GB)
Length: 83 frames (00:08 minutes)
Image resolution: 1392 x 512 pixels
Labels: 1 Cars, 0 Vans, 0 Trucks, 0 Pedestrians, 0 Sitters, 2 Cyclists, 0 Trams, 0 Misc
Downloads: [unsynced+unrectified data] [synced+rectified data] [calibration] [tracklets]

2011_09_26_drive_0005 (0.6 GB)
Length: 160 frames (00:16 minutes)
Image resolution: 1392 x 512 pixels
Labels: 9 Cars, 3 Vans, 0 Trucks, 2 Pedestrians, 0 Sitters, 1 Cyclists, 0 Trams, 0 Misc
Downloads: [unsynced+unrectified data] [synced+rectified data] [calibration] [tracklets]



The KITTI Vision Benchmark Suite

A project of Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago

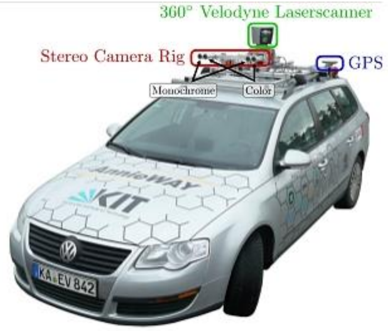
TOYOTA TECHNOLOGICAL INSTITUTE CHICAGO
KIT Karlsruhe Institute of Technology
home setup stereo flow sceneflow depth odometry object tracking road semantics raw data submit results

Andreas Geiger (MPI Tübingen) | Philip Lenz (KIT) | Christoph Stiller (KIT) | Raquel Urtasun (University of Toronto)

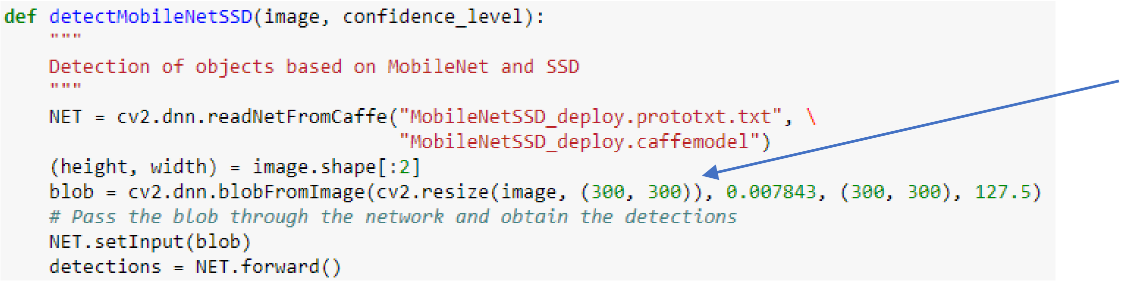
Raw Data

This page contains our raw data recordings, sorted by category (see menu above). So far, we included only sequences, for which we either have 3D object labels or which occur in our odometry benchmark training set. The dataset comprises the following information, captured and synchronized at 10 Hz:

- Raw (unsynced+unrectified) and processed (synced+rectified) grayscale stereo sequences (0.5 Megapixels, stored in png format)
- Raw (unsynced+unrectified) and processed (synced+rectified) color stereo sequences (0.5 Megapixels, stored in png format)
- 3D Velodyne point clouds (100k points per frame, stored as binary float matrix)
- 3D GPS/IMU data (location, speed, acceleration, meta information, stored as text file)
- Calibration (Camera, Camera-to-GPS/IMU, Camera-to-Velodyne, stored as text file)
- 3D object tracklet labels (cars, trucks, trams, pedestrians, cyclists, stored as xml file)

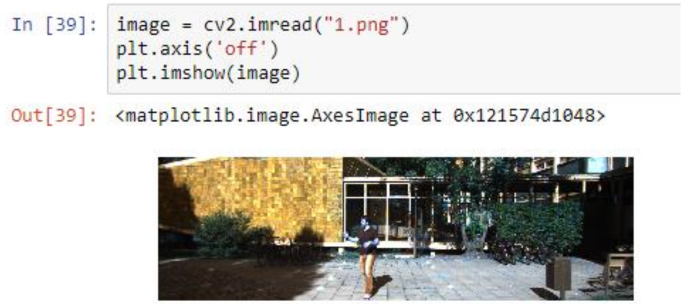


Open question: Indoor? Self generated?

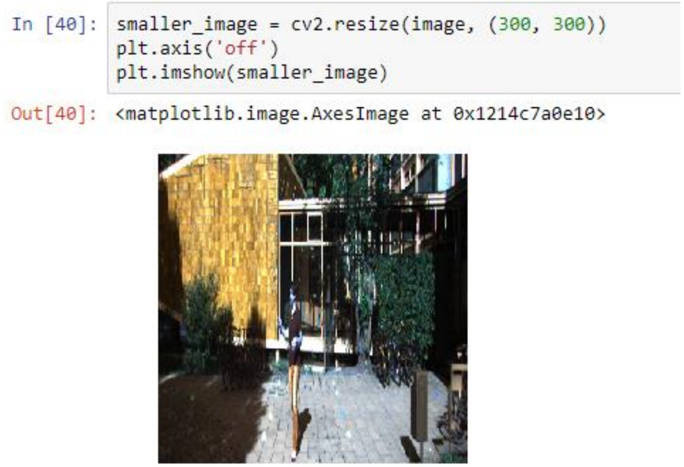


```python
def detectMobileNetSSD(image, confidence_level):
    """
    Detection of objects based on MobileNet and SSD
    """
    NET = cv2.dnn.readNetFromCaffe("MobileNetSSD_deploy.prototxt.txt", \
                                   "MobileNetSSD_deploy.caffemodel")
    (height, width) = image.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 0.007843, (300, 300), 127.5)
    # Pass the blob through the network and obtain the detections
    NET.setInput(blob)
    detections = NET.forward()
```

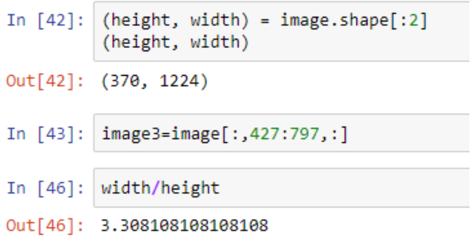
Image is resized to 300*300 pixels



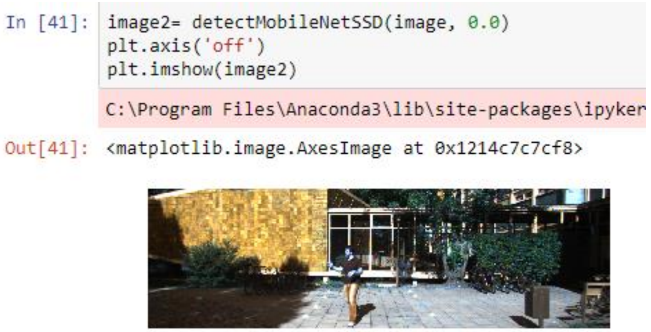
```
In [39]: image = cv2.imread("1.png")
         plt.axis('off')
         plt.imshow(image)

Out[39]: <matplotlib.image.AxesImage at 0x121574d1048>
```



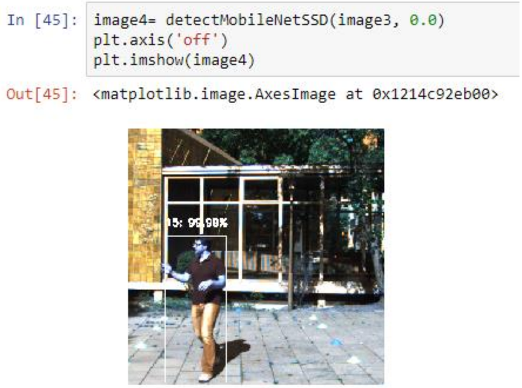
```
In [40]: smaller_image = cv2.resize(image, (300, 300))
         plt.axis('off')
         plt.imshow(smaller_image)

Out[40]: <matplotlib.image.AxesImage at 0x1214c7a0e10>
```



```
In [42]: (height, width) = image.shape[:2]
         (height, width)

Out[42]: (370, 1224)

In [43]: image3=image[:,427:797,:]

In [46]: width/height

Out[46]: 3.308108108108108
```

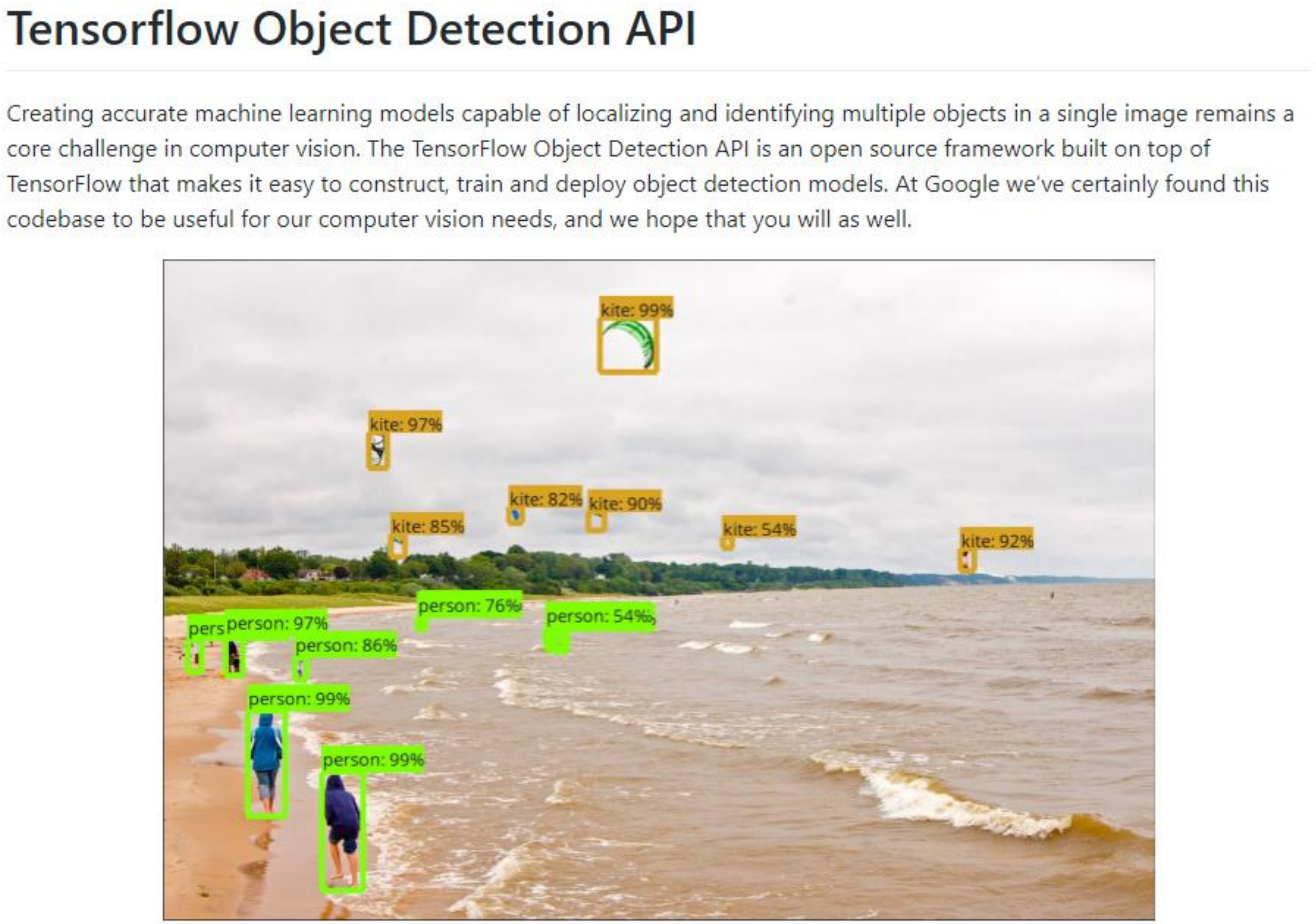


```
In [41]: image2= detectMobileNetSSD(image, 0.0)
         plt.axis('off')
         plt.imshow(image2)

         C:\Program Files\Anaconda3\lib\site-packages\ipyker

Out[41]: <matplotlib.image.AxesImage at 0x1214c7c7cf8>
```



```
In [45]: image4= detectMobileNetSSD(image3, 0.0)
         plt.axis('off')
         plt.imshow(image4)

Out[45]: <matplotlib.image.AxesImage at 0x1214c92eb00>
```

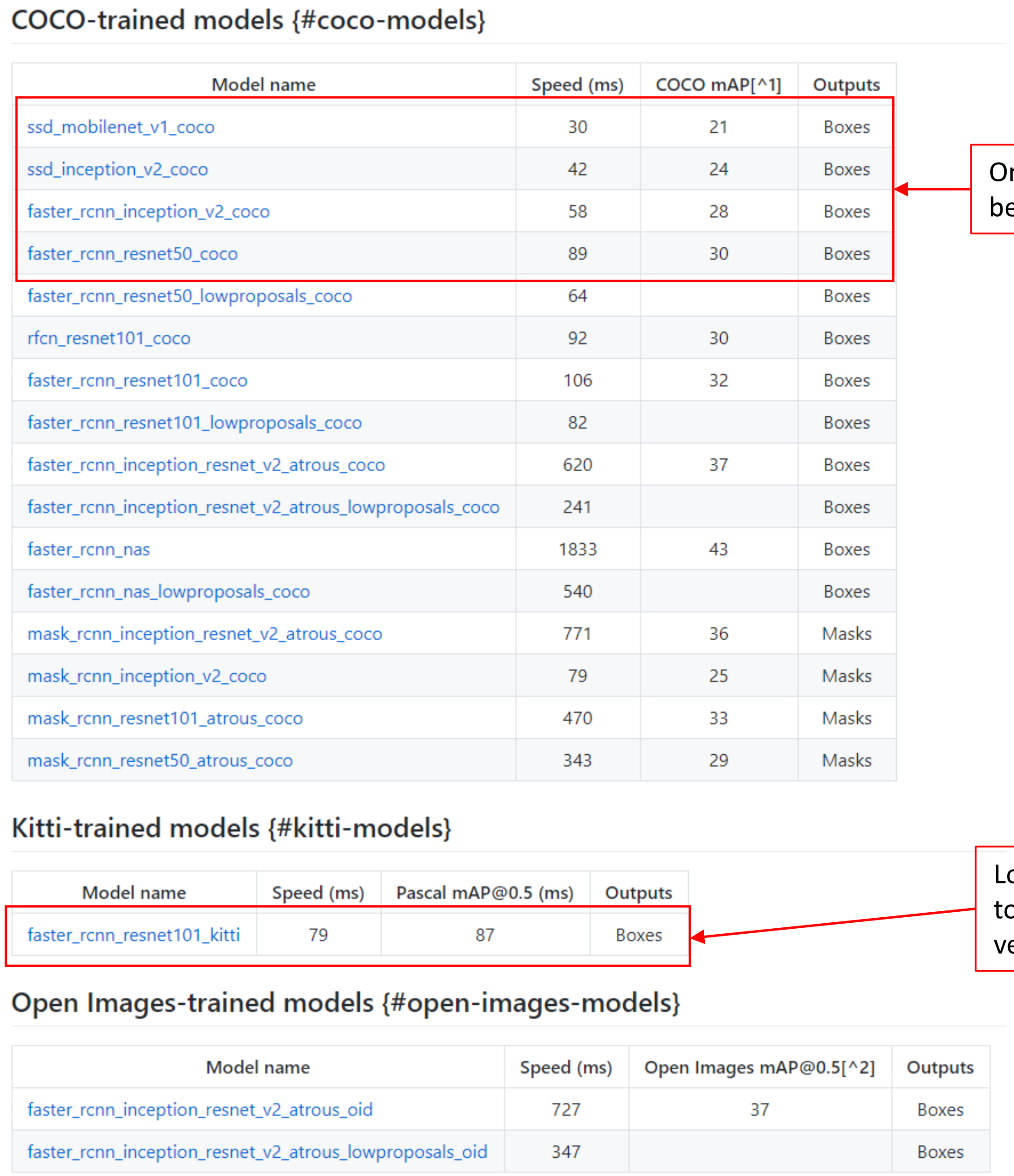
Resized KITTI image is too deformed to be useful. We need another network implementation!



Tensorflow Object Detection API

Creating accurate machine learning models capable of localizing and identifying multiple objects in a single image remains a core challenge in computer vision. The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models. At Google we've certainly found this codebase to be useful for our computer vision needs, and we hope that you will as well.

https://github.com/tensorflow/models/tree/master/research/object_detection



COCO-trained models {#coco-models}

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks

One of these will be the final model.

