



# Image-Based Situation Awareness Audit 1.3.2018

Sakari Lampola



Previous Audit 11.1.2018

# Previous Audit

---

Open questions:

- Role of classical object tracking algorithms?
- What to do with multiple bounding boxes around one object?
- Appropriate minimum confidence level?
- What to do with false detections inside other objects?
- What to do with false detections from the background?
- How to set Kalman filter parameters for image object filtering?
- Hungarian algorithms special case for hidden objects

To do:

- Close open questions
- Image object status
- Image object velocity estimation
- Probabilistic approach for matching detected and image objects
- 2d -> 3d transformation
- World object state estimation

Other:

- Semantic segmentation
- Organisations to follow: ICCV, ICRA, NIPS, IROS, arXiv
- Camera motion (yaw, pitch)
- Grid or continuous presentation?
- Class specific attributes
- Object history

The slide features a white central area with the text "Project Plan". The background is composed of four geometric shapes: a dark gray triangle in the top-left, a light gray triangle in the top-right, a light gray triangle in the bottom-left, and a yellow triangle in the bottom-right. All triangles are separated by thin white lines.

# Project Plan

# 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

# Method Follow-Up

Computer Vision and Pattern Recognition

Authors and titles for recent submissions

- Fri, 19 Jan 2018
- Thu, 18 Jan 2018
- Wed, 17 Jan 2018
- Tue, 16 Jan 2018
- Mon, 15 Jan 2018

[Total of 54 entries: 1-25 (25/55) 151-75 176-241]  
[showing 25 entries per page: fewer | more | all]

**Fri, 19 Jan 2018**

[1] [arXiv:1801.06104 \[pdf, other\]](#)  
**Invariants of multidimensional time series based on their iterated-integral signature**  
Joscha Diehl, Jeremy Reizenstein  
Subjects: Computer Vision and Pattern Recognition (cs.CV); Representation Theory (math.RT)

[2] [arXiv:1801.06066 \[pdf, other\]](#)  
**RED-Net: A Recurrent Encoder-Decoder Network for Video-based Face Alignment**  
Xi Peng, Rogério S. Feris, Xaboyi Wang, Dimitris N. Metaxas  
Comments: International Journal of Computer Vision, arXiv admin note: text overlap with arXiv:1608.05477  
Subjects: Computer Vision and Pattern Recognition (cs.CV)

[3] [arXiv:1801.05968 \[pdf, other\]](#)  
**3D CNN-based classification using sMRI and MD-DTI images for Alzheimer disease studies**  
Alexander Khvachkov, Karim Adenigbal, Jenny Benois-Pineau, Andrey Krivos, Gwenaelle Catheline  
Subjects: Computer Vision and Pattern Recognition (cs.CV)

[4] [arXiv:1801.05944 \[pdf, other\]](#)  
**PTB-TIR: A Thermal Infrared Pedestrian Tracking Benchmark**  
Qiao Liu, Zhenyu He  
Comments: 10 pages  
Subjects: Computer Vision and Pattern Recognition (cs.CV)

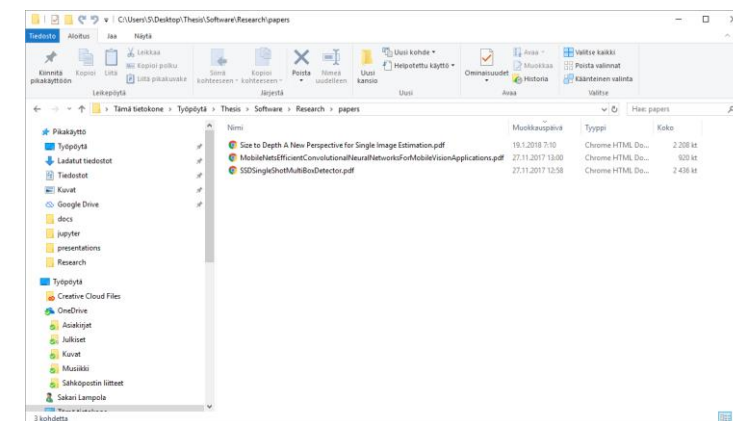
[5] [arXiv:1801.05918 \[pdf\]](#)  
**Extend the shallow part of Single Shot MultiBox Detector via Convolutional Neural Network**  
Liwon Zheng, Canmiao Fu, Yong Zhao  
Comments: 7 pages, 3 figures, 3 tables  
Subjects: Computer Vision and Pattern Recognition (cs.CV)

[6] [arXiv:1801.05912 \[pdf, other\]](#)  
**On the influence of Dice loss function in multi-class organ segmentation of abdominal CT using 3D fully convolutional networks**  
Chen Shen, Holger R. Roth, Hirohisa Oda, Masahiro Oda, Yuichiro Hayashi, Kazunari Misawa, Kensaku Mori  
Comments: presented at MICCAI- November 2017, Takamatsu, Japan (this hep URL)  
Subjects: Computer Vision and Pattern Recognition (cs.CV)

[7] [arXiv:1801.05895 \[pdf, other\]](#)  
**Sparsely Connected Convolutional Networks**  
Ligeng Zhu, Ruizhi Deng, Zhenwei Deng, Greg Mori, Ping Tan  
Subjects: Computer Vision and Pattern Recognition (cs.CV)

**Thu, 18 Jan 2018**

[8] [arXiv:1801.05787 \[pdf, other\]](#)  
**Faster gaze prediction with dense networks and Fisher pruning**  
Lucas Theis, Iryna Korshunova, Aliyhan Tegan, Ferenc Huszar



# Image Object Velocity Estimation

Image object velocity is necessary for:

- predicting image object locations when matching new measurements
- identifying image objects
- predicting image object locations for hidden objects

Image object

- id
- status
- x\_min
- x\_max
- y\_min
- y\_max
- vx\_min
- vx\_max
- vy\_min
- vy\_max
- class
- confidence
- appearance

Estimation algorithm

Image Object Kalman Filtering

Bounding box corner location

State vector  $s$ :

$$s = \begin{bmatrix} l \\ v \end{bmatrix}$$

where

$l$  = location coordinate ( $x_{min}$ ,  $x_{max}$ ,  $y_{min}$ ,  $y_{max}$ ) of the bounding box corner in the image

$v$  = velocity ( $vx_{min}$ ,  $vx_{max}$ ,  $vy_{min}$ ,  $vy_{max}$ ) of the bounding box corner in the image

State equation in differential form:

$$\frac{ds(t)}{dt} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} * s(t) + \epsilon(t) = A_1 * s$$

State equation in difference form:

$$s(k+1) = (I + \Delta * A_1) * s(k) + \epsilon(k)$$

$$= \begin{bmatrix} 1 & \Delta \\ 0 & 1 \end{bmatrix} * s(k) + \epsilon(k) = A * s(k) + \epsilon(k)$$

where  $\Delta$  is the time increment and  $\epsilon$  Gaussian noise with covariance  $R$ .

Measurement equation

$$z(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} * s(k) + \delta(k) = C * s(k) + \delta(k)$$

Where  $\delta$  is Gaussian noise with covariance matrix  $Q$ .

Kalman filter initialization:

$$\mu(0) = \begin{bmatrix} l(0) \\ 0 \end{bmatrix}$$

where  $l(0)$  is the first location measurement.

$$\Sigma(0) = \begin{bmatrix} 10.0 & 0 \\ 0 & 10000.0 \end{bmatrix}$$

where 10.0 and 10000.0 are believed initial error variances of location and velocity.

$$R = \begin{bmatrix} 1.0 & 0 \\ 0 & 1.0 \end{bmatrix}$$

where diagonal elements are believed state equation variances of location and velocity.

$$Q = [10.0]$$

Where 10.0 is the believed measurement variance.

Kalman filter update:

$$\mu_1(k) = A * \mu(k-1)$$

$$\Sigma_1(k) = A * \Sigma(k-1) * A^T + R$$

$$K(k) = \Sigma_1(k) * C^T * (C * \Sigma_1(k) * C^T + Q)^{-1}$$

$$\mu(k) = \mu_1(k) + K(k) * (z(k) - C * \mu_1(k))$$

$$\Sigma(k) = (I - K(k) * C) * \Sigma_1(k)$$

Asiakirjan loppu ■

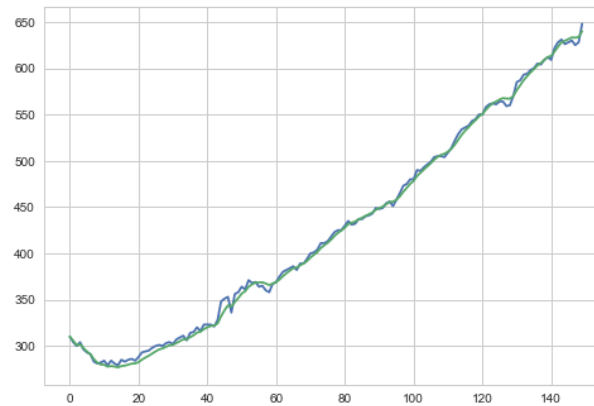
Numerical values are estimated using grid search and 10 step ahead mean prediction error. Values rounded. Later adjusted by experiments.