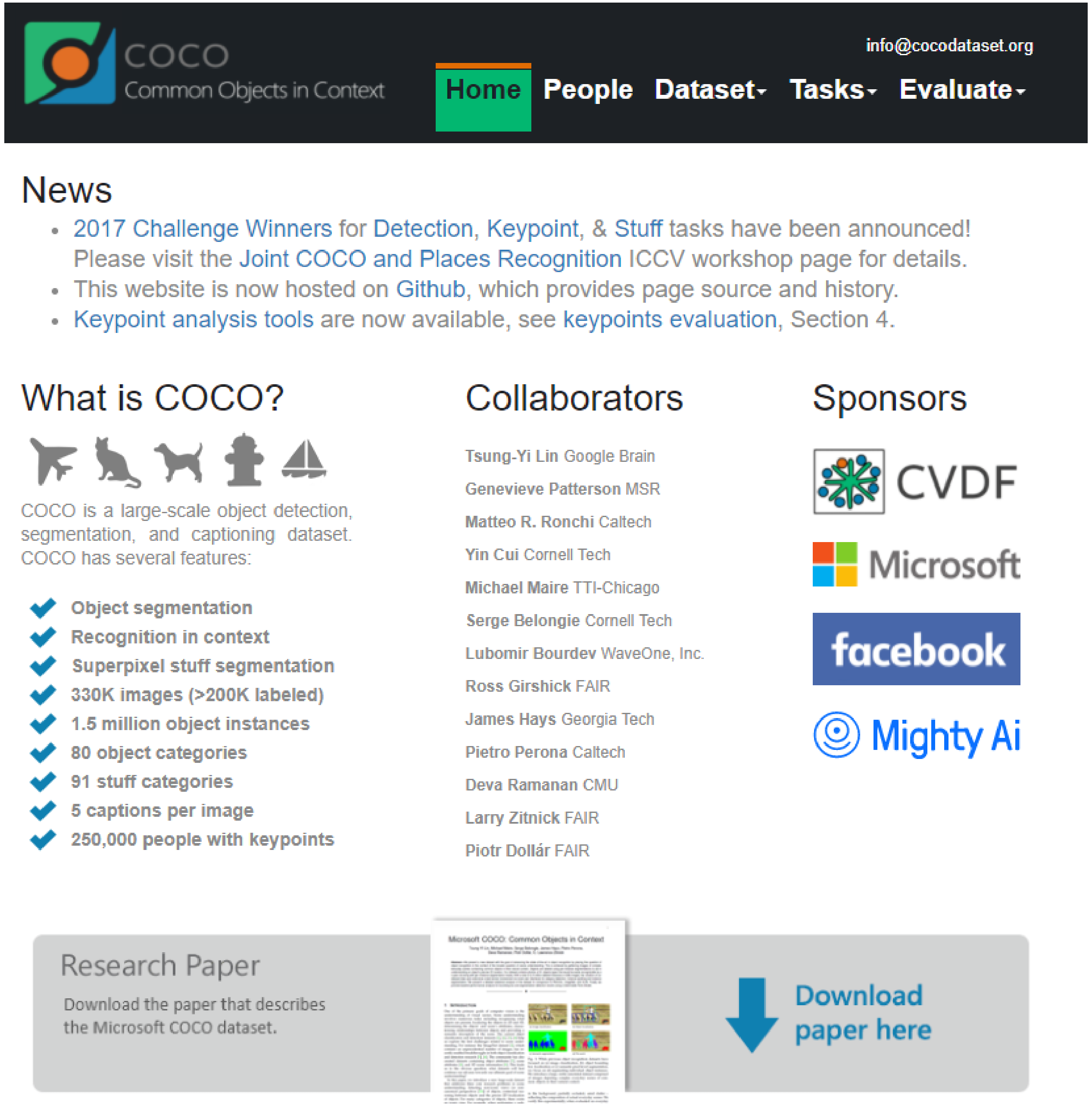
Kitti-trained models {#kitti-models}

Model name	Speed (ms)	Pascal mAP@0.5 (ms)	Outputs
faster_rcnn_resnet101_kitti	79	87	Boxes

Lottery prize!!!! Will be used to implement localization and velocity estimation

Open Images-trained models {#open-images-models}

Model name	Speed (ms)	Open Images mAP@0.5[^2]	Outputs
faster_rcnn_inception_resnet_v2_atrous_oid	727	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_oid	347		Boxes



COCO
Common Objects in Context

info@cocodataset.org

Home People Dataset▾ Tasks▾ Evaluate▾

News

- 2017 Challenge Winners for Detection, Keypoint, & Stuff tasks have been announced! Please visit the Joint COCO and Places Recognition ICCV workshop page for details.
- This website is now hosted on Github, which provides page source and history.
- Keypoint analysis tools are now available, see keypoints evaluation, Section 4.

What is COCO?

COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features:

- ✓ Object segmentation
- ✓ Recognition in context
- ✓ Superpixel stuff segmentation
- ✓ 330K images (>200K labeled)
- ✓ 1.5 million object instances
- ✓ 80 object categories
- ✓ 91 stuff categories
- ✓ 5 captions per image
- ✓ 250,000 people with keypoints

Collaborators

Tsung-Yi Lin Google Brain
Genevieve Patterson MSR
Matteo R. Ronchi Caltech
Yin Cui Cornell Tech
Michael Maire TTI-Chicago
Serge Belongie Cornell Tech
Lubomir Bourdev WaveOne, Inc.
Ross Girshick FAIR
James Hays Georgia Tech
Pietro Perona Caltech
Deva Ramanan CMU
Larry Zitnick FAIR
Piotr Dollár FAIR

Sponsors

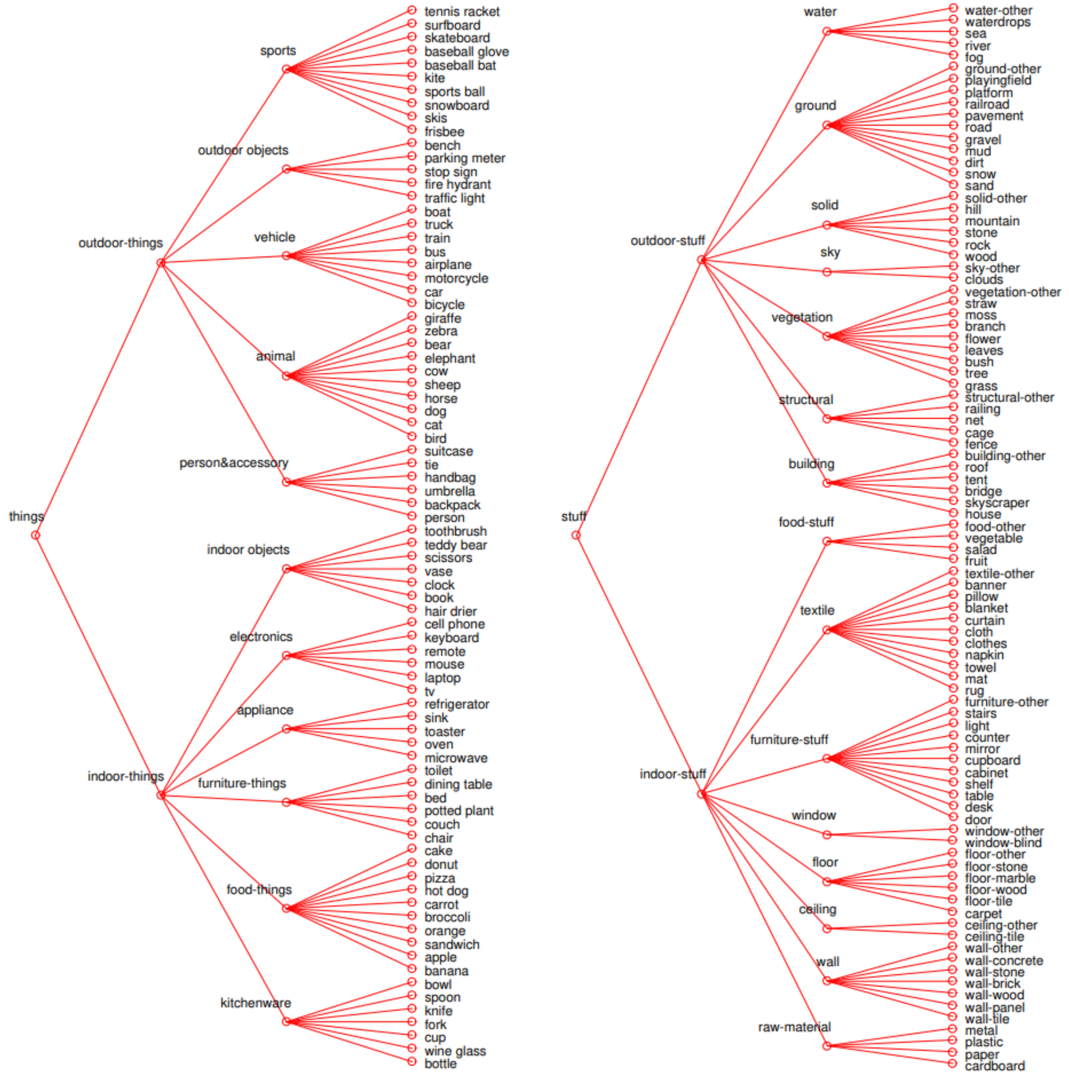
CVDF
Microsoft
facebook
Mighty Ai

Research Paper

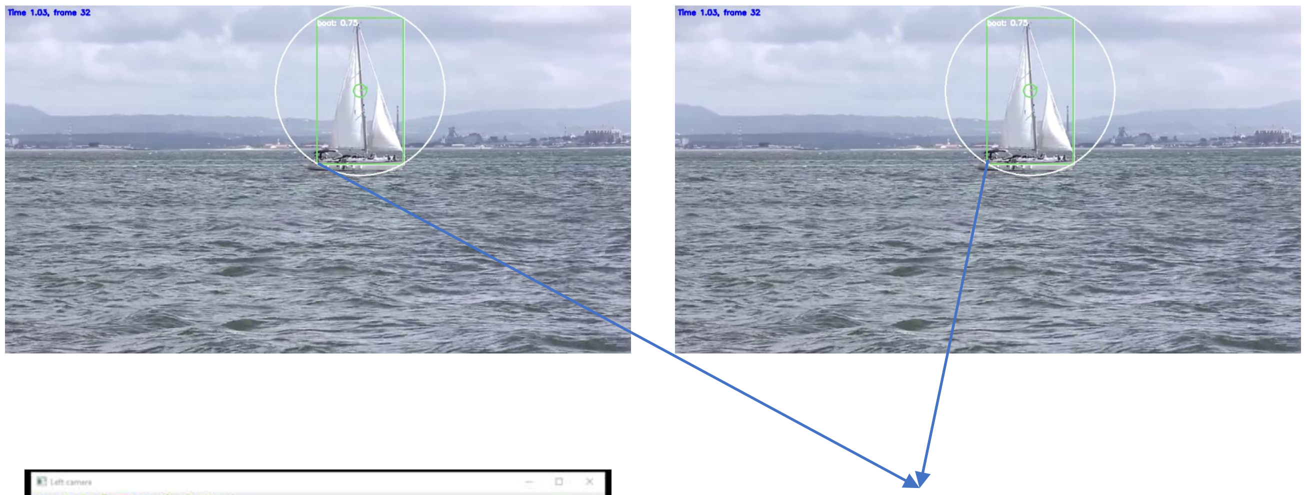
Download the paper that describes the Microsoft COCO dataset.

Microsoft COCO: Common Objects in Context

Download paper here

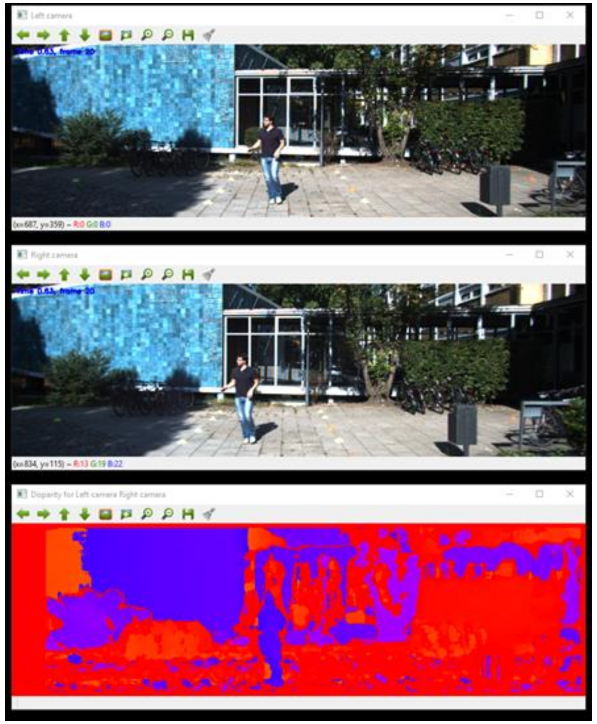


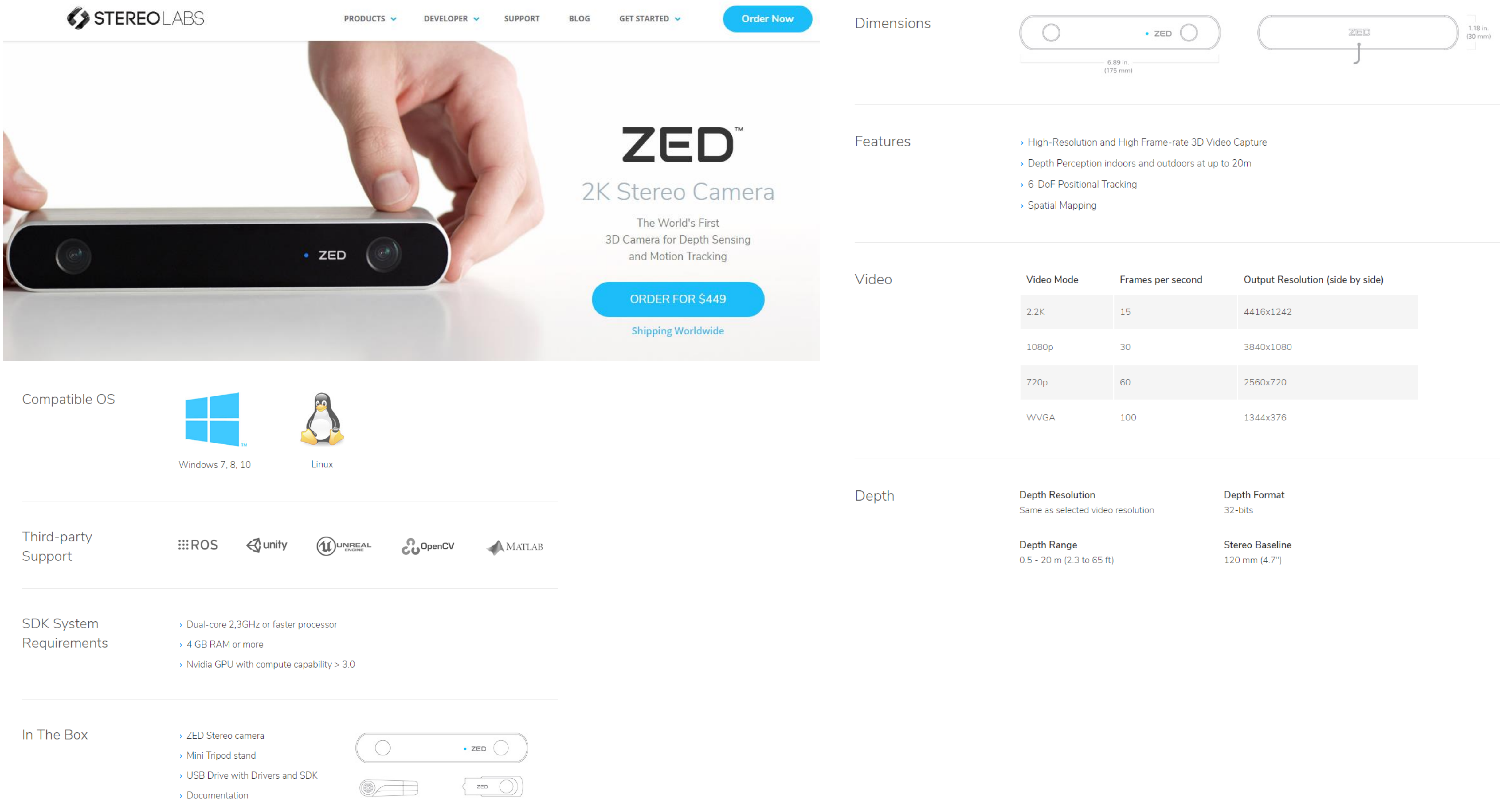
Stereo Vision


Left

Right

disparity d


Left camera
Time 0.63, frame 20
(x=687, y=359) ~ R:0 G:0 B:0
Right camera
Time 0.63, frame 20
(x=834, y=115) ~ R:13 G:19 B:22
Disparity for Left camera Right camera



ZED™

2K Stereo Camera

The World's First
3D Camera for Depth Sensing
and Motion Tracking

Compatible OS

Windows 7, 8, 10

Linux

Third-party Support

SDK System Requirements

- Dual-core 2,3GHz or faster processor
- 4 GB RAM or more
- Nvidia GPU with compute capability > 3.0

In The Box

- ZED Stereo camera
- Mini Tripod stand
- USB Drive with Drivers and SDK
- Documentation

Dimensions

Features

- High-Resolution and High Frame-rate 3D Video Capture
- Depth Perception indoors and outdoors at up to 20m
- 6-DoF Positional Tracking
- Spatial Mapping

Video

Video Mode	Frames per second	Output Resolution (side by side)
2.2K	15	4416x1242
1080p	30	3840x1080
720p	60	2560x720
WVGA	100	1344x376

Depth

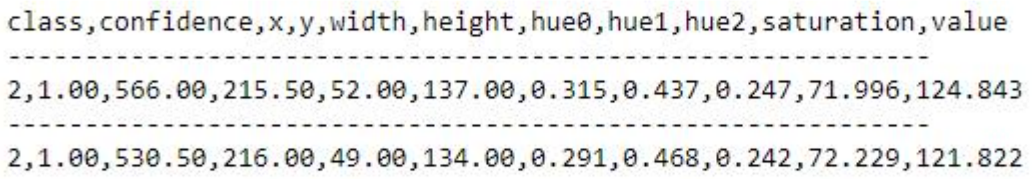
Depth Resolution
Same as selected video resolution

Depth Format
32-bits

Depth Range
0.5 - 20 m (2.3 to 65 ft)

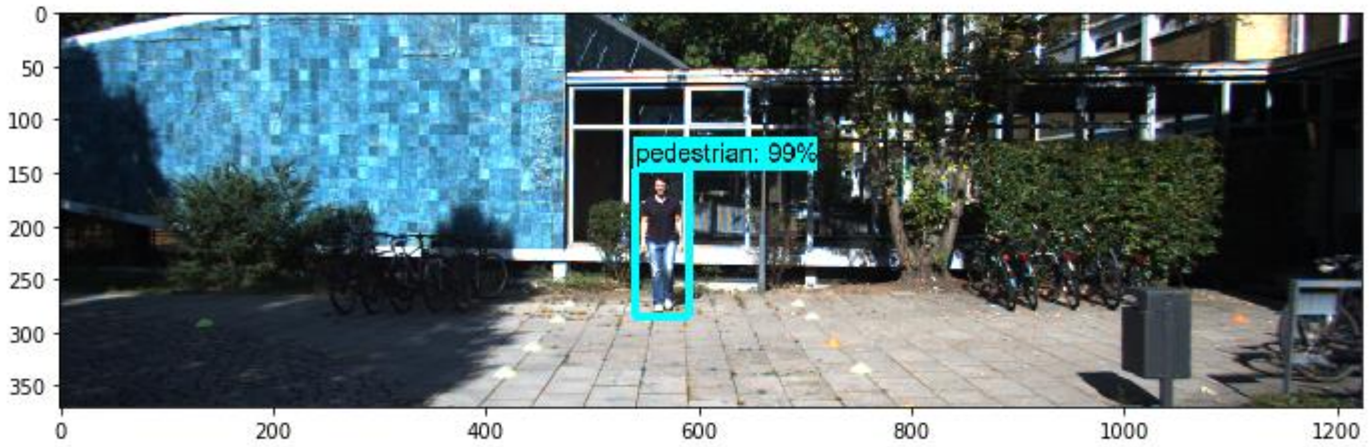
Stereo Baseline
120 mm (4.7")

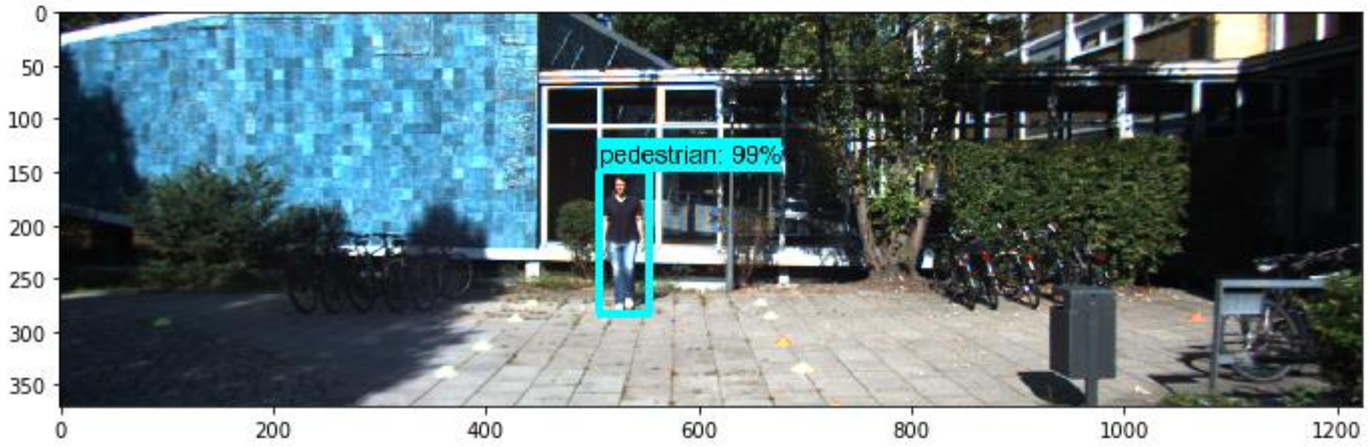
Mapping left and right image patterns



```
class,confidence,x,y,width,height,hue0,hue1,hue2,saturation,value
------------------------------------------------------------
2,1.00,566.00,215.50,52.00,137.00,0.315,0.437,0.247,71.996,124.843
------------------------------------------------------------
2,1.00,530.50,216.00,49.00,134.00,0.291,0.468,0.242,72.229,121.822
```

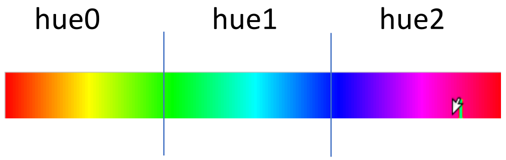
Pattern features to match

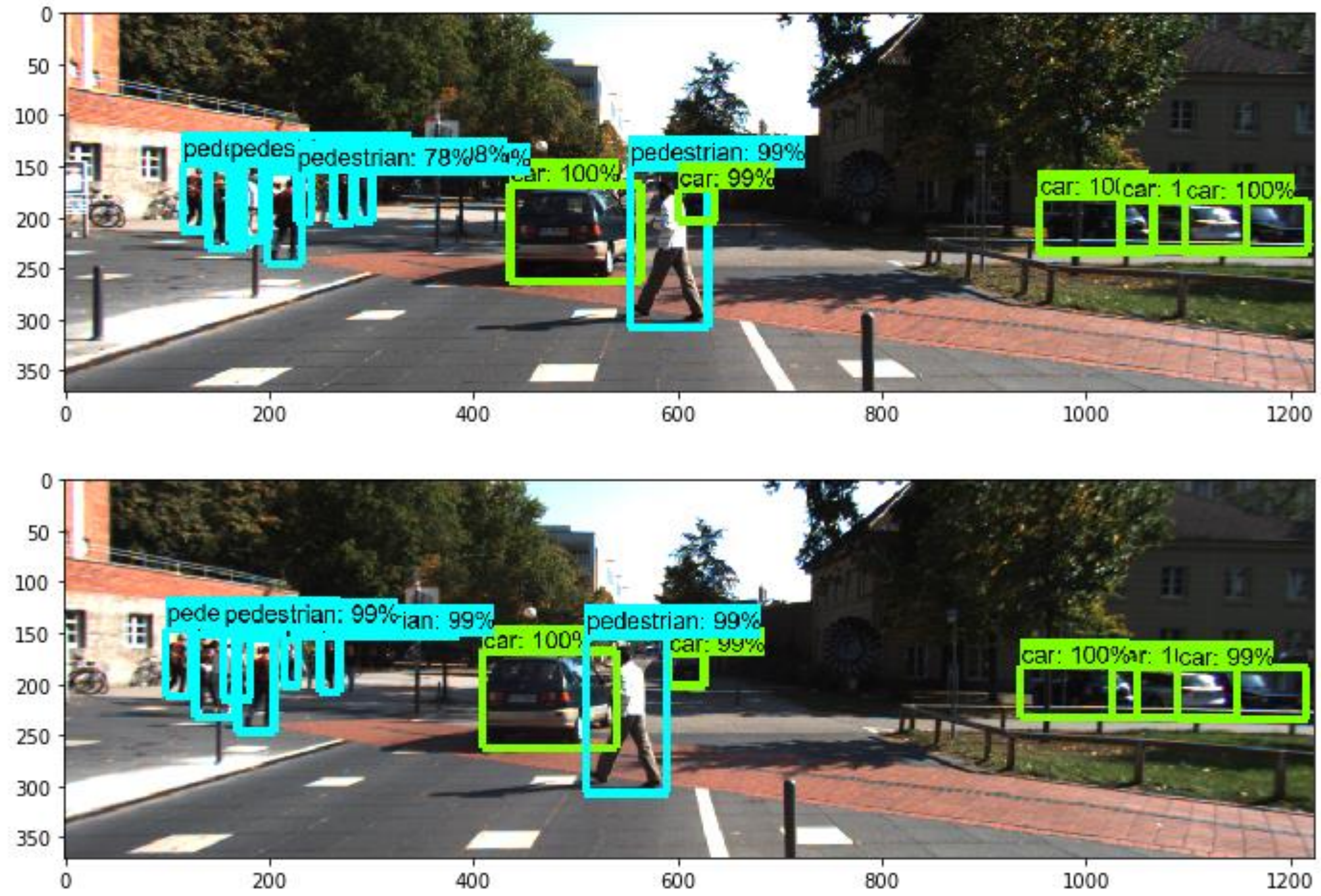




x,y = bounding box center location
saturation,value = mean values

3-bin hue histogram:





```
class,confidence,x,y,width,height,hue0,hue1,hue2,saturation,value
-----------------------------------------------------------
1,1.00,1010.50,206.50,113.00,53.00,0.354,0.438,0.208,74.793,48.389
1,1.00,1096.00,207.00,124.00,48.00,0.313,0.514,0.173,75.248,79.118
1,1.00,1157.50,208.50,121.00,49.00,0.274,0.551,0.175,79.360,81.186
1,1.00,500.50,215.00,129.00,94.00,0.367,0.389,0.244,75.836,70.045
2,1.00,592.00,229.50,78.00,155.00,0.550,0.254,0.196,59.824,104.288
1,1.00,619.50,187.50,37.00,31.00,0.286,0.323,0.391,58.958,58.623
2,1.00,159.00,190.50,36.00,79.00,0.434,0.220,0.346,77.767,116.533
2,1.00,293.50,180.50,17.00,45.00,0.354,0.299,0.346,78.097,70.915
2,1.00,127.50,180.00,25.00,64.00,0.493,0.174,0.333,97.595,144.548
2,1.00,216.00,200.00,34.00,90.00,0.422,0.256,0.322,82.717,112.057
2,1.00,184.50,184.00,33.00,72.00,0.413,0.229,0.358,68.688,159.507
2,1.00,177.50,185.50,33.00,75.00,0.395,0.244,0.361,70.105,148.906
2,0.98,272.50,179.50,19.00,51.00,0.397,0.261,0.342,80.279,69.196
2,0.79,235.00,180.50,16.00,47.00,0.483,0.209,0.309,94.992,127.202
-----------------------------------------------------------
1,1.00,1087.00,207.50,124.00,49.00,0.292,0.507,0.201,71.784,76.482
1,1.00,993.50,207.00,115.00,50.00,0.348,0.410,0.242,72.279,47.679
1,1.00,475.00,215.00,132.00,94.00,0.311,0.366,0.323,69.133,74.729
1,1.00,609.00,186.00,36.00,30.00,0.144,0.298,0.557,59.536,30.091
2,1.00,550.50,230.00,81.00,152.00,0.564,0.262,0.175,60.986,116.519
2,1.00,187.50,200.50,37.00,91.00,0.394,0.285,0.322,78.999,107.080
2,1.00,147.50,191.50,39.00,81.00,0.381,0.260,0.359,69.986,128.833
2,1.00,222.50,178.50,15.00,45.00,0.376,0.330,0.293,78.055,120.033
2,1.00,260.00,180.50,20.00,55.00,0.394,0.267,0.339,79.143,69.303
2,1.00,112.00,178.50,26.00,65.00,0.509,0.172,0.319,90.011,143.473
1,1.00,1153.50,209.00,127.00,50.00,0.209,0.593,0.198,77.370,78.260
2,0.99,169.00,180.50,28.00,67.00,0.432,0.317,0.251,75.630,159.326
```

Pattern matching based on feature difference is required!

Probabilistic model answering the question: What is the probability two patterns represent the same object?

Feature vector F:

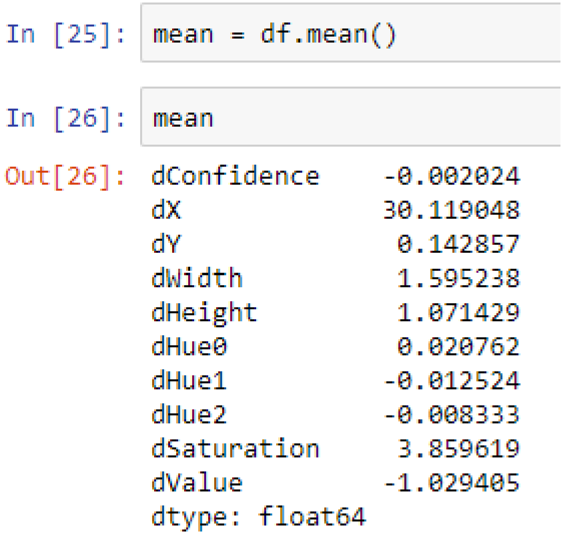
- confidence
- x
- y
- width
- heigth
- hue0
- hue1
- hue2
- saruration
- value

Note: Class is not included as it is **required** to be the same

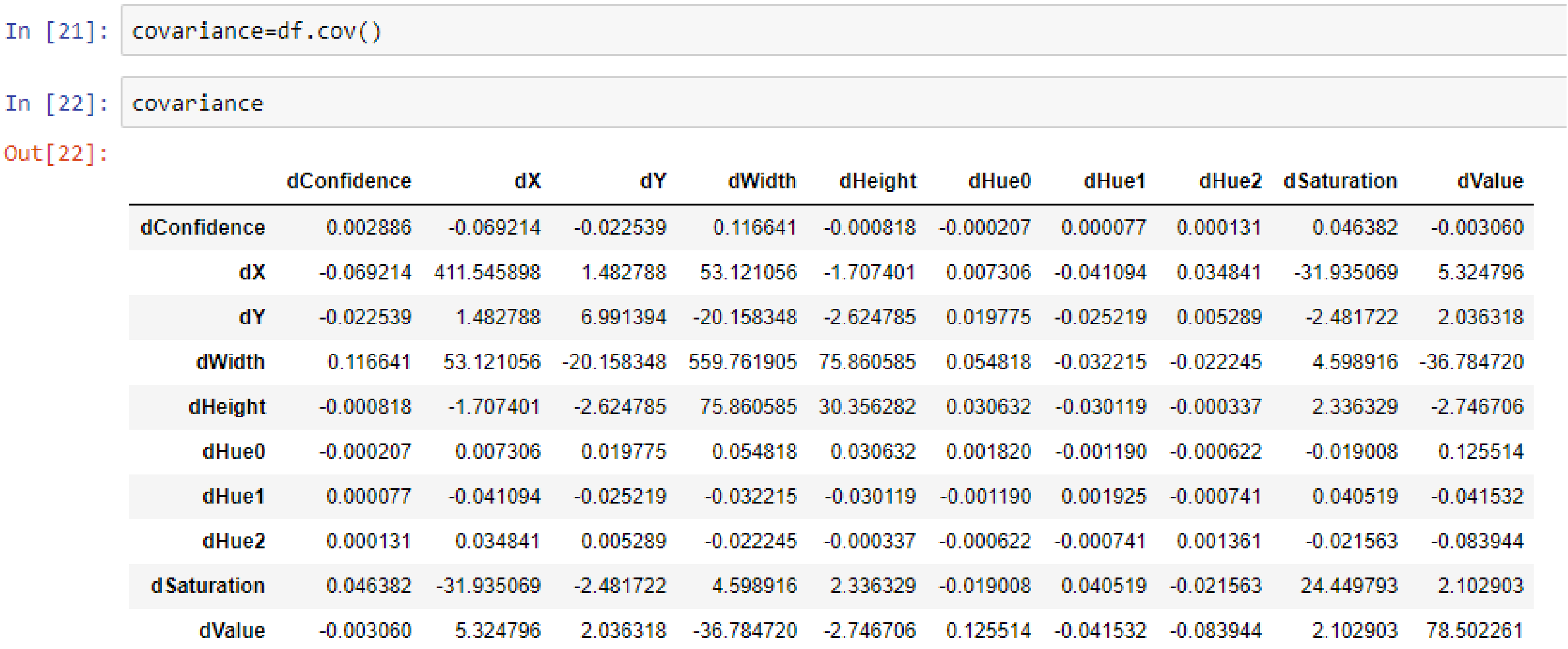
Assumption:

P(i and j are same pattern) ~ $N(F_i - F_j \mid \mu_F , V_F)$

μ_F , V_F were estimated by matching 84 patterns in 28 KITTI stereo image pairs representing city, residential, campus and person categories, including both cars and pedestrians.