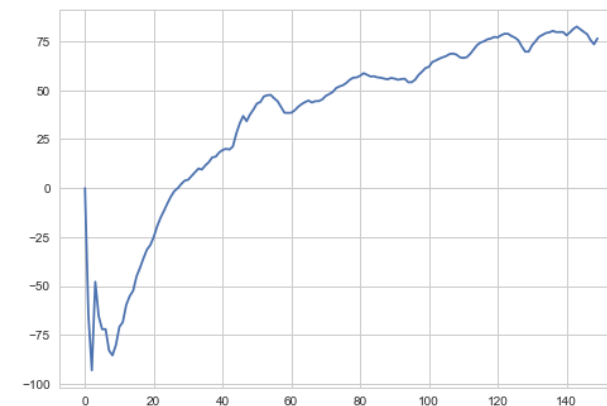
# Image Object Velocity Estimation

---

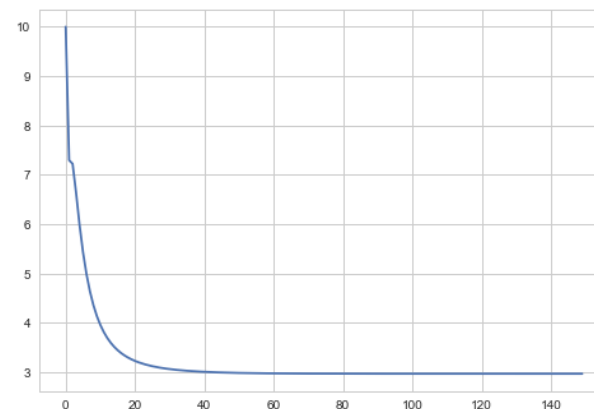
## Moving object (car)



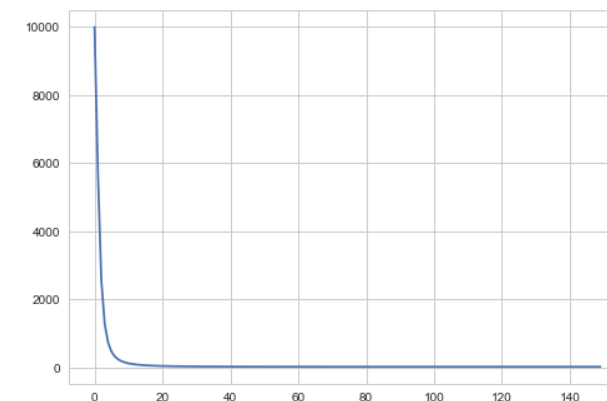
Measured and filtered location (upper left corner)



Estimated velocity



Location variance

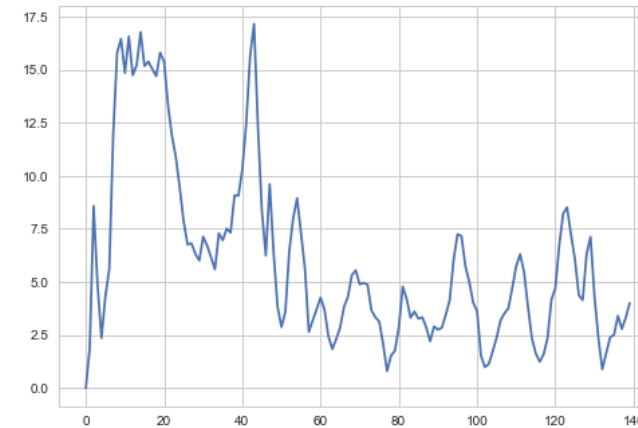
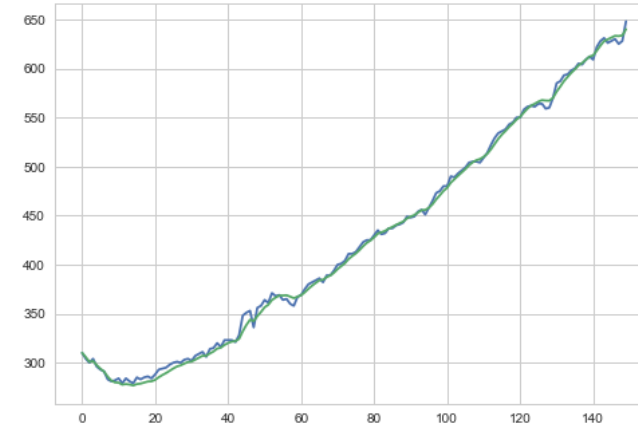


Velocity variance

# Image Object Velocity Estimation

---

Moving object (car)

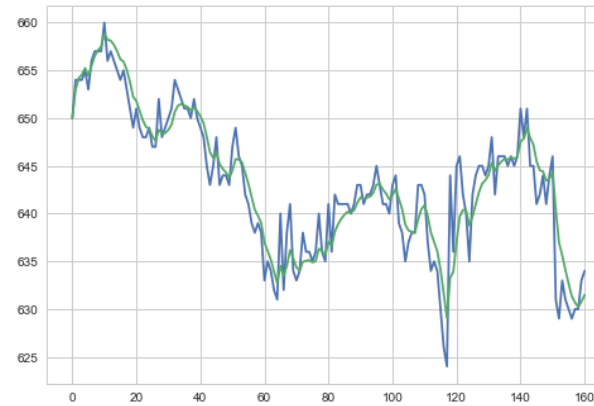


10 step ahead mean prediction error

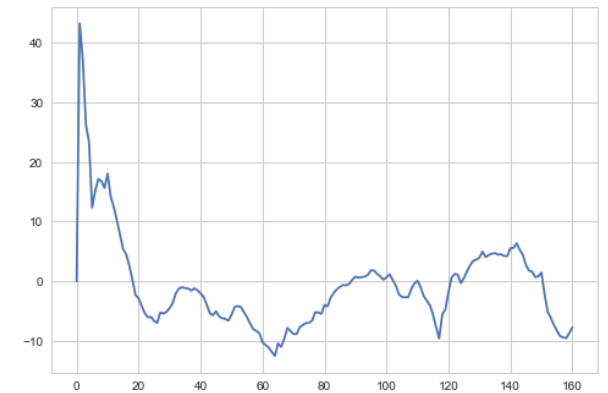
# Image Object Velocity Estimation

---

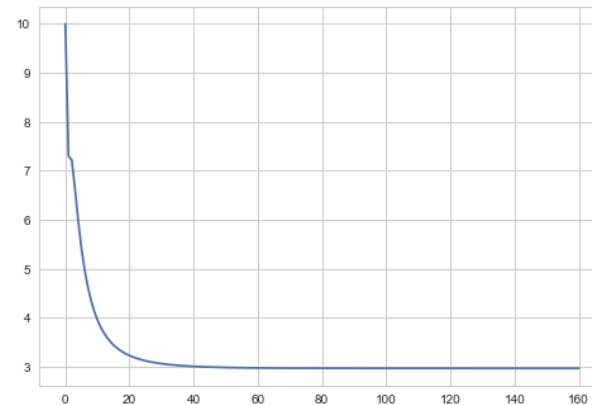
## Static object (calf)



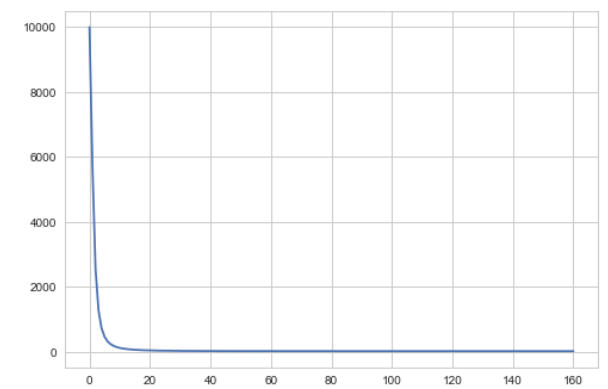
Measured and filtered location (upper left corner)



Estimated velocity



Location variance

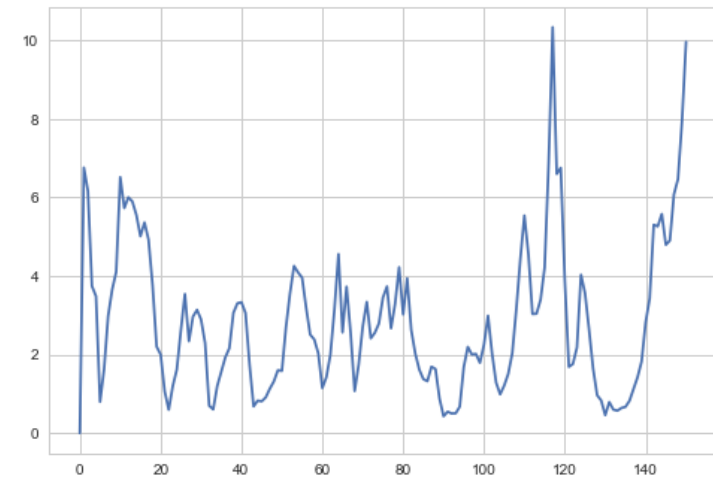
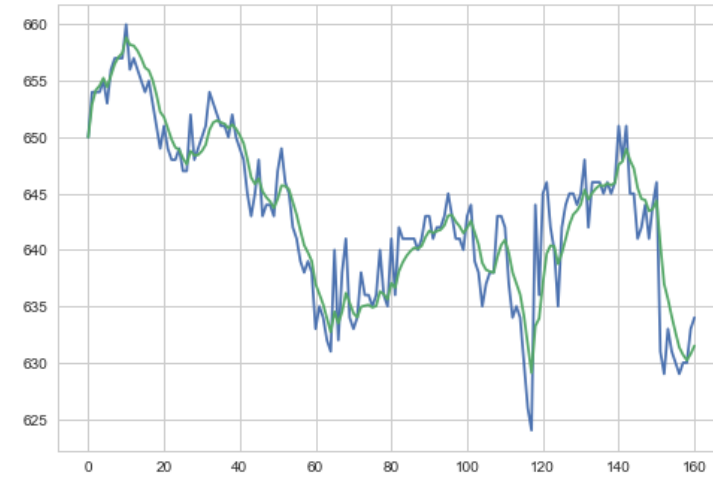