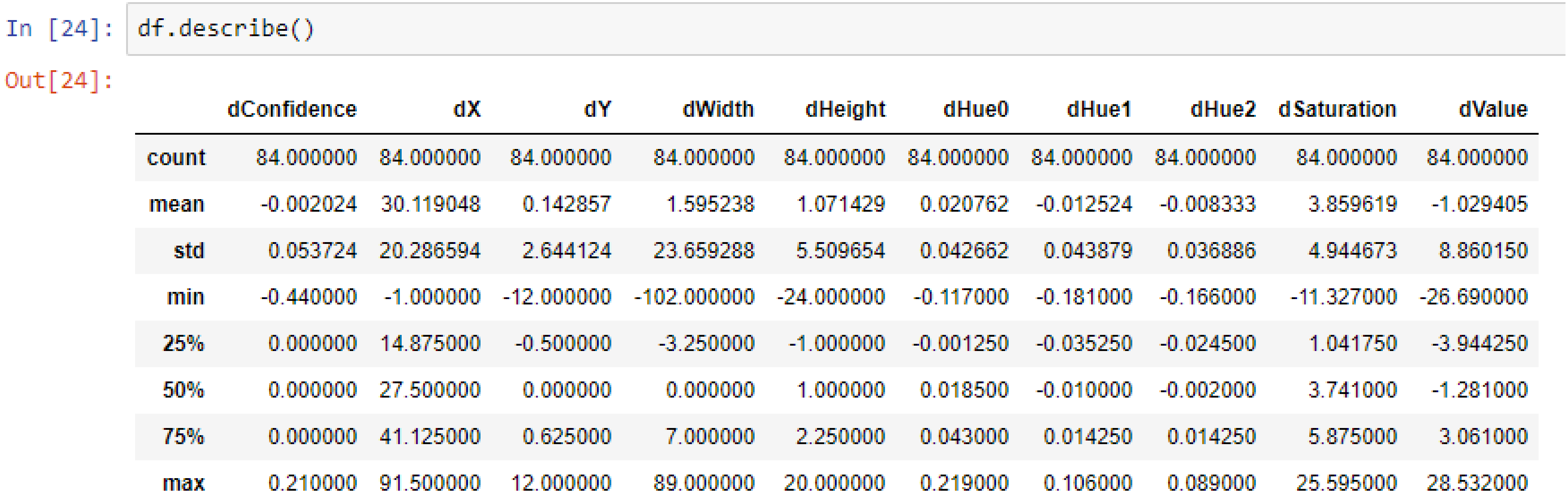
```
In [25]:  mean = df.mean()

In [26]:  mean

Out[26]:  dConfidence    -0.002024
          dX             30.119048
          dY              0.142857
          dWidth          1.595238
          dHeight         1.071429
          dHue0           0.020762
          dHue1          -0.012524
          dHue2          -0.008333
          dSaturation     3.859619
          dValue         -1.029405
          dtype: float64
```



```
In [21]:  covariance=df.cov()

In [22]:  covariance

Out[22]:
```

	dConfidence	dX	dY	dWidth	dHeight	dHue0	dHue1	dHue2	dSaturation	dValue
dConfidence	0.002886	-0.069214	-0.022539	0.116641	-0.000818	-0.000207	0.000077	0.000131	0.046382	-0.003060
dX	-0.069214	411.545898	1.482788	53.121056	-1.707401	0.007306	-0.041094	0.034841	-31.935069	5.324796
dY	-0.022539	1.482788	6.991394	-20.158348	-2.624785	0.019775	-0.025219	0.005289	-2.481722	2.036318
dWidth	0.116641	53.121056	-20.158348	559.761905	75.860585	0.054818	-0.032215	-0.022245	4.598916	-36.784720
dHeight	-0.000818	-1.707401	-2.624785	75.860585	30.356282	0.030632	-0.030119	-0.000337	2.336329	-2.746706
dHue0	-0.000207	0.007306	0.019775	0.054818	0.030632	0.001820	-0.001190	-0.000622	-0.019008	0.125514
dHue1	0.000077	-0.041094	-0.025219	-0.032215	-0.030119	-0.001190	0.001925	-0.000741	0.040519	-0.041532
dHue2	0.000131	0.034841	0.005289	-0.022245	-0.000337	-0.000622	-0.000741	0.001361	-0.021563	-0.083944
dSaturation	0.046382	-31.935069	-2.481722	4.598916	2.336329	-0.019008	0.040519	-0.021563	24.449793	2.102903
dValue	-0.003060	5.324796	2.036318	-36.784720	-2.746706	0.125514	-0.041532	-0.083944	2.102903	78.502261

Note: Mean disparity (dX) is appr. 30 pixels

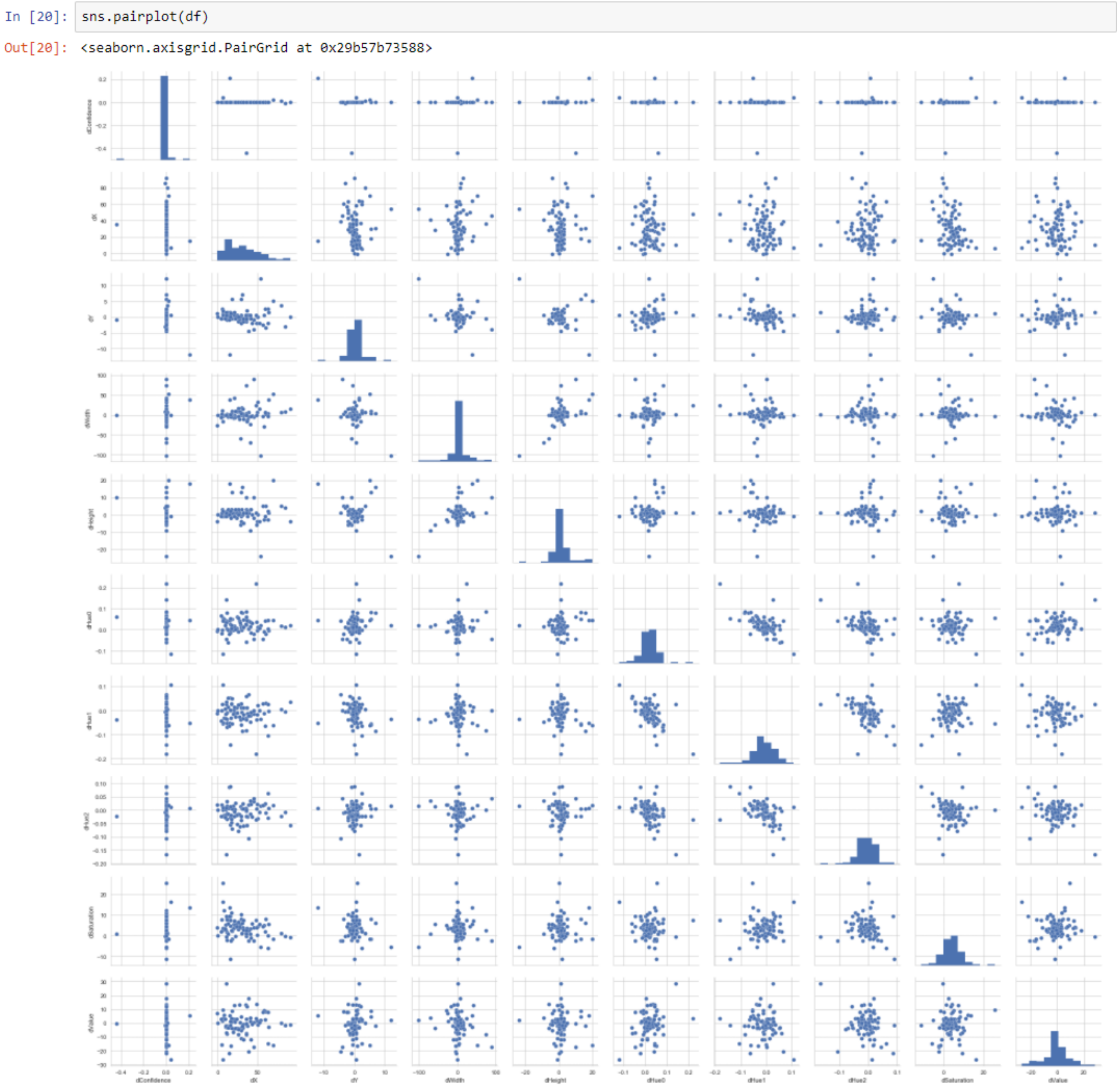


In [24]:

```
df.describe()
```

Out[24]:

	dConfidence	dX	dY	dWidth	dHeight	dHue0	dHue1	dHue2	dSaturation	dValue
count	84.000000	84.000000	84.000000	84.000000	84.000000	84.000000	84.000000	84.000000	84.000000	84.000000
mean	-0.002024	30.119048	0.142857	1.595238	1.071429	0.020762	-0.012524	-0.008333	3.859619	-1.029405
std	0.053724	20.286594	2.644124	23.659288	5.509654	0.042662	0.043879	0.036886	4.944673	8.860150
min	-0.440000	-1.000000	-12.000000	-102.000000	-24.000000	-0.117000	-0.181000	-0.166000	-11.327000	-26.690000
25%	0.000000	14.875000	-0.500000	-3.250000	-1.000000	-0.001250	-0.035250	-0.024500	1.041750	-3.944250
50%	0.000000	27.500000	0.000000	0.000000	1.000000	0.018500	-0.010000	-0.002000	3.741000	-1.281000
75%	0.000000	41.125000	0.625000	7.000000	2.250000	0.043000	0.014250	0.014250	5.875000	3.061000
max	0.210000	91.500000	12.000000	89.000000	20.000000	0.219000	0.106000	0.089000	25.595000	28.532000



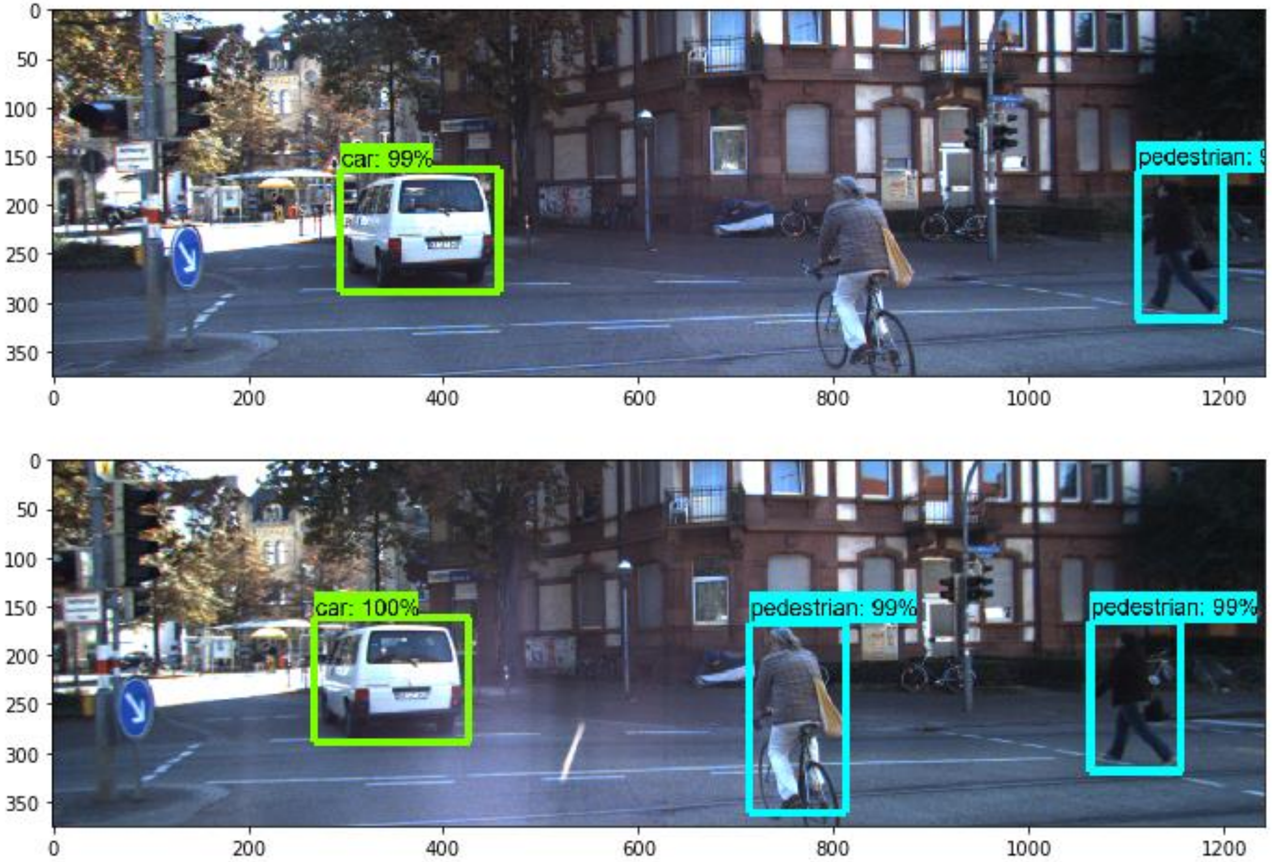
```
In [20]: sns.pairplot(df)

Out[20]: <seaborn.axisgrid.PairGrid at 0x29b57b73588>
```

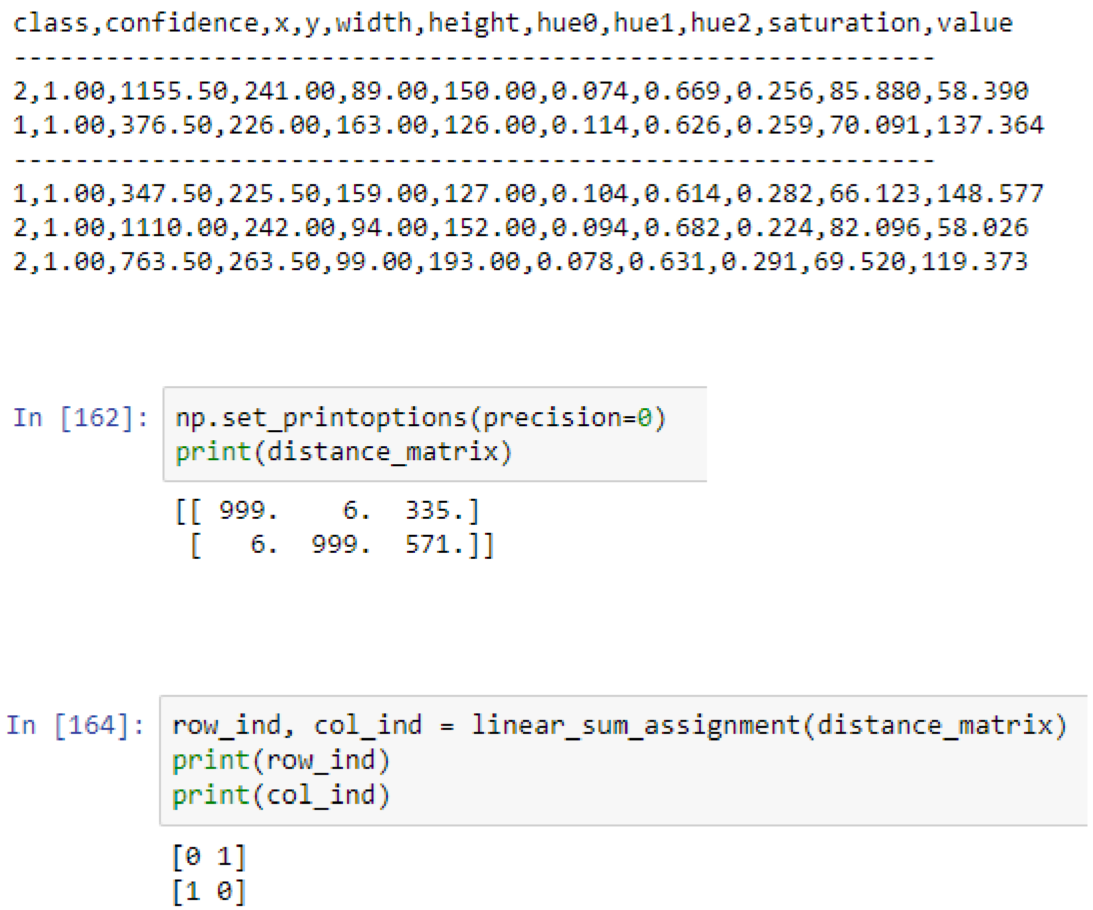
Pattern matching is done using Hungarian algorithm with the distance metrics:

d_{ij} = -log(P(i and j are same pattern)) = $-\log(N(F_i - F_j \mid \mu_F , V_F))$

If the probability that the patterns are same is near 1, the distance will be near zero. As the probability decreases, the distance increases. The log is required to compare small numbers without numerical issues.

Simple example

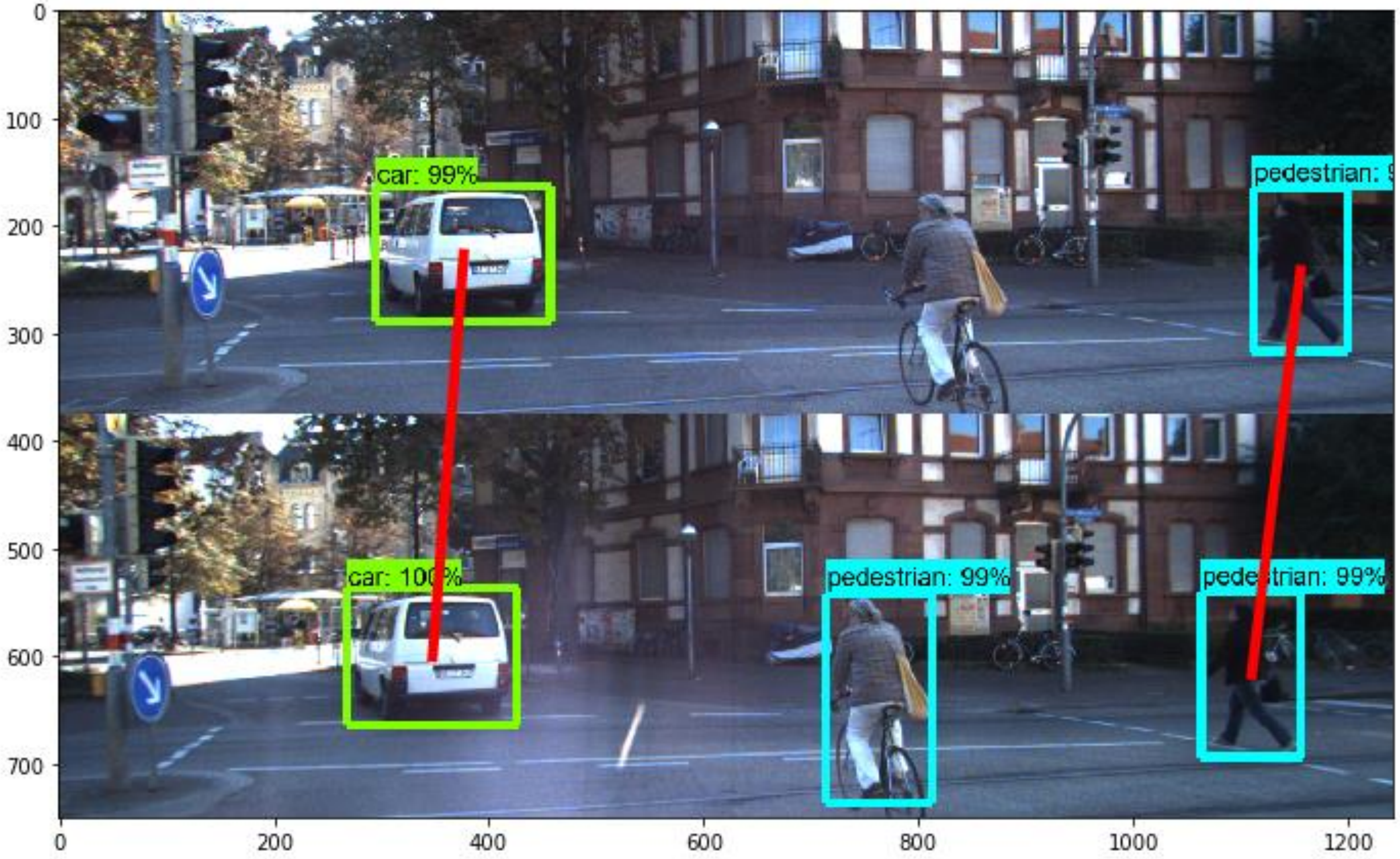




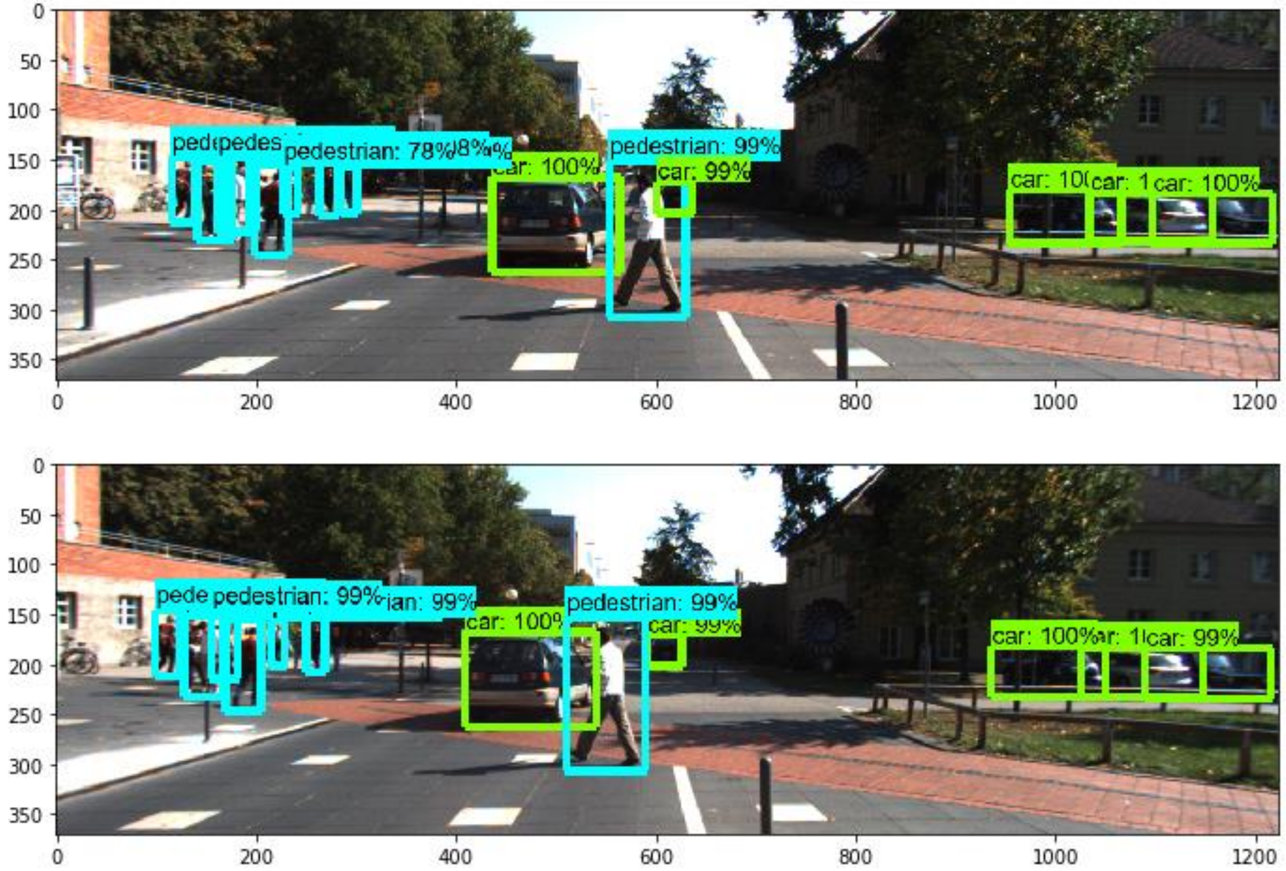
```
class,confidence,x,y,width,height,hue0,hue1,hue2,saturation,value
------------------------------------------------------------
2,1.00,1155.50,241.00,89.00,150.00,0.074,0.669,0.256,85.880,58.390
1,1.00,376.50,226.00,163.00,126.00,0.114,0.626,0.259,70.091,137.364
------------------------------------------------------------
1,1.00,347.50,225.50,159.00,127.00,0.104,0.614,0.282,66.123,148.577
2,1.00,1110.00,242.00,94.00,152.00,0.094,0.682,0.224,82.096,58.026
2,1.00,763.50,263.50,99.00,193.00,0.078,0.631,0.291,69.520,119.373
```

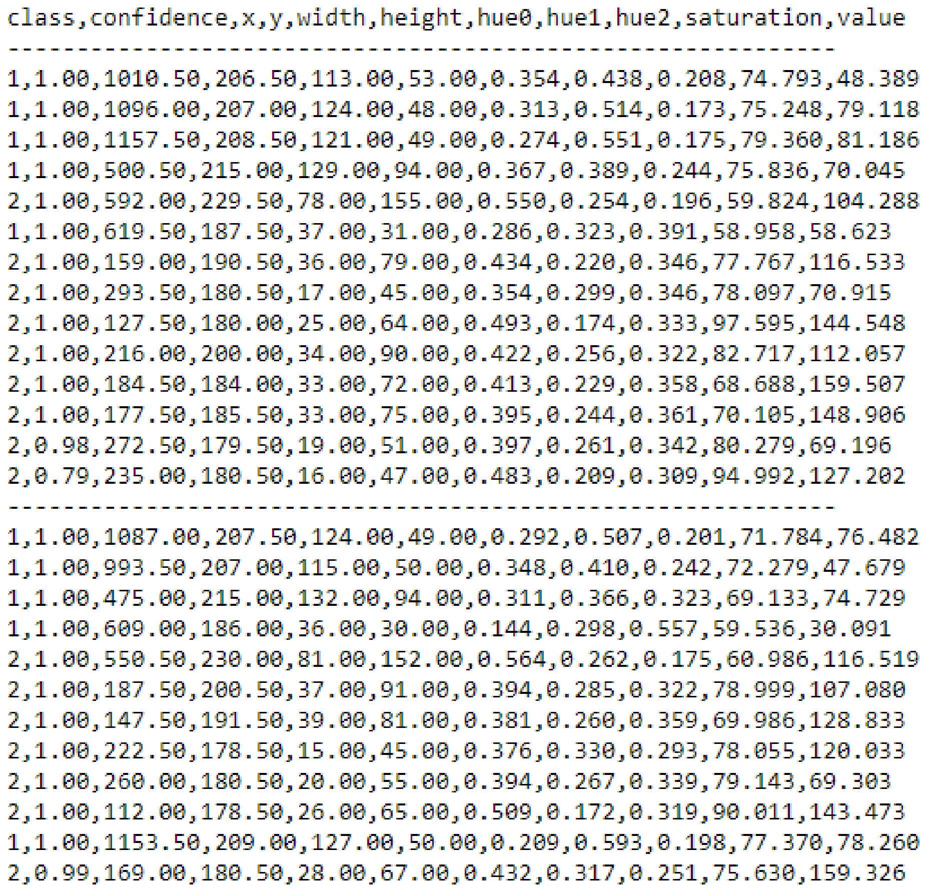
```
In [162]: np.set_printoptions(precision=0)
          print(distance_matrix)

          [[ 999.    6.  335.]
           [   6.  999.  571.]]
```

```
In [164]: row_ind, col_ind = linear_sum_assignment(distance_matrix)
          print(row_ind)
          print(col_ind)

          [0 1]
          [1 0]
```

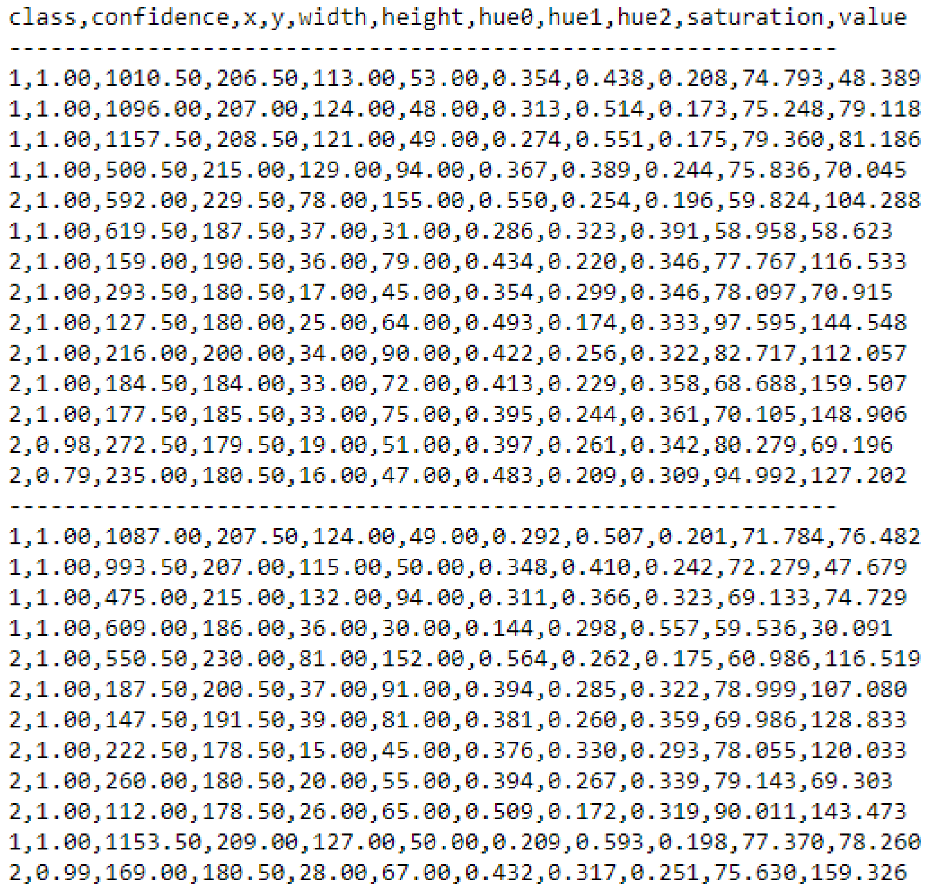

car: 99%
pedestrian: 9
car: 100%
pedestrian: 99%
pedestrian: 99%
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

Complicated example

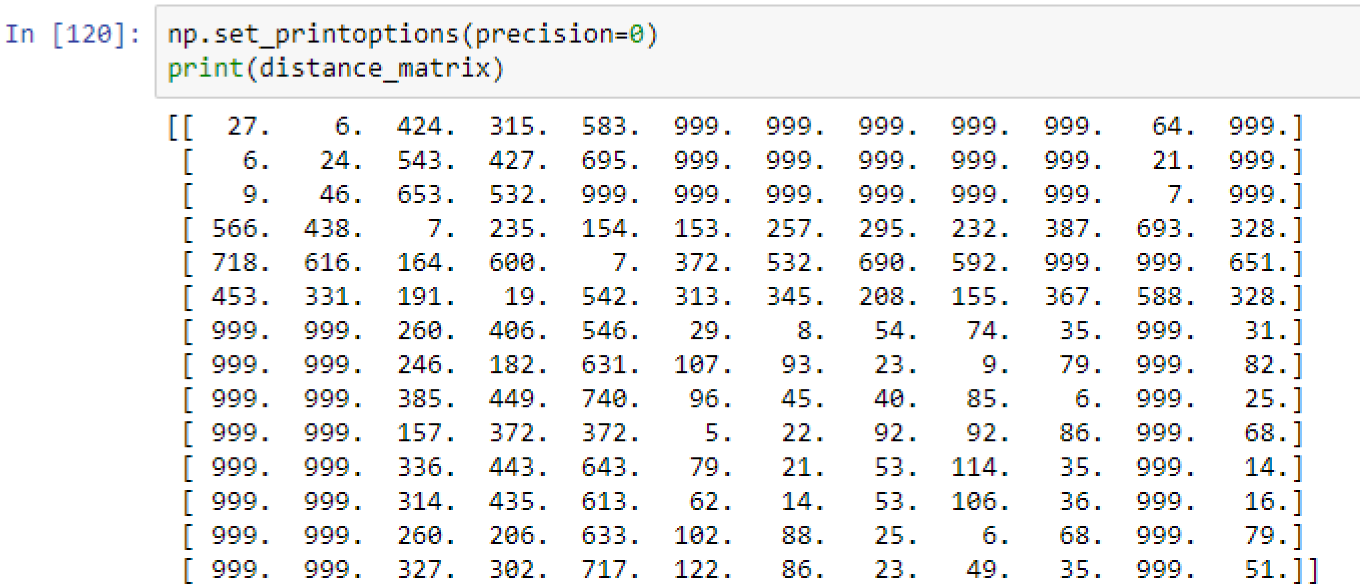




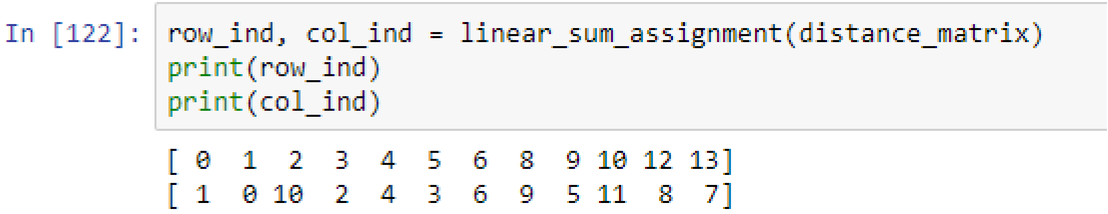
```
class,confidence,x,y,width,height,hue0,hue1,hue2,saturation,value
------------------------------------------------------------
1,1.00,1010.50,206.50,113.00,53.00,0.354,0.438,0.208,74.793,48.389
1,1.00,1096.00,207.00,124.00,48.00,0.313,0.514,0.173,75.248,79.118
1,1.00,1157.50,208.50,121.00,49.00,0.274,0.551,0.175,79.360,81.186
1,1.00,500.50,215.00,129.00,94.00,0.367,0.389,0.244,75.836,70.045
2,1.00,592.00,229.50,78.00,155.00,0.550,0.254,0.196,59.824,104.288
1,1.00,619.50,187.50,37.00,31.00,0.286,0.323,0.391,58.958,58.623
2,1.00,159.00,190.50,36.00,79.00,0.434,0.220,0.346,77.767,116.533
2,1.00,293.50,180.50,17.00,45.00,0.354,0.299,0.346,78.097,70.915
2,1.00,127.50,180.00,25.00,64.00,0.493,0.174,0.333,97.595,144.548
2,1.00,216.00,200.00,34.00,90.00,0.422,0.256,0.322,82.717,112.057
2,1.00,184.50,184.00,33.00,72.00,0.413,0.229,0.358,68.688,159.507
2,1.00,177.50,185.50,33.00,75.00,0.395,0.244,0.361,70.105,148.906
2,0.98,272.50,179.50,19.00,51.00,0.397,0.261,0.342,80.279,69.196
2,0.79,235.00,180.50,16.00,47.00,0.483,0.209,0.309,94.992,127.202
------------------------------------------------------------
1,1.00,1087.00,207.50,124.00,49.00,0.292,0.507,0.201,71.784,76.482
1,1.00,993.50,207.00,115.00,50.00,0.348,0.410,0.242,72.279,47.679
1,1.00,475.00,215.00,132.00,94.00,0.311,0.366,0.323,69.133,74.729
1,1.00,609.00,186.00,36.00,30.00,0.144,0.298,0.557,59.536,30.091
2,1.00,550.50,230.00,81.00,152.00,0.564,0.262,0.175,60.986,116.519
2,1.00,187.50,200.50,37.00,91.00,0.394,0.285,0.322,78.999,107.080
2,1.00,147.50,191.50,39.00,81.00,0.381,0.260,0.359,69.986,128.833
2,1.00,222.50,178.50,15.00,45.00,0.376,0.330,0.293,78.055,120.033
2,1.00,260.00,180.50,20.00,55.00,0.394,0.267,0.339,79.143,69.303
2,1.00,112.00,178.50,26.00,65.00,0.509,0.172,0.319,90.011,143.473
1,1.00,1153.50,209.00,127.00,50.00,0.209,0.593,0.198,77.370,78.260
2,0.99,169.00,180.50,28.00,67.00,0.432,0.317,0.251,75.630,159.326
```