Velocity variance

# Image Object Velocity Estimation

---

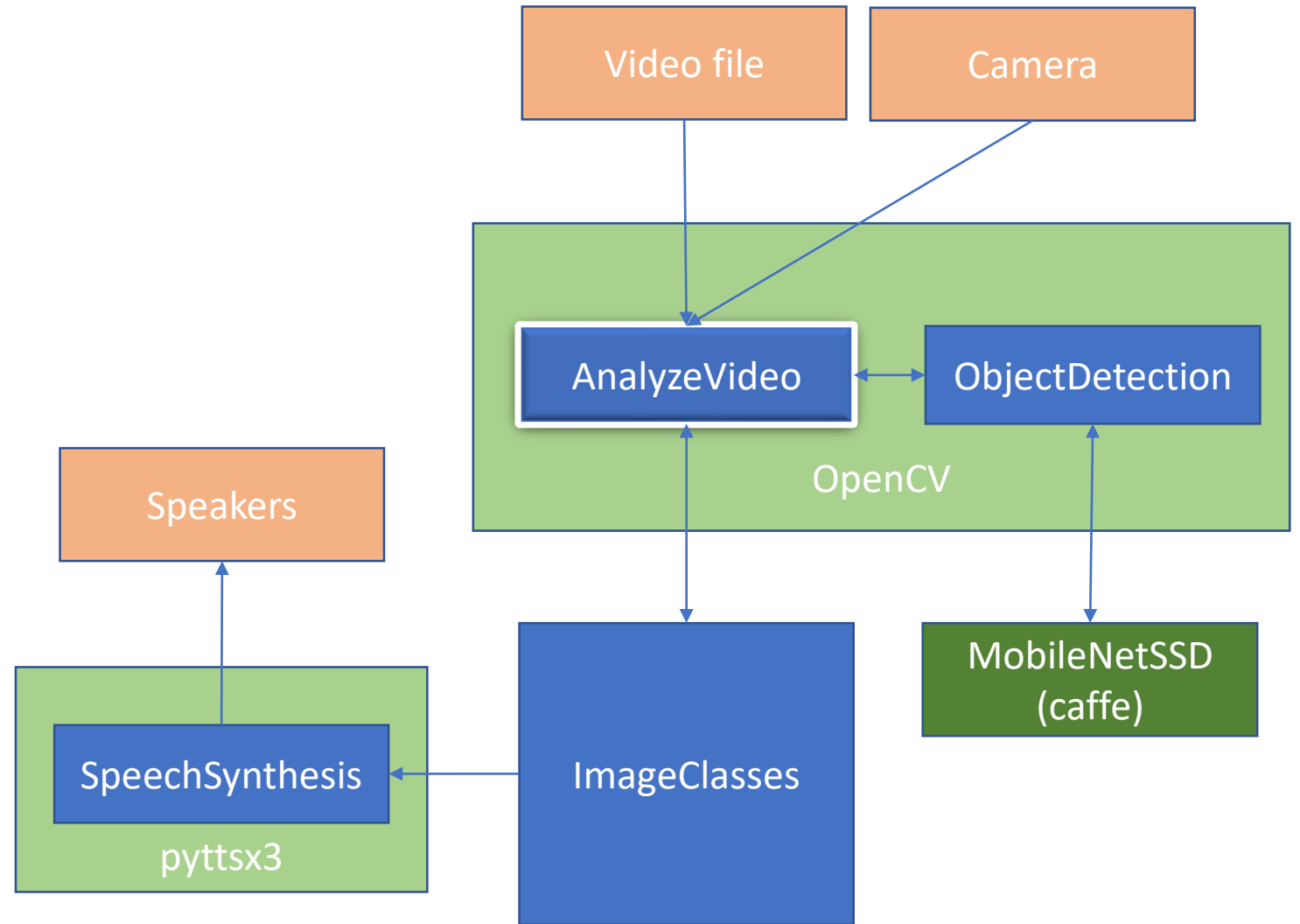
Static object (calf)



10 step ahead mean prediction error

# Speech Synthesis

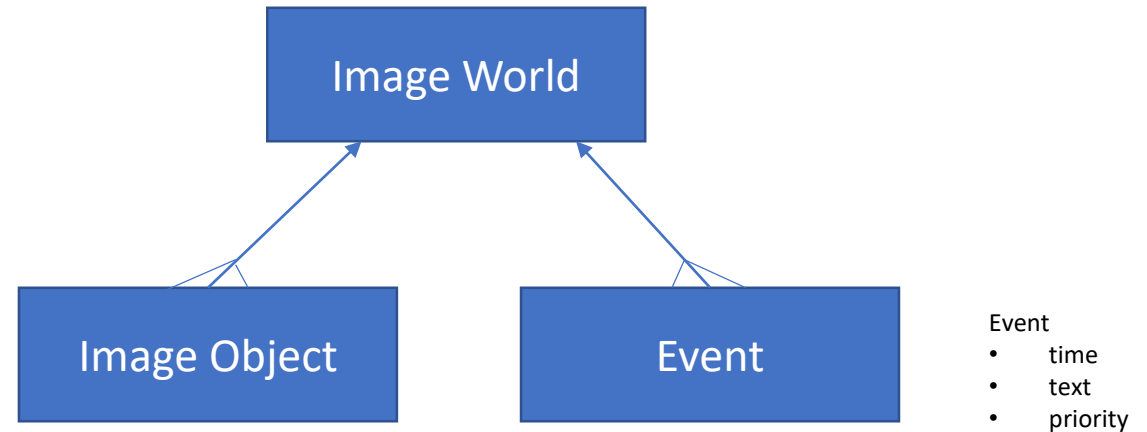
## Software Architecture



# Speech Synthesis

---

## Entities



- Event is generated when
  - new image object is created
  - image object status is changed
- Event will pause the video for the duration of speech (not in the final version)
- Events are collected (history)

# Confidence Level

---

SSD Mobilenet implementation:

# extract the confidence (i.e., probability) associated with the prediction

	A	B	C	D	E	F	G	H	I	J
1	Objects detected		Confidence level							
2	Video	Correct	0,00	0,20	0,40	0,60	0,80	0,90	0,95	1,00
3	CarsOnHighway001.mpg	39	49	49	39	36	34	32	32	0
4	Calf-2679.mp4	1	2	2	2	2	1	1	1	0
5	Dunes-7238.mp4	1	7	7	6	5	2	2	2	0
6	Sofa-11294.mp4	1	2	2	1	1	1	1	1	0
7	Cars133.mp4	5	9	9	6	5	5	5	5	0
8	BlueTit2975.mp4	1	3	3	2	1	1	1	1	0
9	Railway-4106.mp4	1	10	10	5	3	3	1	1	0
10	Hiker1010.mp4	1	4	4	0	0	0	0	0	0
11	Cat-3740.mp4	1	3	3	2	2	1	1	1	0
12	SailingBoat6415.mp4	1	1	1	1	1	1	1	1	0
13	AWomanStandsOnTheSeashore-10058.mp4	1	1	1	1	1	1	1	1	0
14	Dog-4028.mp4	1	4	4	2	1	1	1	1	0
15	Boat-10876.mp4	1	2	2	1	1	1	1	0	0
16	Horse-2980.mp4	1	3	3	3	2	2	1	1	0
17	Sheep-12727.mp4	1	1	1	1	1	1	1	1	1

Good value for creating a new image object is between 0.8 and 0.9.

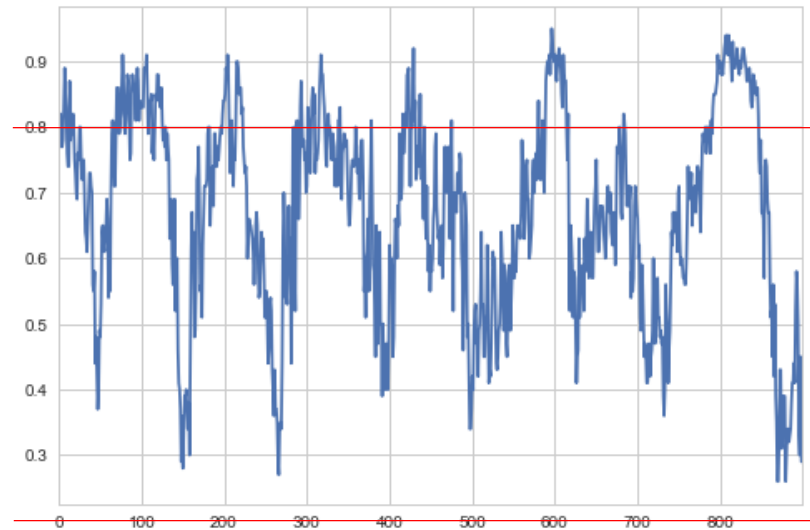
The 'good' value also depends on other hyperparameters.