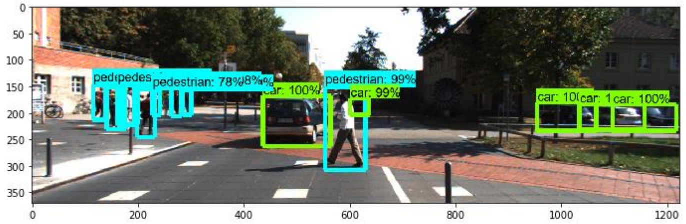
```
class,confidence,x,y,width,height,hue0,hue1,hue2,saturation,value
------------------------------------------------------------------
1,1.00,1010.50,206.50,113.00,53.00,0.354,0.438,0.208,74.793,48.389
1,1.00,1096.00,207.00,124.00,48.00,0.313,0.514,0.173,75.248,79.118
1,1.00,1157.50,208.50,121.00,49.00,0.274,0.551,0.175,79.360,81.186
1,1.00,500.50,215.00,129.00,94.00,0.367,0.389,0.244,75.836,70.045
2,1.00,592.00,229.50,78.00,155.00,0.550,0.254,0.196,59.824,104.288
1,1.00,619.50,187.50,37.00,31.00,0.286,0.323,0.391,58.958,58.623
2,1.00,159.00,190.50,36.00,79.00,0.434,0.220,0.346,77.767,116.533
2,1.00,293.50,180.50,17.00,45.00,0.354,0.299,0.346,78.097,70.915
2,1.00,127.50,180.00,25.00,64.00,0.493,0.174,0.333,97.595,144.548
2,1.00,216.00,200.00,34.00,90.00,0.422,0.256,0.322,82.717,112.057
2,1.00,184.50,184.00,33.00,72.00,0.413,0.229,0.358,68.688,159.507
2,1.00,177.50,185.50,33.00,75.00,0.395,0.244,0.361,70.105,148.906
2,0.98,272.50,179.50,19.00,51.00,0.397,0.261,0.342,80.279,69.196
2,0.79,235.00,180.50,16.00,47.00,0.483,0.209,0.309,94.992,127.202
------------------------------------------------------------------
1,1.00,1087.00,207.50,124.00,49.00,0.292,0.507,0.201,71.784,76.482
1,1.00,993.50,207.00,115.00,50.00,0.348,0.410,0.242,72.279,47.679
1,1.00,475.00,215.00,132.00,94.00,0.311,0.366,0.323,69.133,74.729
1,1.00,609.00,186.00,36.00,30.00,0.144,0.298,0.557,59.536,30.091
2,1.00,550.50,230.00,81.00,152.00,0.564,0.262,0.175,60.986,116.519
2,1.00,187.50,200.50,37.00,91.00,0.394,0.285,0.322,78.999,107.080
2,1.00,147.50,191.50,39.00,81.00,0.381,0.260,0.359,69.986,128.833
2,1.00,222.50,178.50,15.00,45.00,0.376,0.330,0.293,78.055,120.033
2,1.00,260.00,180.50,20.00,55.00,0.394,0.267,0.339,79.143,69.303
2,1.00,112.00,178.50,26.00,65.00,0.509,0.172,0.319,90.011,143.473
1,1.00,1153.50,209.00,127.00,50.00,0.209,0.593,0.198,77.370,78.260
2,0.99,169.00,180.50,28.00,67.00,0.432,0.317,0.251,75.630,159.326
```

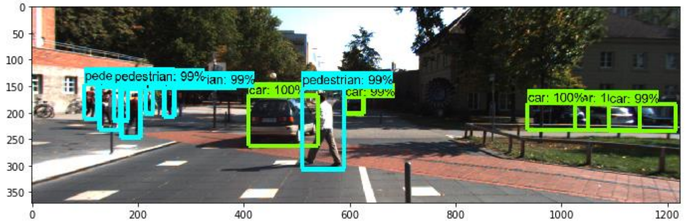


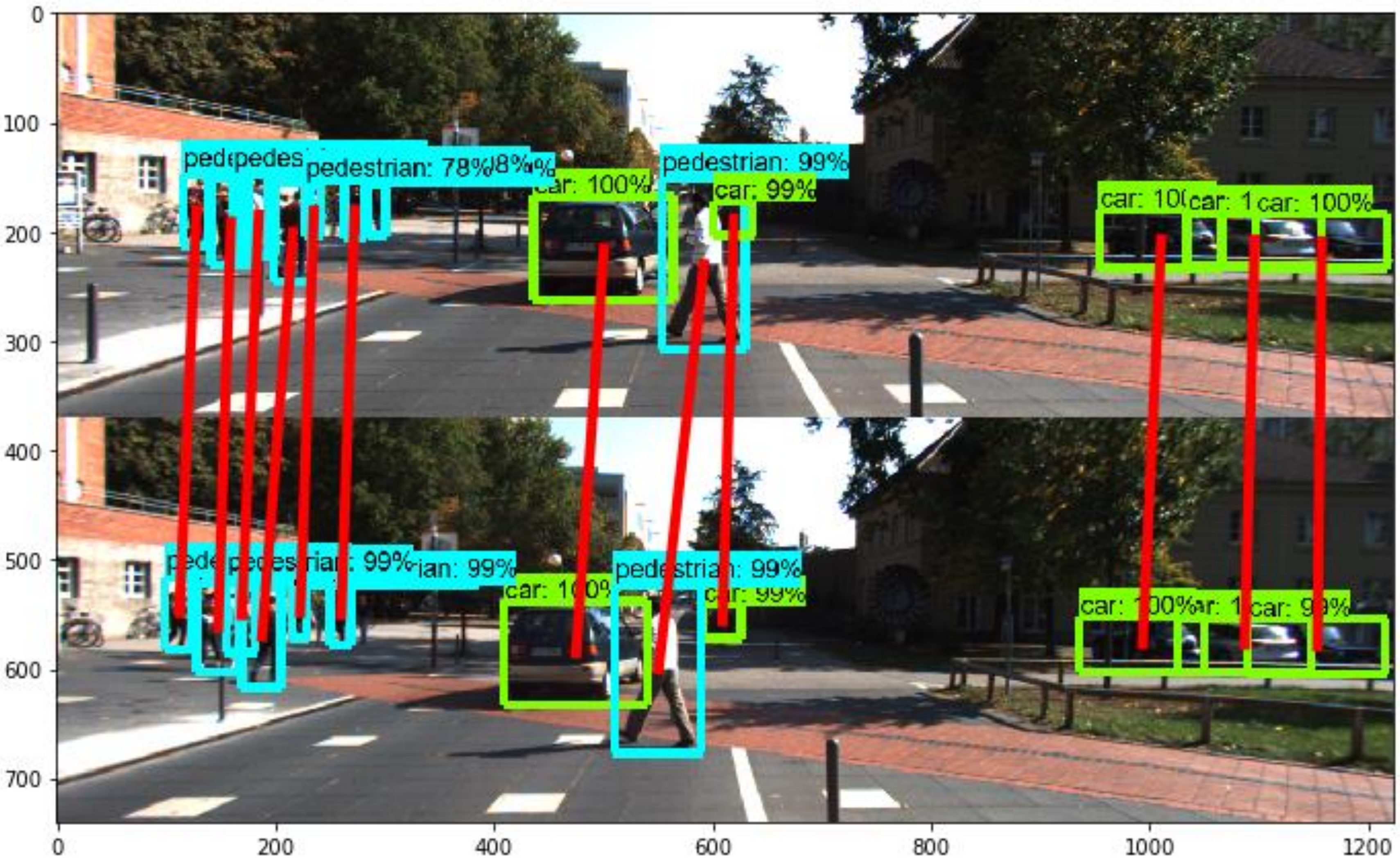
```
In [120]: np.set_printoptions(precision=0)
          print(distance_matrix)

[[  27.    6.  424.  315.  583.  999.  999.  999.  999.  999.   64.  999.]
 [   6.   24.  543.  427.  695.  999.  999.  999.  999.  999.   21.  999.]
 [   9.   46.  653.  532.  999.  999.  999.  999.  999.  999.    7.  999.]
 [ 566.  438.    7.  235.  154.  153.  257.  295.  232.  387.  693.  328.]
 [ 718.  616.  164.  600.    7.  372.  532.  690.  592.  999.  999.  651.]
 [ 453.  331.  191.   19.  542.  313.  345.  208.  155.  367.  588.  328.]
 [ 999.  999.  260.  406.  546.   29.    8.   54.   74.   35.  999.   31.]
 [ 999.  999.  246.  182.  631.  107.   93.   23.    9.   79.  999.   82.]
 [ 999.  999.  385.  449.  740.   96.   45.   40.   85.    6.  999.   25.]
 [ 999.  999.  157.  372.  372.    5.   22.   92.   92.   86.  999.   68.]
 [ 999.  999.  336.  443.  643.   79.   21.   53.  114.   35.  999.   14.]
 [ 999.  999.  314.  435.  613.   62.   14.   53.  106.   36.  999.   16.]
 [ 999.  999.  260.  206.  633.  102.   88.   25.    6.   68.  999.   79.]
 [ 999.  999.  327.  302.  717.  122.   86.   23.   49.   35.  999.   51.]]
```



```
In [122]: row_ind, col_ind = linear_sum_assignment(distance_matrix)
          print(row_ind)
          print(col_ind)

[ 0  1  2  3  4  5  6  8  9 10 12 13]
[ 1  0 10  2  4  3  6  9  5 11  8  7]
```

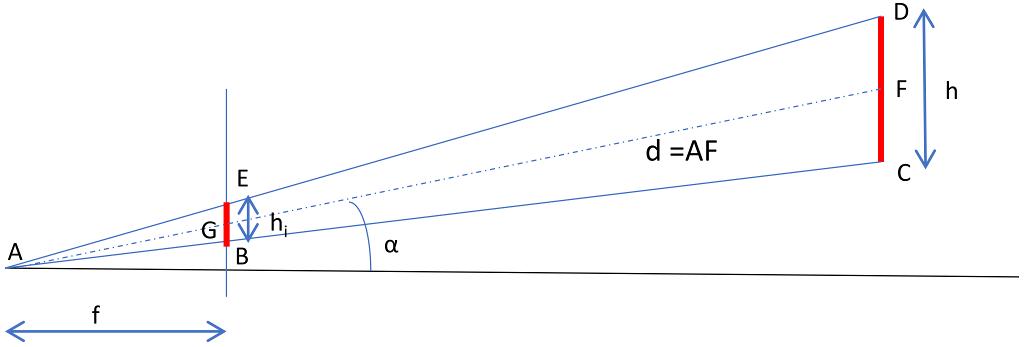





0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
pede
pedes
pedestrian: 78%
8%
%
car: 100%
pedestrian: 99%
car: 99%
car: 10
car: 1
car: 100%
pede
pedes
trian: 99%
ian: 99%
car: 100%
pedestrian: 99%
car: 99%
car: 100%
ar: 1
car: 99%

After implementing stereo vision, we have two distance estimates:

1. Distance based on stereo vision. Accurate in short distances (for Kitti, 20 meters)
2. Distance based on size. Can be used in long distances where stereo vision is inaccurate.



$$d_{size} = \frac{f * r}{\cos(\alpha) * \cos(\beta) * r_i * s_h / p_h}$$

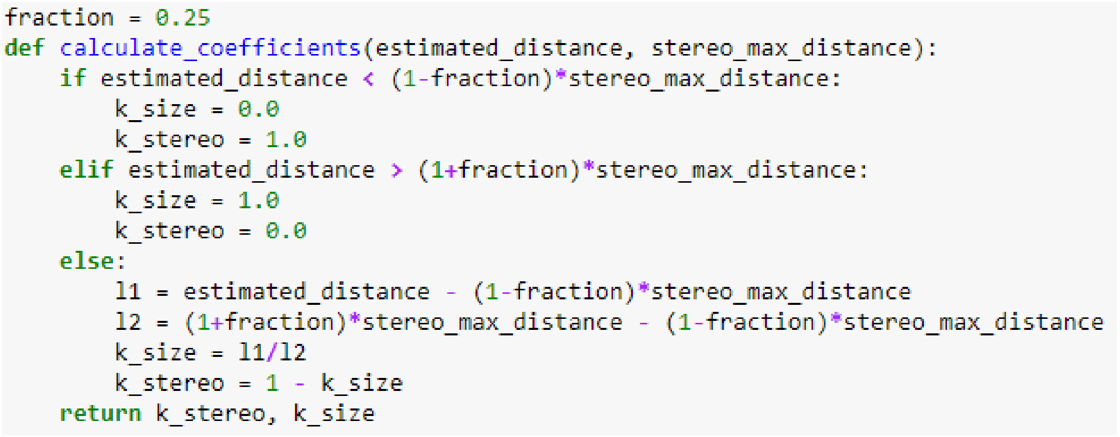
$$d_{stereo} = \frac{f * b}{\cos(\alpha) * \cos(\beta) * ds * s_w / p_w}$$

s_h= *sensor height (m)*
p_h= *image height (pixels)*
r_i = *pattern radius (pixels)*
r = body radius (m), mean from class specific distribution
f = focal length (m)
α = altitude (rad)
β = azimuth (rad)

s_w= *sensor width (m)*
p_w= *image width (pixels)*
f = focal length (m)
b = base line (m)
ds = disparity (pixels)
α = altitude (rad)
β = azimuth (rad)

Combining distance estimates:

$$d = k_{stereo} * d_{stereo} + k_{size} * d_{size}$$

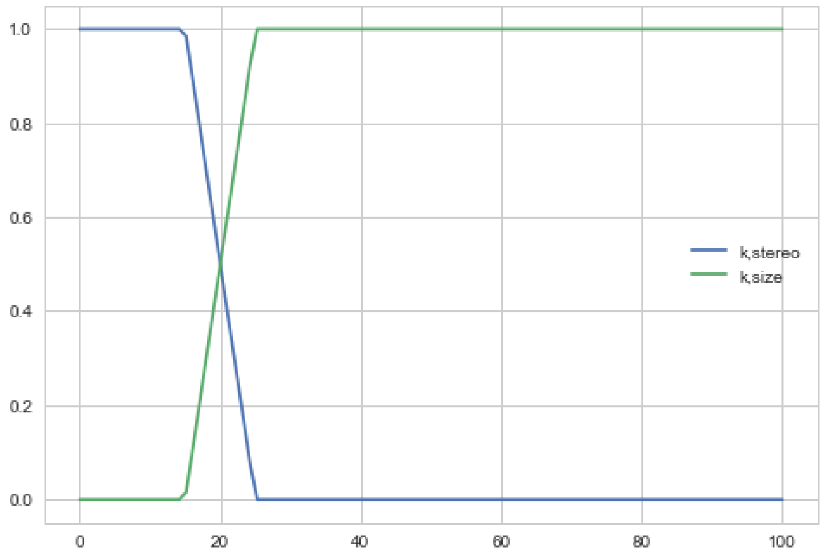


```
fraction = 0.25
def calculate_coefficients(estimated_distance, stereo_max_distance):
    if estimated_distance < (1-fraction)*stereo_max_distance:
        k_size = 0.0
        k_stereo = 1.0
    elif estimated_distance > (1+fraction)*stereo_max_distance:
        k_size = 1.0
        k_stereo = 0.0
    else:
        l1 = estimated_distance - (1-fraction)*stereo_max_distance
        l2 = (1+fraction)*stereo_max_distance - (1-fraction)*stereo_max_distance
        k_size = l1/l2
        k_stereo = 1 - k_size
    return k_stereo, k_size
```

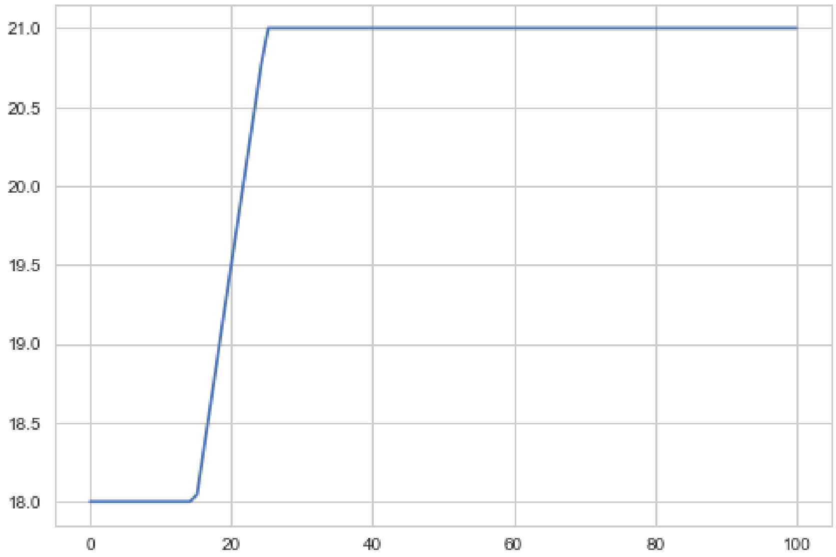
Initialization:

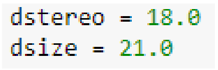
$$estimated\_distance = 0.5 * d_{stereo} + 0.5 * d_{size}$$

Procedure is iterated until convergence or max_iter (or just used once?)



stereo_max_distance = 20.0

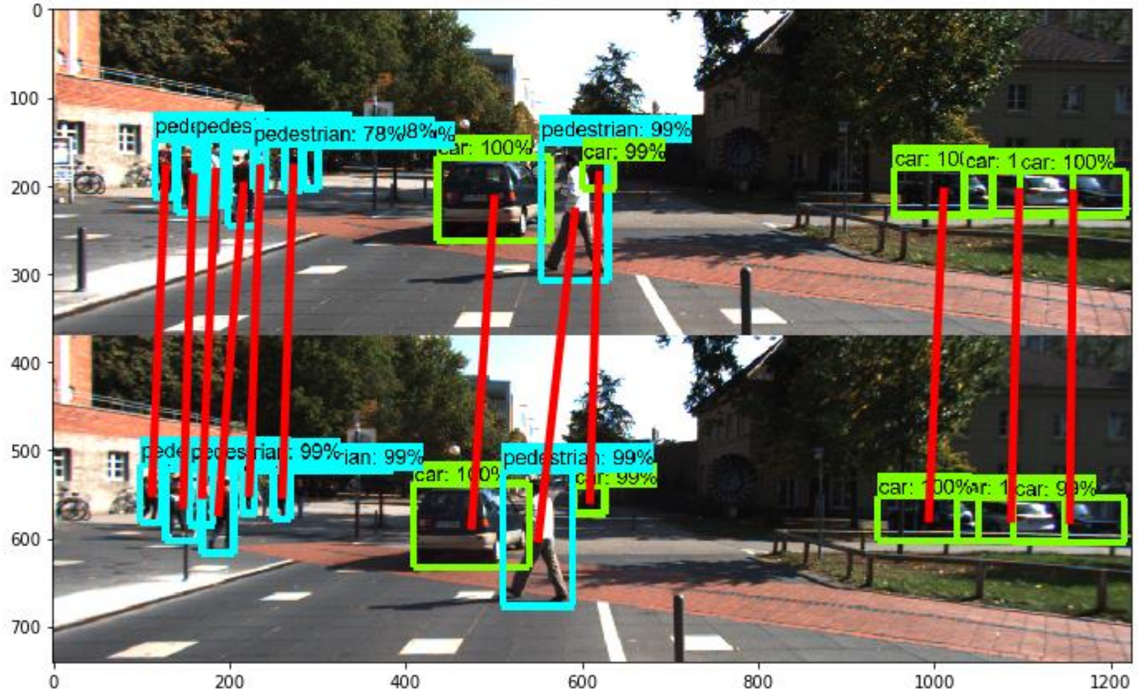




```
dstereo = 18.0
dsize = 21.0
```

Estimating disparity using matched patterns

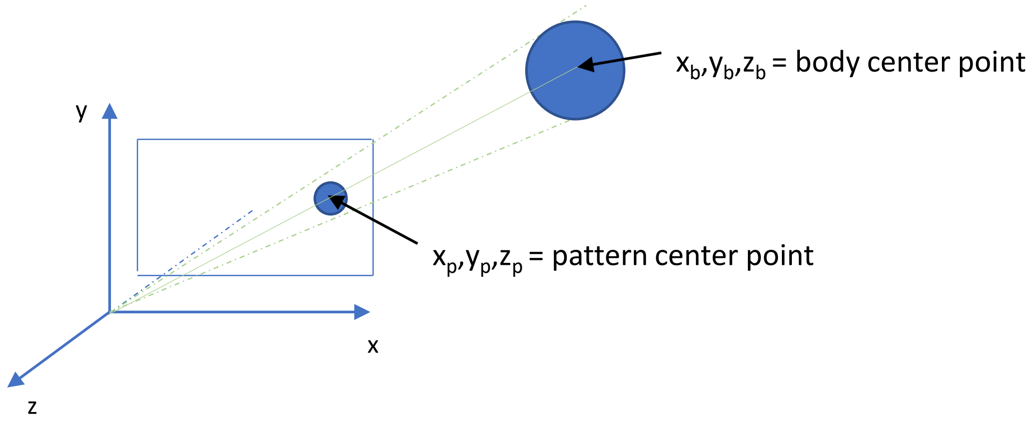
Left camera pattern:
x_min_left
x_max_left



Right camera pattern:
x_min_right
x_max_right

$Pattern\ disparity =$ 0.5 * (x_min_left+x_max_left)-0.5*(x_min_right+x_max_right)

3D projection



x_b, y_b, z_b = body center point

x_p, y_p, z_p = pattern center point

$$(x_b,\ y_b,\ z_b) = t^*\ (x_p,\ y_p,\ z_p)$$

Where:

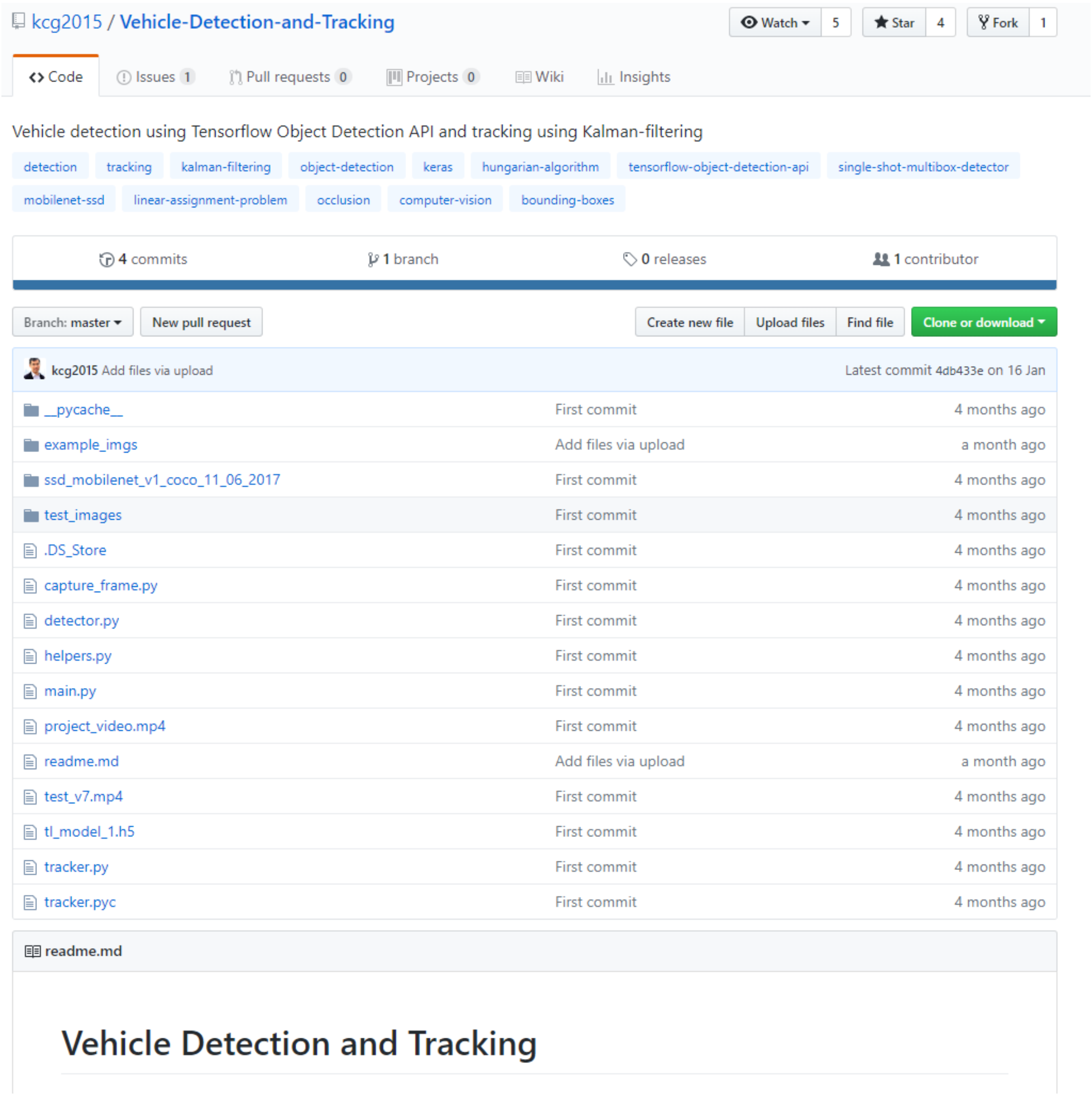
$$(x_p,\ y_p,\ z_p) = \left(-\frac{s_w}{2} + p_x * \frac{s_w}{p_w},\ \ \frac{s_h}{2} - p_y * \frac{s_h}{p_h},\ \ -f\right)$$

$$t = \frac{d}{\sqrt{{x_p}^2 + {y_p}^2 + {z_p}^2}}$$

s_w= *sensor width* (*m*)
s_h= *sensor height* (*m*)
p_w= *image width (pixels)*
p_h= *image height (pixels)*
f = focal length (m)
p_x = *pattern center point location (x, pixels)*
p_y = *pattern center point location (y, pixels)*

Note! Only left image used. Right image is used only for disparity calculation (in the context of distance estimation and 3D projection).

Kalman Filter Parameter Adjustments



Vehicle detection using Tensorflow Object Detection API and tracking using Kalman-filtering

kcg2015 Add files via upload — Latest commit 4db433e on 16 Jan

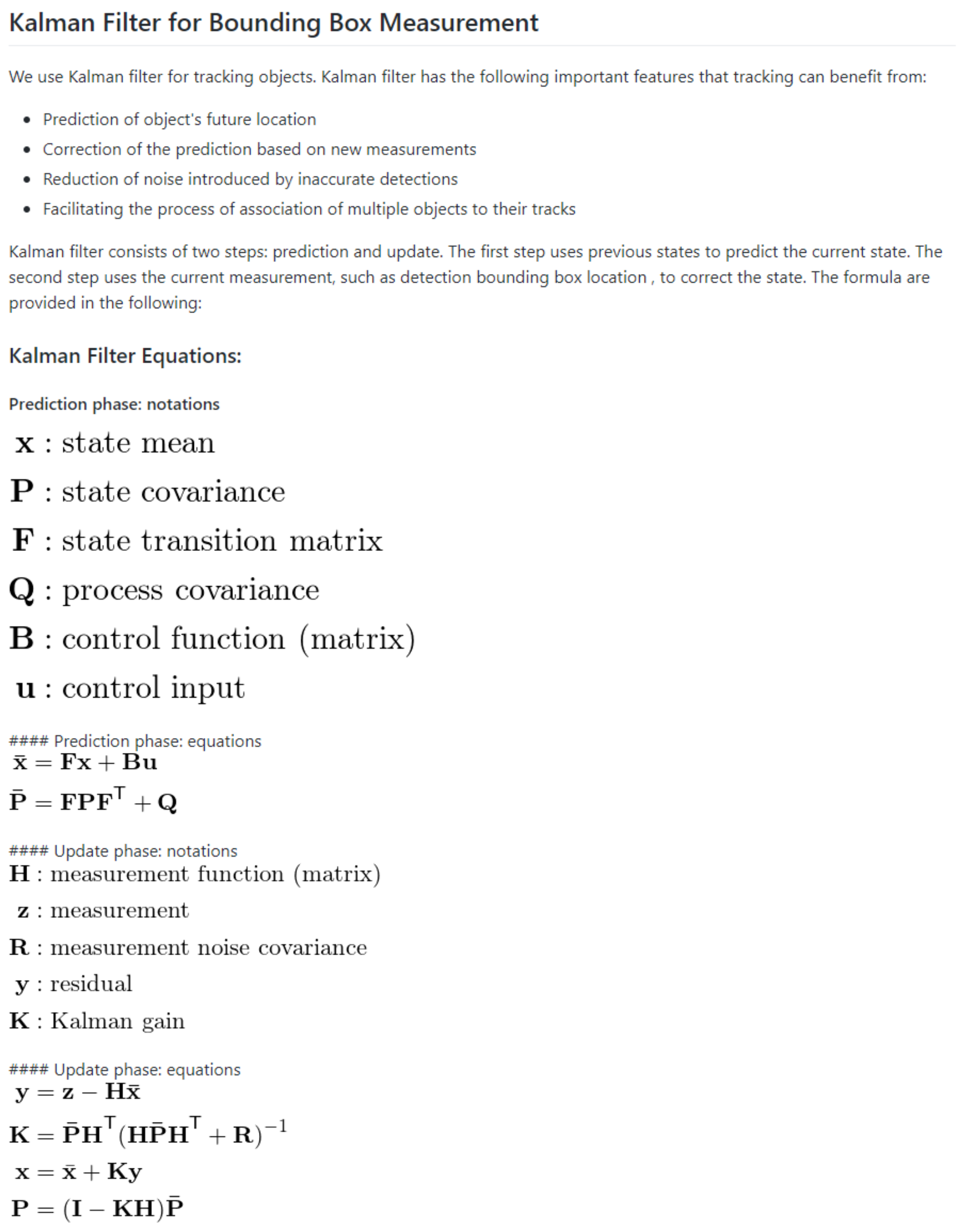
__pycache__	First commit	4 months ago
example_imgs	Add files via upload	a month ago
ssd_mobilenet_v1_coco_11_06_2017	First commit	4 months ago
test_images	First commit	4 months ago
.DS_Store	First commit	4 months ago
capture_frame.py	First commit	4 months ago
detector.py	First commit	4 months ago
helpers.py	First commit	4 months ago
main.py	First commit	4 months ago
project_video.mp4	First commit	4 months ago
readme.md	Add files via upload	a month ago
test_v7.mp4	First commit	4 months ago
tl_model_1.h5	First commit	4 months ago
tracker.py	First commit	4 months ago
tracker.pyc	First commit	4 months ago

readme.md

Vehicle Detection and Tracking

https://github.com/kcg2015/Vehicle-Detection-and-Tracking

Kalman Filter Parameter Adjustments



Kalman Filter for Bounding Box Measurement

We use Kalman filter for tracking objects. Kalman filter has the following important features that tracking can benefit from:

- Prediction of object's future location
- Correction of the prediction based on new measurements
- Reduction of noise introduced by inaccurate detections
- Facilitating the process of association of multiple objects to their tracks

Kalman filter consists of two steps: prediction and update. The first step uses previous states to predict the current state. The second step uses the current measurement, such as detection bounding box location , to correct the state. The formula are provided in the following:

Kalman Filter Equations:

Prediction phase: notations

$\mathbf{x}$: state mean

$\mathbf{P}$: state covariance

$\mathbf{F}$: state transition matrix

$\mathbf{Q}$: process covariance

$\mathbf{B}$: control function (matrix)

$\mathbf{u}$: control input

Prediction phase: equations

$\bar{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{B}\mathbf{u}$

$\bar{\mathbf{P}} = \mathbf{F}\mathbf{P}\mathbf{F}^{\mathsf{T}} + \mathbf{Q}$

Update phase: notations

$\mathbf{H}$: measurement function (matrix)

$\mathbf{z}$: measurement

$\mathbf{R}$: measurement noise covariance

$\mathbf{y}$: residual

$\mathbf{K}$: Kalman gain

Update phase: equations

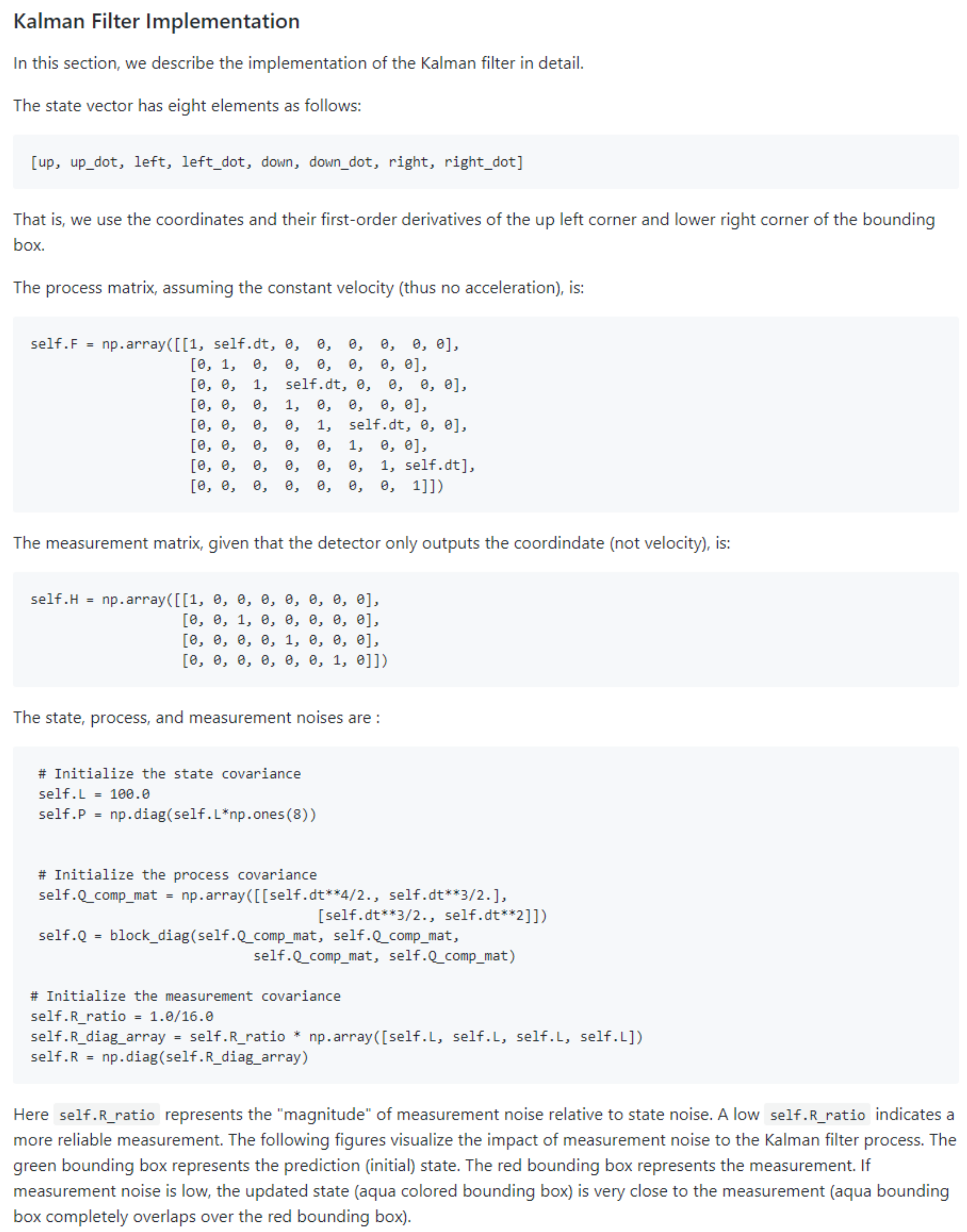
$\mathbf{y} = \mathbf{z} - \mathbf{H}\bar{\mathbf{x}}$

$\mathbf{K} = \bar{\mathbf{P}}\mathbf{H}^{\mathsf{T}}(\mathbf{H}\bar{\mathbf{P}}\mathbf{H}^{\mathsf{T}} + \mathbf{R})^{-1}$

$\mathbf{x} = \bar{\mathbf{x}} + \mathbf{K}\mathbf{y}$

$\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{H})\bar{\mathbf{P}}$

Kalman Filter Parameter Adjustments



Kalman Filter Implementation

In this section, we describe the implementation of the Kalman filter in detail.