# Confidence Level

---



Confidence level has dynamics



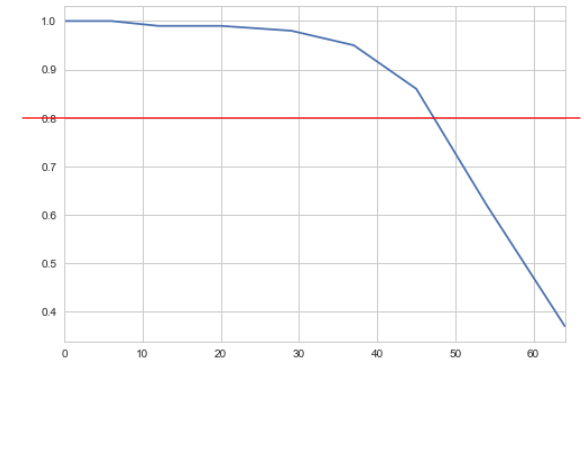
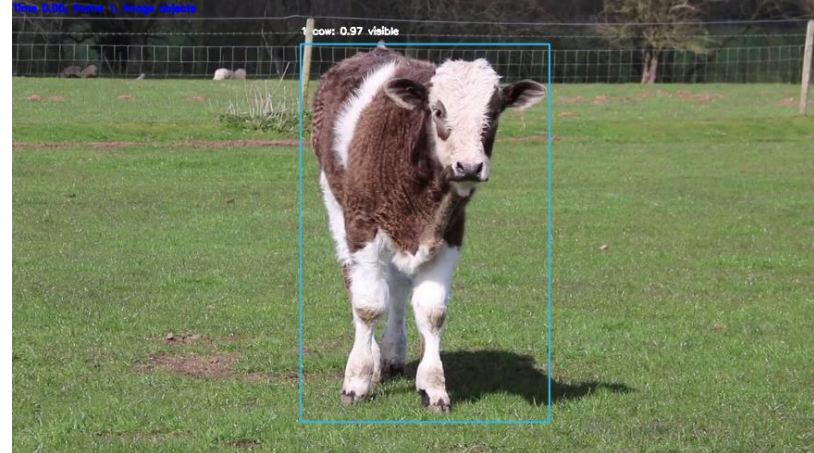
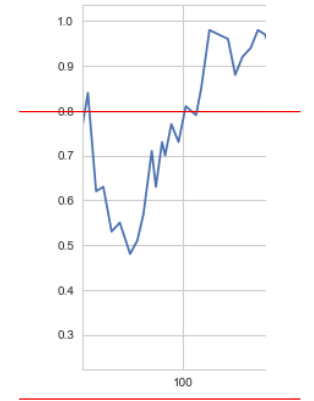
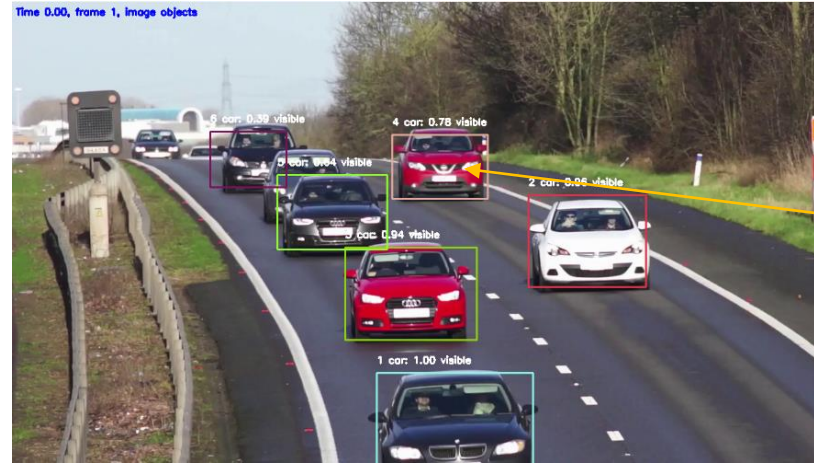
Different levels for creating and updating image object. Hyperparameters:

- CONFIDENCE\_LEVEL\_CREATE (0.8)
- CONFIDENCE\_LEVEL\_UPDATE (0.2)

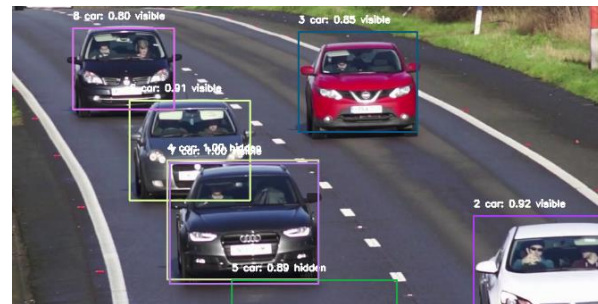


# Confidence Level

---



# Border Behaviour



Box size and form distorted

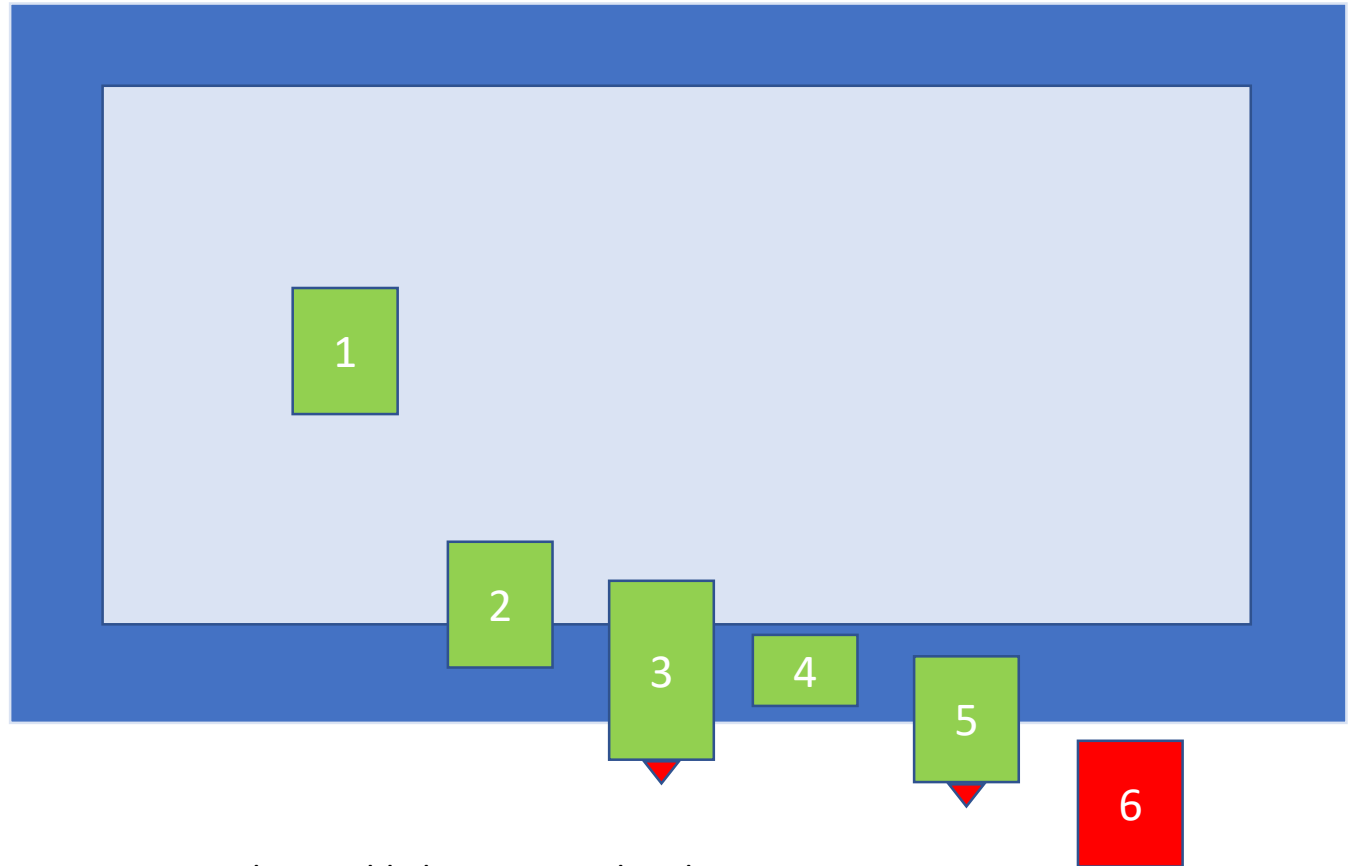
	x_max_c	x_max_m	x_max_p	y_max_c	y_max_m	y_max_p
time						
1.48	1208.859	1209.0	1205.616	646.300	652.0	640.731
1.52	1221.500	1236.0	1212.044	653.697	656.0	649.501
1.56	1232.488	1242.0	1224.941	660.427	661.0	656.939
1.60	1241.599	1246.0	1236.095	668.758	673.0	663.679
1.64	1251.081	1256.0	1245.282	677.391	682.0	672.083
1.68	1258.430	1258.0	1254.848	687.143	694.0	680.794
1.72	1265.965	1266.0	1262.190	694.428	695.0	690.663
1.76	1272.740	1271.0	1269.725	704.340	711.0	697.956
1.80	1280.741	1282.0	1276.471	711.433	711.0	707.979
1.84	1287.573	1286.0	1284.493	717.291	714.0	715.066
1.88	1292.323	1286.0	1291.299	722.517	718.0	720.869
1.92	1292.517	1276.0	1295.946	728.172	725.0	726.022
1.96	1291.385	1273.0	1295.873	731.168	722.0	731.626
2.00	1291.974	1279.0	1294.445	732.465	720.0	734.474
2.04	1291.500	1277.0	1294.826	732.500	718.0	735.572
2.08	1290.547	1276.0	1294.121	733.994	724.0	735.375
2.12	1289.259	1275.0	1292.938	736.016	728.0	736.711
2.16	1289.533	1280.0	1291.424	736.959	727.0	738.606
2.20	1290.113	1282.0	1291.548	737.402	727.0	739.392
2.24	1290.640	1283.0	1292.000	735.994	722.0	739.671