The state vector has eight elements as follows:

```
[up, up_dot, left, left_dot, down, down_dot, right, right_dot]
```

That is, we use the coordinates and their first-order derivatives of the up left corner and lower right corner of the bounding box.

The process matrix, assuming the constant velocity (thus no acceleration), is:

```
self.F = np.array([[1, self.dt, 0,  0,  0,  0,  0, 0],
                    [0, 1,  0,  0,  0,  0,  0, 0],
                    [0, 0,  1,  self.dt, 0,  0,  0, 0],
                    [0, 0,  0,  1,  0,  0,  0, 0],
                    [0, 0,  0,  0,  1,  self.dt, 0, 0],
                    [0, 0,  0,  0,  0,  1,  0, 0],
                    [0, 0,  0,  0,  0,  0,  1, self.dt],
                    [0, 0,  0,  0,  0,  0,  0,  1]])
```

The measurement matrix, given that the detector only outputs the coordindate (not velocity), is:

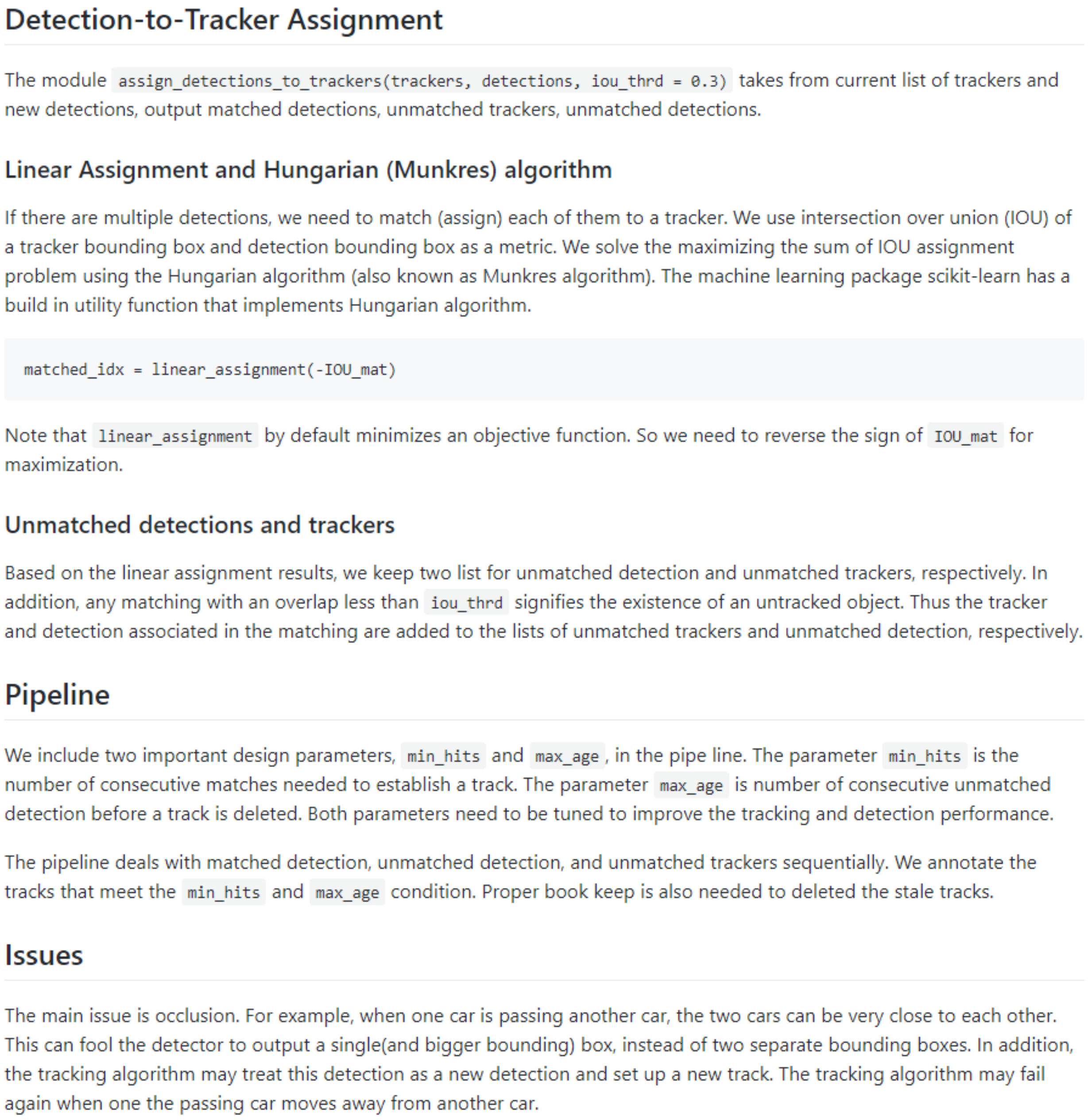
```
self.H = np.array([[1, 0, 0, 0, 0, 0, 0, 0],
                   [0, 0, 1, 0, 0, 0, 0, 0],
                   [0, 0, 0, 0, 1, 0, 0, 0],
                   [0, 0, 0, 0, 0, 0, 1, 0]])
```

The state, process, and measurement noises are :

```
 # Initialize the state covariance
 self.L = 100.0
 self.P = np.diag(self.L*np.ones(8))


 # Initialize the process covariance
 self.Q_comp_mat = np.array([[self.dt**4/2., self.dt**3/2.],
                                    [self.dt**3/2., self.dt**2]])
 self.Q = block_diag(self.Q_comp_mat, self.Q_comp_mat,
                            self.Q_comp_mat, self.Q_comp_mat)

# Initialize the measurement covariance
self.R_ratio = 1.0/16.0
self.R_diag_array = self.R_ratio * np.array([self.L, self.L, self.L, self.L])
self.R = np.diag(self.R_diag_array)
```

Here `self.R_ratio` represents the "magnitude" of measurement noise relative to state noise. A low `self.R_ratio` indicates a more reliable measurement. The following figures visualize the impact of measurement noise to the Kalman filter process. The green bounding box represents the prediction (initial) state. The red bounding box represents the measurement. If measurement noise is low, the updated state (aqua colored bounding box) is very close to the measurement (aqua bounding box completely overlaps over the red bounding box).

Kalman Filter Parameter Adjustments



Detection-to-Tracker Assignment

The module `assign_detections_to_trackers(trackers, detections, iou_thrd = 0.3)` takes from current list of trackers and new detections, output matched detections, unmatched trackers, unmatched detections.

Linear Assignment and Hungarian (Munkres) algorithm

If there are multiple detections, we need to match (assign) each of them to a tracker. We use intersection over union (IOU) of a tracker bounding box and detection bounding box as a metric. We solve the maximizing the sum of IOU assignment problem using the Hungarian algorithm (also known as Munkres algorithm). The machine learning package scikit-learn has a build in utility function that implements Hungarian algorithm.

```
matched_idx = linear_assignment(-IOU_mat)
```

Note that `linear_assignment` by default minimizes an objective function. So we need to reverse the sign of `IOU_mat` for maximization.

Unmatched detections and trackers

Based on the linear assignment results, we keep two list for unmatched detection and unmatched trackers, respectively. In addition, any matching with an overlap less than `iou_thrd` signifies the existence of an untracked object. Thus the tracker and detection associated in the matching are added to the lists of unmatched trackers and unmatched detection, respectively.

Pipeline

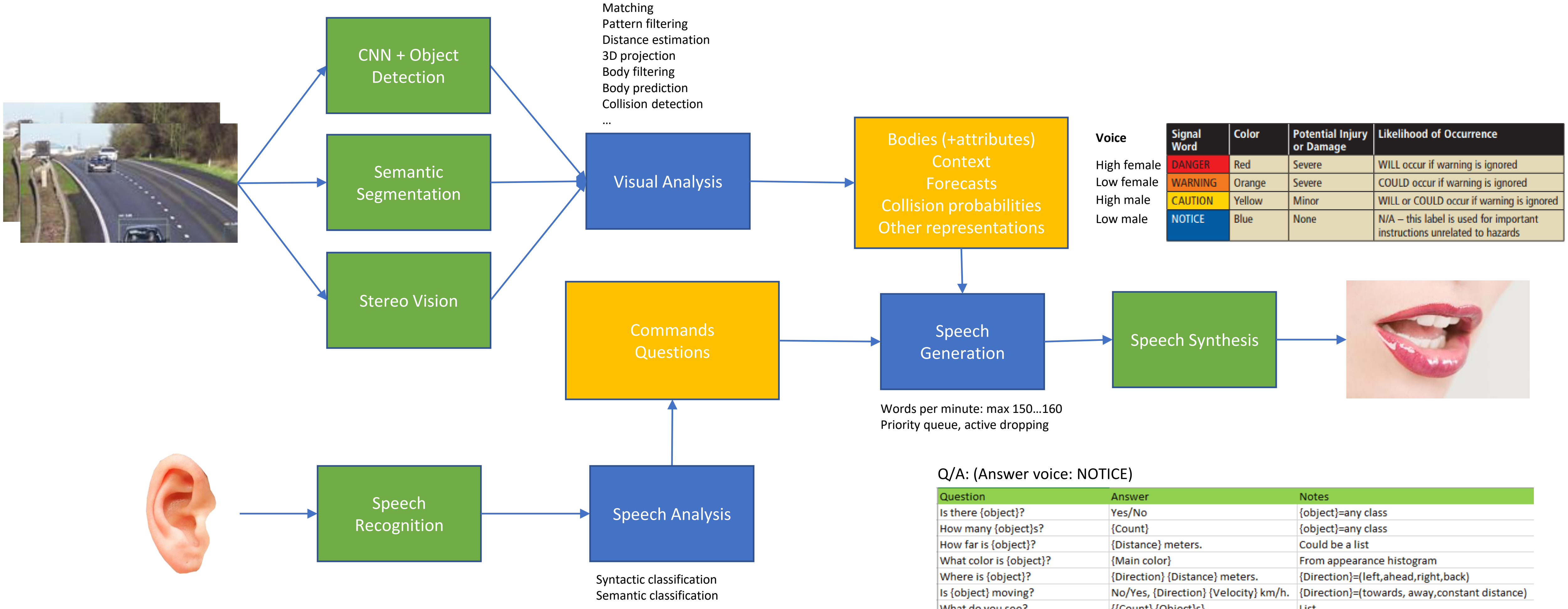
We include two important design parameters, `min_hits` and `max_age`, in the pipe line. The parameter `min_hits` is the number of consecutive matches needed to establish a track. The parameter `max_age` is number of consecutive unmatched detection before a track is deleted. Both parameters need to be tuned to improve the tracking and detection performance.

The pipeline deals with matched detection, unmatched detection, and unmatched trackers sequentially. We annotate the tracks that meet the `min_hits` and `max_age` condition. Proper book keep is also needed to deleted the stale tracks.

Issues

The main issue is occlusion. For example, when one car is passing another car, the two cars can be very close to each other. This can fool the detector to output a single(and bigger bounding) box, instead of two separate bounding boxes. In addition, the tracking algorithm may treat this detection as a new detection and set up a new track. The tracking algorithm may fail again when one the passing car moves away from another car.

The Big Picture



CNN + Object Detection

Semantic Segmentation

Stereo Vision

Matching
Pattern filtering
Distance estimation
3D projection
Body filtering
Body prediction
Collision detection
...

Visual Analysis

Bodies (+attributes)
Context
Forecasts
Collision probabilities
Other representations

Voice	Signal Word	Color	Potential Injury or Damage	Likelihood of Occurrence
High female	DANGER	Red	Severe	WILL occur if warning is ignored
Low female	WARNING	Orange	Severe	COULD occur if warning is ignored
High male	CAUTION	Yellow	Minor	WILL or COULD occur if warning is ignored
Low male	NOTICE	Blue	None	N/A – this label is used for important instructions unrelated to hazards

Commands
Questions

Speech Generation

Words per minute: max 150...160
Priority queue, active dropping

Speech Synthesis

Speech Recognition

Speech Analysis

Syntactic classification
Semantic classification

Q/A: (Answer voice: NOTICE)

Question	Answer	Notes
Is there {object}?	Yes/No	{object}=any class
How many {object}s?	{Count}	{object}=any class
How far is {object}?	{Distance} meters.	Could be a list
What color is {object}?	{Main color}	From appearance histogram
Where is {object}?	{Direction} {Distance} meters.	{Direction}=(left,ahead,right,back)
Is {object} moving?	No/Yes, {Direction} {Velocity} km/h.	{Direction}=(towards, away,constant distance)
What do you see?	{{Count} {Object}s}	List
What is to {Direction} of {Object}?	{Object}	{Direction}=(left,right)
Is {Object} {Direction} {Object}?	Yes/No	{Direction}=(left,right,above,under)
Is {Object} free?	Yes/No	For example chair with/without other objects

Generated based on situation:

Signal word	Sentence	Notes
DANGER	{Object} will collide, move {Direction}.	{Object}=(bicycle,boat,bus,car,cow,horse,motorbike,person,train) {Direction}=(left,right,forward,backward)
WARNING	{Object} might collide, move {Direction}.	{Object}=(bicycle,boat,bus,car,cow,horse,motorbike,person,train) {Direction}=(left,right,forward,backward)
CAUTION	{Object} might collide, move {Direction}.	{Object}=(bird,cat,dog) {Direction}=(left,right,forward,backward)
WARNING	{Object} ahead, turn. Distance {Distance} meters.	{Object}=(chair,dining table,sofa)
NOTICE	{Object} is approaching. Distance {Distance} meters.	{Object}=(bicycle,bird,boat,bus,car,cat,cow,dog,horse,motorbike,person,train)
NOTICE	{Object} is leaving. Distance {Distance} meters.	{Object}=(bicycle,bird,boat,bus,car,cat,cow,dog,horse,motorbike,person,train)
NOTICE	{Caption}	
NOTICE	{Answer}	

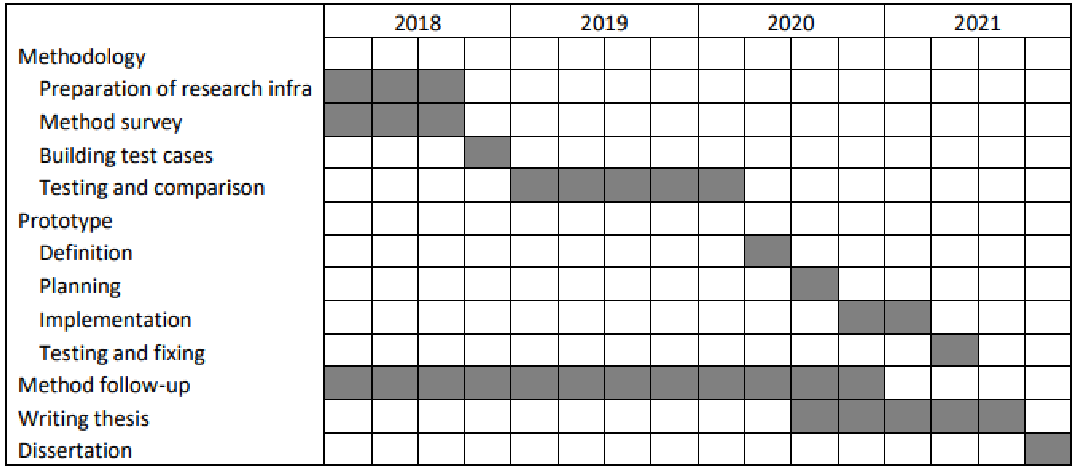
Commands:

Command	Notes
Repeat answer	Answer to previous question is generated until stopped
Stop repeating	Stop answering
Be quiet	Output speech is off
Speak to me	Output speech is on

Next Steps

Next steps

- Kalman filter parameter adjustments (Q1)
- Dataset selection (Q1)
- Stereo vision (Q2)
- Camera yaw, pitch, roll estimation (Q2)
- Speech recognition (Q2)
- Semantic segmentation (Q2)
- Experiments in the wild (Q2)
- Paper (Q3)
- Speech analysis (Q3)
- Speech generation (Q3)
- Use cases (Q4)



	2018				2019				2020				2021			
Methodology																
Preparation of research infra	▨	▨	▨													
Method survey	▨	▨	▨													
Building test cases				▨												
Testing and comparison					▨	▨	▨	▨	▨							
Prototype																
Definition										▨						
Planning											▨					
Implementation												▨	▨			
Testing and fixing														▨		
Method follow-up	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				
Writing thesis											▨	▨	▨	▨	▨	
Dissertation																▨

Discussion

Thank you!

lampola@student.tut.fi
https://github.com/SakariLampola/Thesis