Hyperparameter BORDER\_WIDTH (30)

In [10]: # image size 1280 \* 720

# Border Behaviour

---



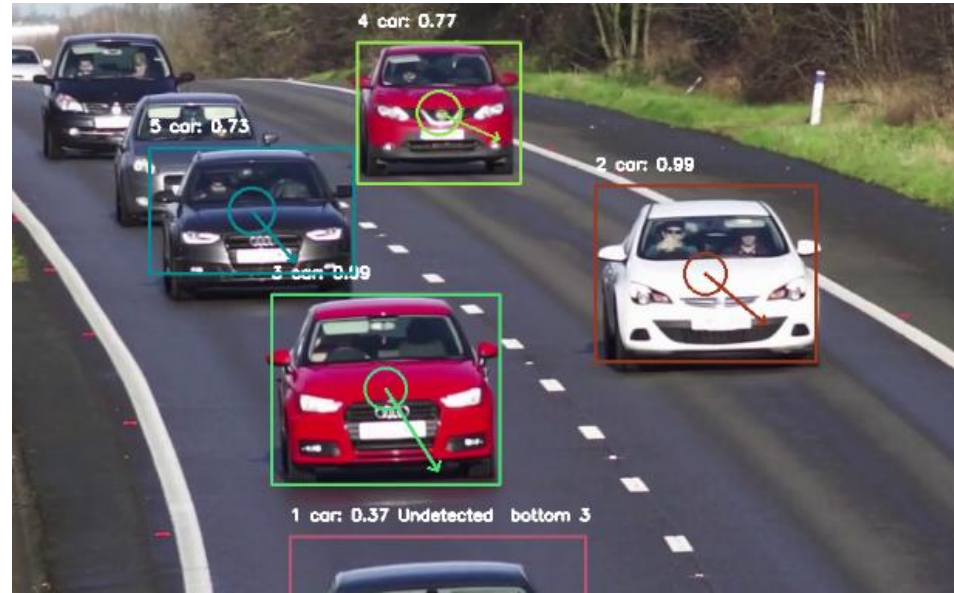
- Type 3 and 5: world object not updated
- Type 6: removed, world object acceleration fixed
- If an object touches 3 borders, it is removed

Done for:

- left
- right
- top
- bottom

# Visual Presentation

---



- Ellipse axes proportional to the standard deviation of the location ( $2 \times \text{std}$ , corresponding to 95% probability)
- Arrow direction and length proportional to velocity (measured in pixels/second)

# Object Retention

---

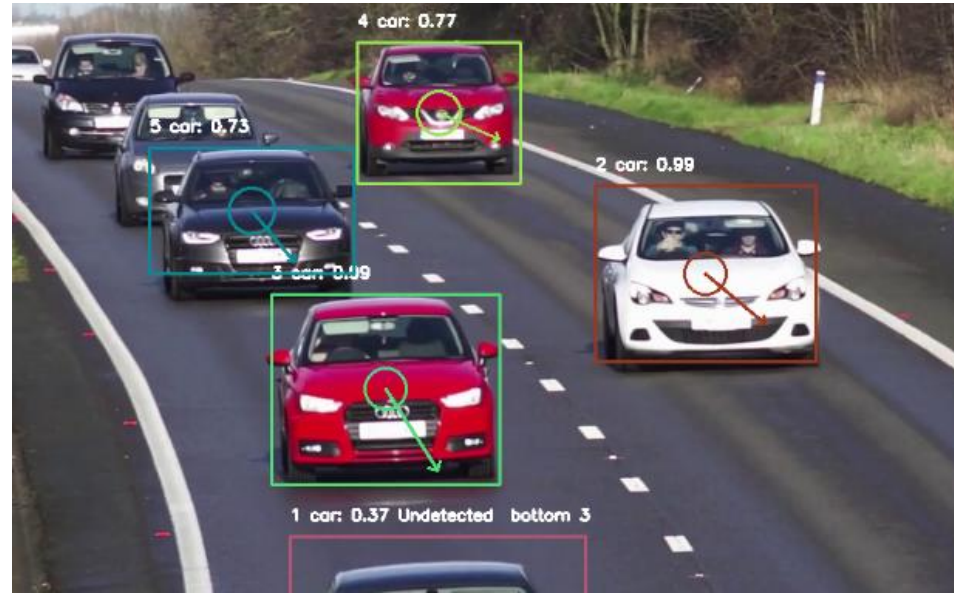


Image objects are removed if not detected in RETENTION\_COUNT\_MAX (30) successive frames.

Diagram illustrating the formation of a virtual image by a thin lens. The object AB has height  $h_i$  and is located at a distance  $f$  from the lens. The virtual image DC has height  $h$  and is located at a distance  $d$  from the lens. The focal length is  $f$ , and the angle of observation is  $\alpha$ . The distance from the object to the image is  $AF = d$  (distance).

$$\frac{0.5 * h_i}{0.5 * h} = \frac{AG}{d} = \frac{\frac{f}{\cos(\alpha)}}{d} = \frac{f}{d * \cos(\alpha)}$$

Similar equations for horizontal direction  
( $\beta$ =azimuth)

# Distance Estimation

$$d = \frac{f * h}{\cos(\alpha) * \cos(\beta) * h_i} = \frac{f * h}{\cos(\alpha) * \cos(\beta) * h_i * s_h / p_h}$$

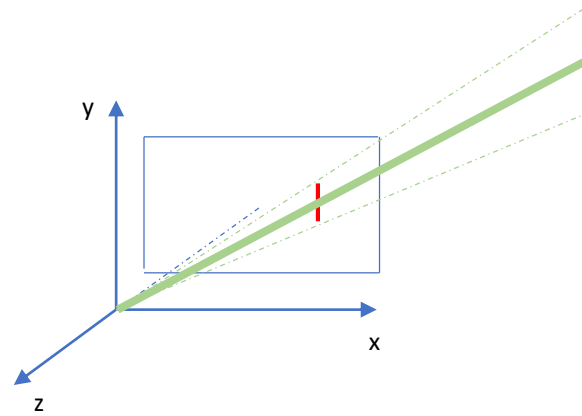
$s_w$  = sensor width (m)  
 $s_h$  = sensor height (m)  
 $p_w$  = image width (pixels)  
 $p_h$  = image height (pixels)  
 $h_i$  = object height (pixels)  
 $h$  = object height (m)  
 $f$  = focal length (m)  
 $\alpha$  = altitude (rad)  
 $\beta$  = azimuth (rad)

Example (Nikon D800E):

$s_w$  = sensor width (m) = 0.0359 m  
 $s_h$  = sensor height (m) = 0.0240 m  
 $p_w$  = image width (pixels) = 7360  
 $p_h$  = image height (pixels) = 4912  
 $h_i$  = object height (pixels) = 100  
 $h$  = object height (m) = 1.0 m  
 $f$  = focal length (m) = 0.050 m  
 $\alpha$  = altitude (rad) = 0.0  
 $\beta$  = azimuth (rad) = 0.0

$$d = \frac{0.050m * 1m}{1.0 * 1.0 * 100 * 0.024m / 4912} = 102.33 m$$

# 2d -> 3d Transformation



From pixel coordinates (sensor plane) to 3d camera coordinates:

$$(x_c, y_c, z_c) = \left( -\frac{s_w}{2} + xp * \frac{s_w}{p_w}, \frac{s_h}{2} - yp * \frac{s_h}{p_h}, -f \right)$$

Object center will be on the line:

$$(x_o, y_o, z_o) = t * (x_c, y_c, z_c)$$

The length of the line is:

$$d = \frac{f * h}{\cos(\alpha) * \cos(\beta) * h_i * s_h / p_h}$$

$$\alpha = \arctan(y_c / f)$$

$$\beta = \arctan(x_c / f)$$

$s_w$  = sensor width (m)  
 $s_h$  = sensor height (m)  
 $p_w$  = image width (pixels)  
 $p_h$  = image height (pixels)  
 $h_i$  = object height (pixels)  
 $h$  = object height (m)  
 $f$  = focal length (m)  
 $\alpha$  = altitude (rad)



# 2d -> 3d Transformation

---

$$t^2 * (x_c^2 + y_c^2 + z_c^2) = d^2$$

$$t = \frac{d}{\sqrt{x_c^2 + y_c^2 + z_c^2}}$$

# 2d -> 3d Transformation

Example:

$s_w$  = sensor width (m) = 0.0359 m

$s_h$  = sensor height (m) = 0.0240 m

$p_w$  = image width (pixels) = 7360

$p_h$  = image height (pixels) = 4912

$h_i$  = object height (pixels) = 100

$h$  = object height (m) = 1.0 m

$f$  = focal length (m) = 0.050 m

$x_p$  = 1200

$y_p$  = 2000



$$(x_c, y_c, z_c) = \left(-\frac{s_w}{2} + x_p * \frac{s_w}{p_w}, \frac{s_h}{2} - y_p * \frac{s_h}{p_h}, -f\right)$$

$$= \left(-\frac{0.0359}{2} + 1200 * \frac{0.0359}{7360}, \frac{0.0240}{2} - 2000 * \frac{0.0240}{4912}, -0.050\right) = (-0.0121, 0.0022, -0.0500)$$

$$\alpha = \arctan(y_c/f) = 0.0445 \quad \beta = \arctan(x_c/f) = -0.2374$$

$$d = \frac{f * h}{\cos(\alpha) * \cos(\beta) * h_i * s_h/p_h}$$

$$= \frac{0.050 * 1}{\cos(0.0445) * \cos(-0.2374) * 100 * 0.0240/4912} = 105.39$$

$$t = \frac{105.39}{\sqrt{-0.0121^2 + 0.0022^2 + -0.0500^2}} = 2.0468e+03$$

# 2d -> 3d Transformation

---

Object location in 3d camera coordinates:

$$(x_o, y_o, z_o) = t^* (x_c, y_c, z_c)$$

$$= 2.0468e+03 * (-0.0121, 0.0022, -0.0500)$$

$$(-24.7593, 4.5602, -102.3389)$$



Open questions:

- Derivation ok?
- Assumptions ok?
  - Optical axis in sensor center?

# 2d -> 3d Transformation

---

Parameters:

$s_w$  = sensor width (m)

$s_h$  = sensor height (m)

$p_w$  = image width (pixels)

$p_h$  = image height (pixels)

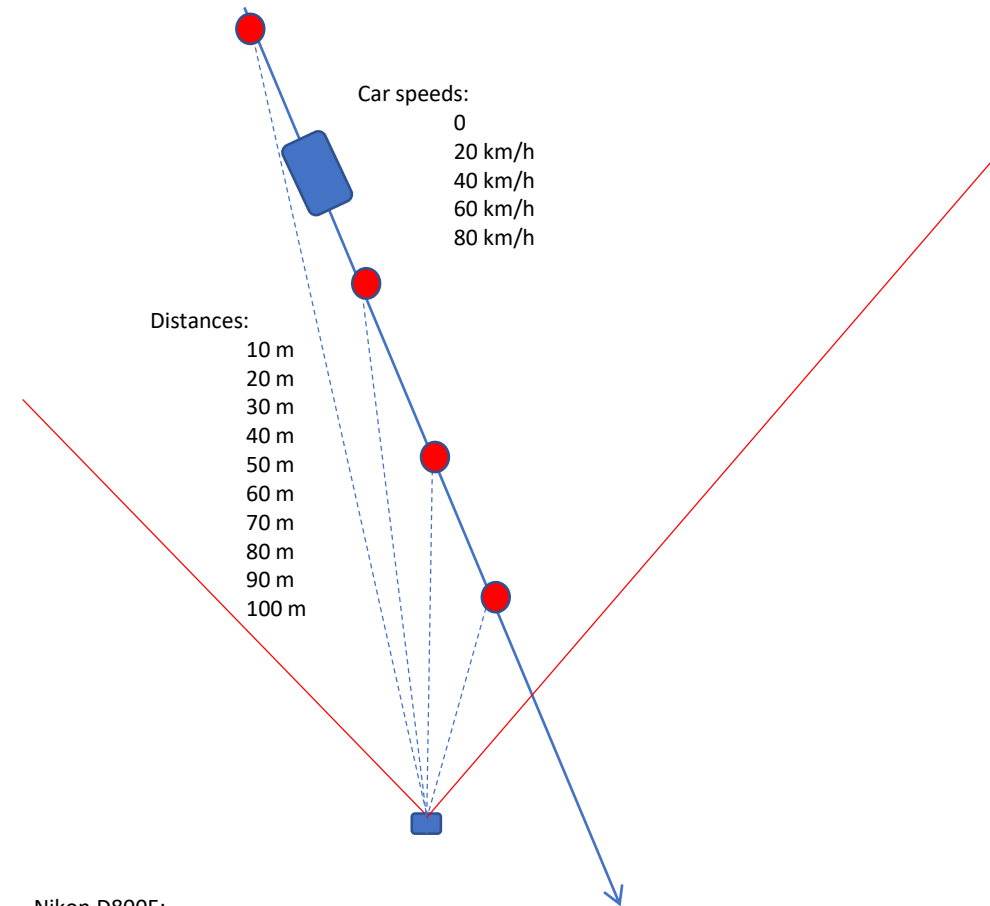
$f$  = focal length (m)

Open questions:

- Video metadata often lacks sensor and focal parameters
- Focal length can change during shooting (zooming)

## Experiment in the wild

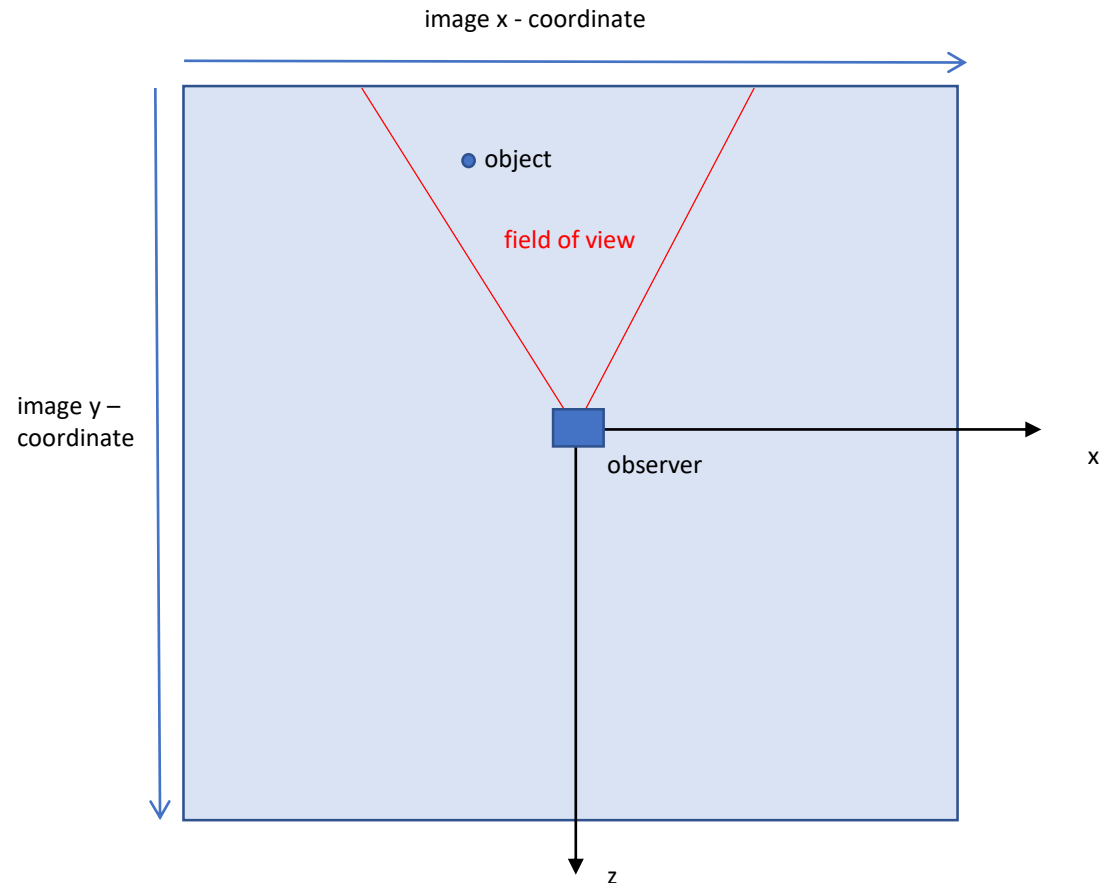
# 2d -> 3d Transformation



Nikon D800E:

$s_w$  = sensor width (m) = 0.0359 m  
 $s_h$  = sensor height (m) = 0.0240 m  
 $p_w$  = image width (pixels) = 7360  
 $p_h$  = image height (pixels) = 4912  
 $h$  = object height (m) = 1.5 m  
 $f$  = focal length (m) = 0.050 m  
 $fov = 2 * \text{atan}\left(\frac{s_w}{2*f}\right) = 39.49^\circ$

# Map Presentation



$x_w$  = object world  $x$  coordinate (m)  
 $z_w$  = object world  $z$  coordinate (m)  
 $x_i$  = object image  $x$  coordinate (pixel)  
 $y_i$  = object image  $y$  coordinate (pixel)  
 $h_w$  = image area world height (m)  
 $h_i$  = image area image height (pixels)  
 $w_w$  = image area world width (m)  
 $w_i$  = image area image width (pixels)

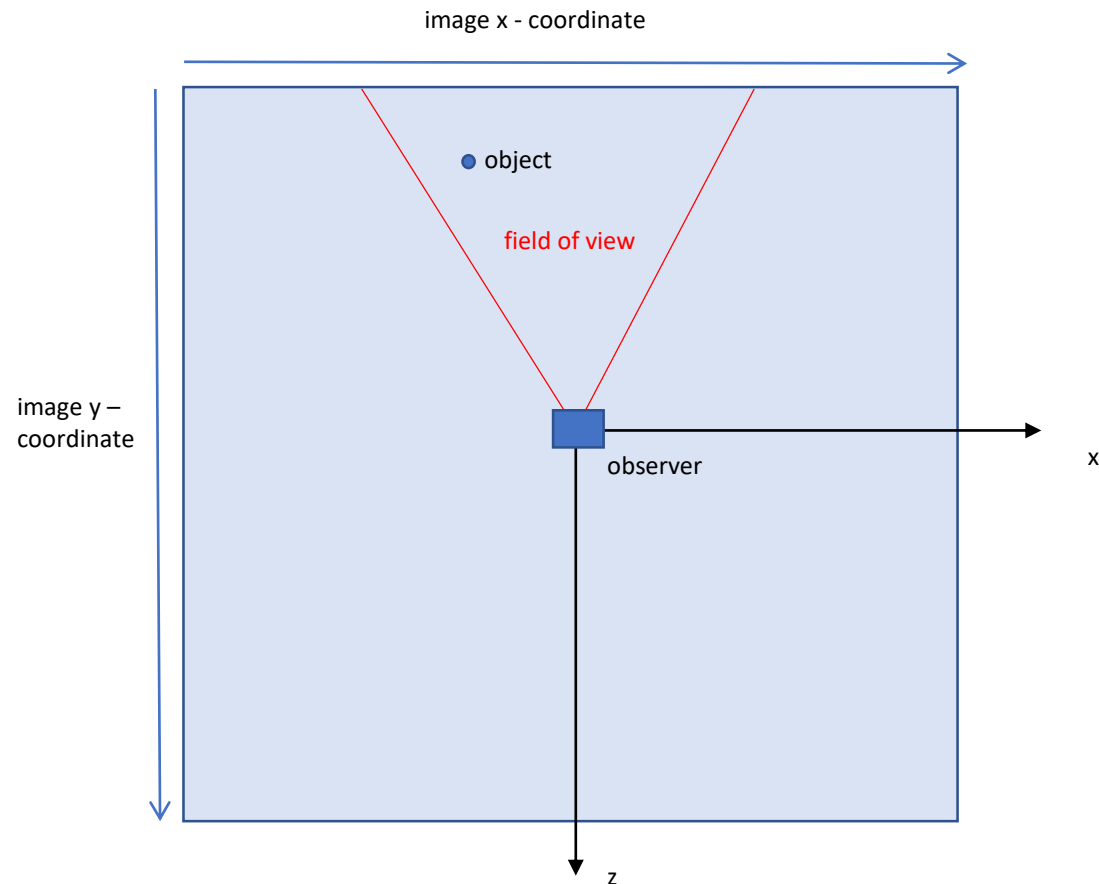
$$(x_i, y_i) = \left( \frac{w_i}{2} + x_w * \frac{w_i}{w_w}, \frac{h_i}{2} + z_w * \frac{h_i}{h_w} \right)$$

$$\frac{w_i}{w_w} = \frac{h_i}{h_w} = p \text{ (pixel/meter ratio)}$$

$$(x_i, y_i) = \left( \frac{w_i}{2} + x_w * p, \frac{h_i}{2} + z_w * p \right)$$

# Map Presentation

---

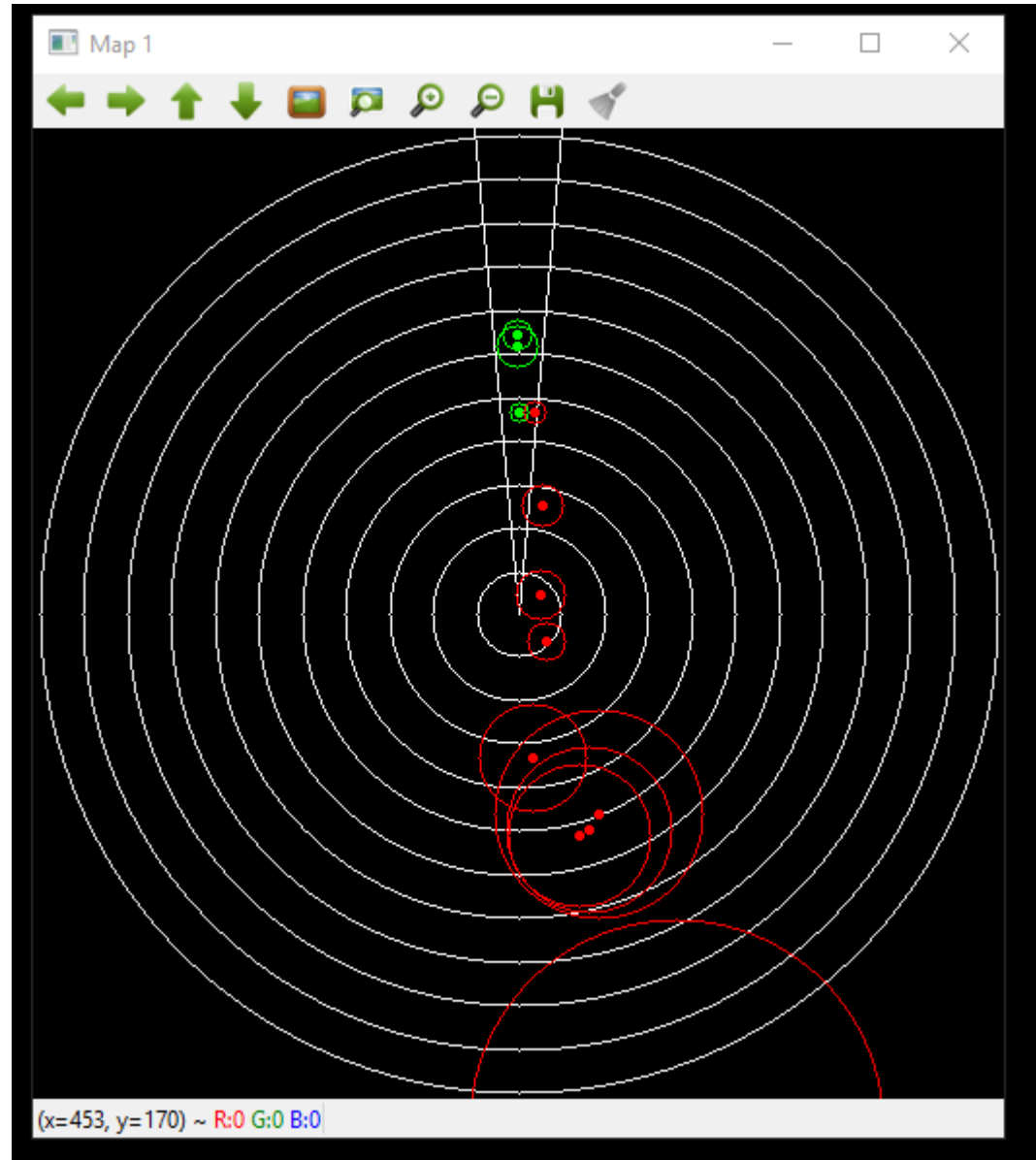


$$FOV = 2 * \operatorname{atan}\left(\frac{s_w}{2 * f}\right)$$

$s_w$  = sensor width (m)  
 $f$  = focal length (m)

# Map Presentation

---

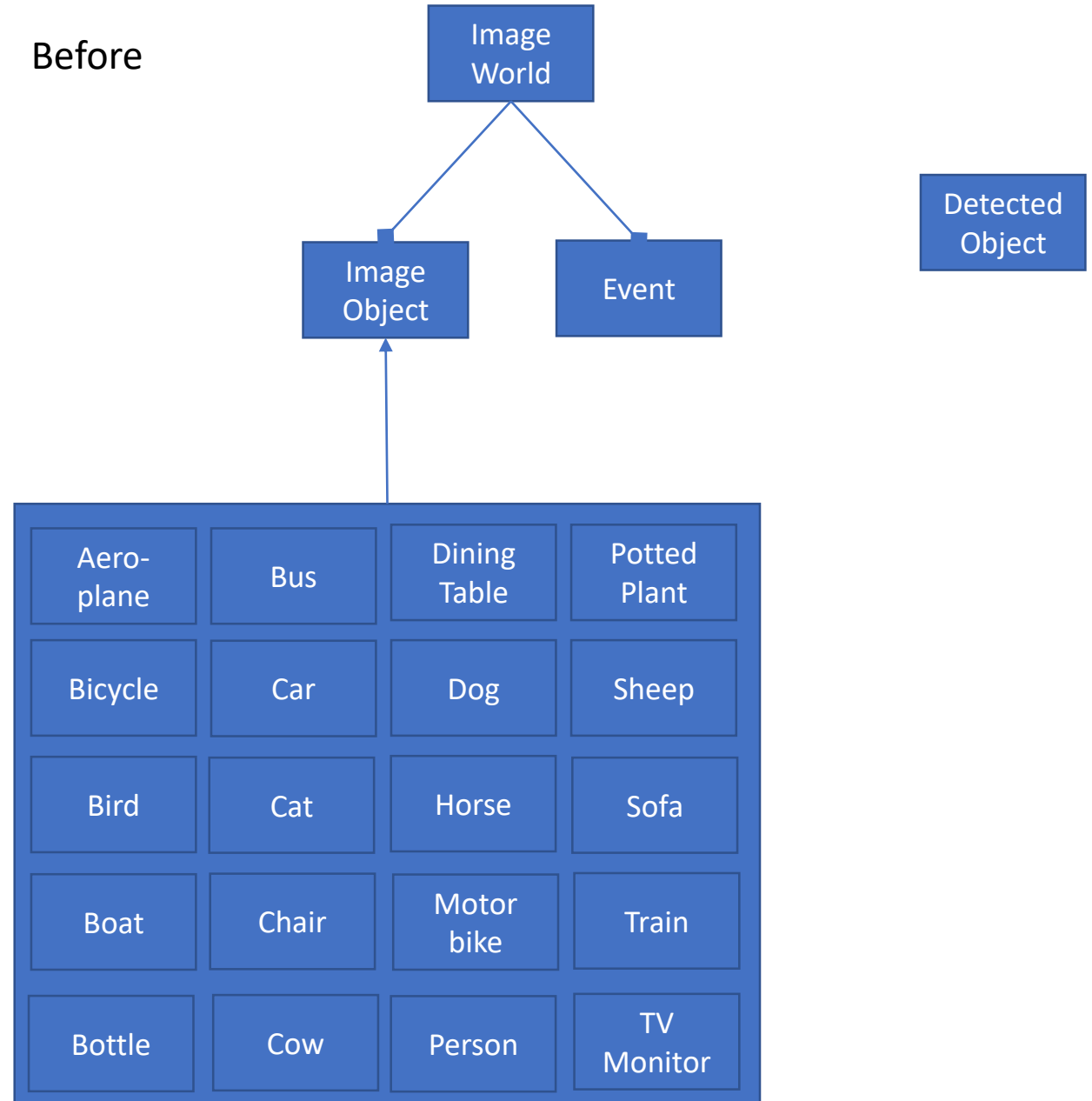




# Entity Diagram

---

Before



# Entity Diagram

---

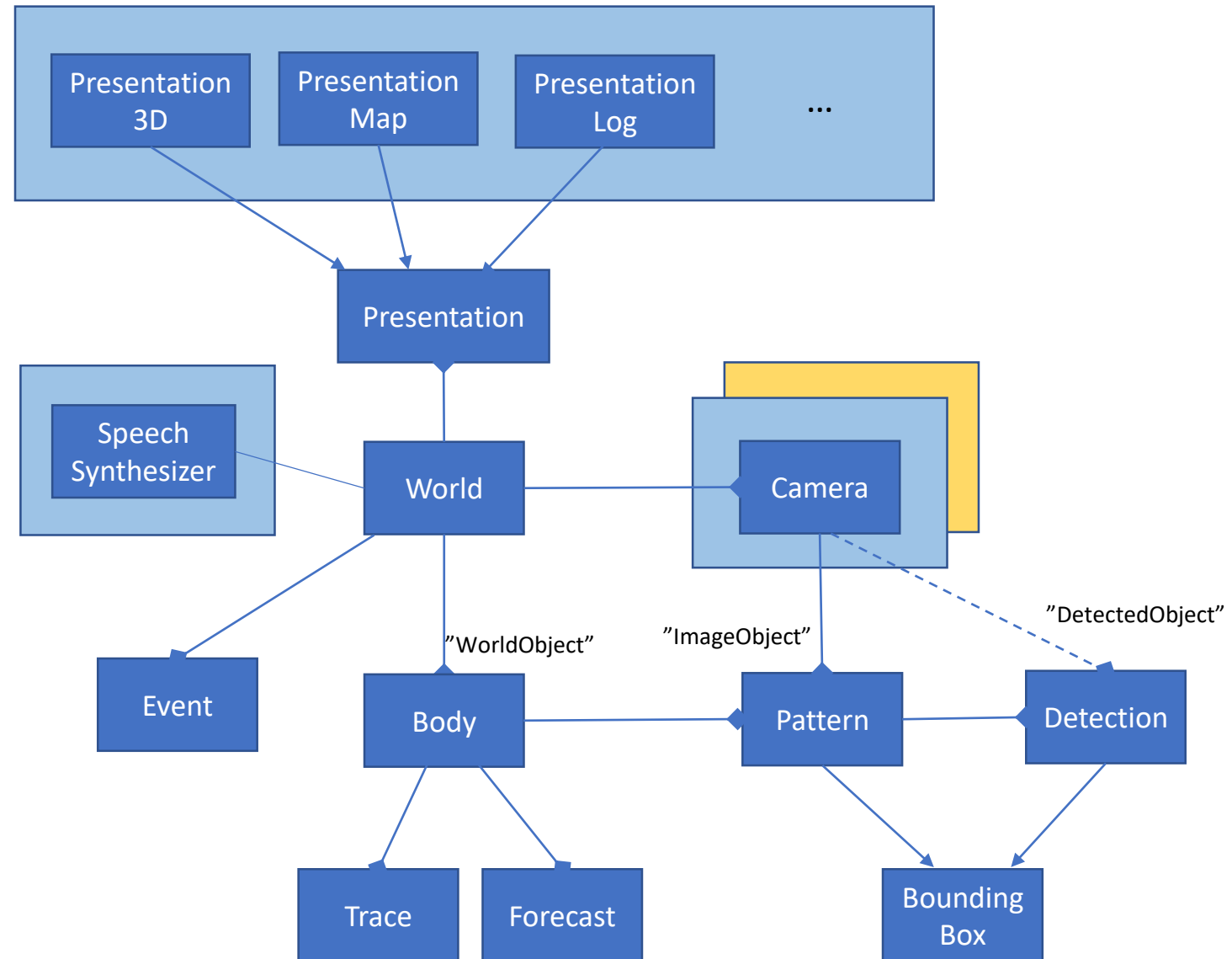
V2.0 goal

- Detected classes not hardcoded
- Object class may change
- Support for many cameras , rotations
- Names less awkward
- Cleaning
- Python style guide followed, excluding line length
- Code optimization
- One package

Name of the software package: ShadowWorld  
(Plato: Allegory of the Cave)

# Entity Diagram

V2.0



# Body Kalman Filtering

---

- Second order model does not work, constant acceleration makes bodies bounce back or get enormous velocities.
- In world, constant acceleration for several (tens) of seconds is not common
- First order model works! (No wonder it's popular in robotics...)
- When measurement is lost, the body is switched into constant velocity mode
- In filtering terminology: velocity mode = only prediction, no correction
- Algorithms described in [docs] folder documents

# Body Kalman Filtering

---

State vector  $s$ :

$$s = \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{bmatrix}$$

where

$(x, y, z)$  = location of the world object center point

$(v_x, v_y, v_z)$  = velocity of the object

State equation in differential form:

$$\frac{ds(t)}{dt} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * s(t) + \epsilon(t) = A_1 * s + \epsilon(t)$$

State equation in difference form:

$$s(k+1) = (I + \Delta * A_1) * s(k) + \epsilon(k)$$

$$= \begin{bmatrix} 1 & 0 & 0 & \Delta & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} * s(k) + \epsilon(k) = A * s(k) + \epsilon(k)$$

where  $\Delta$  is the time increment and  $\epsilon$  Gaussian noise with covariance  $R$ .

# Body Kalman Filtering

---

Measurement equation

$$z(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} * s(k) + \delta(k) = C * s(k) + \delta(k)$$

Where  $\delta$  is Gaussian noise with covariance matrix Q.

Kalman filter initialization:

$$\mu(0) = \begin{bmatrix} x(0) \\ y(0) \\ z(0) \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

where  $x(0)$ ,  $y(0)$ ,  $z(0)$  is the first location measurement.

$$\Sigma(0) = \begin{bmatrix} \alpha & 0 & 0 & 0 & 0 & 0 \\ 0 & \alpha & 0 & 0 & 0 & 0 \\ 0 & 0 & \alpha & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta & 0 & 0 \\ 0 & 0 & 0 & 0 & \beta & 0 \\ 0 & 0 & 0 & 0 & 0 & \beta \end{bmatrix}$$

where  $\alpha$  and  $\beta$  are believed variances of location and velocity.

$$R = \begin{bmatrix} r_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & r_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & r_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & r_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & r_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & r_2 \end{bmatrix}$$

where  $r_1$  and  $r_2$  are believed variances of location and velocity.

$$Q = \begin{bmatrix} q & 0 & 0 \\ 0 & q & 0 \\ 0 & 0 & q \end{bmatrix}$$

Where  $q$  is the believed measurement variance.

# Body Kalman Filtering

Kalman filter update:

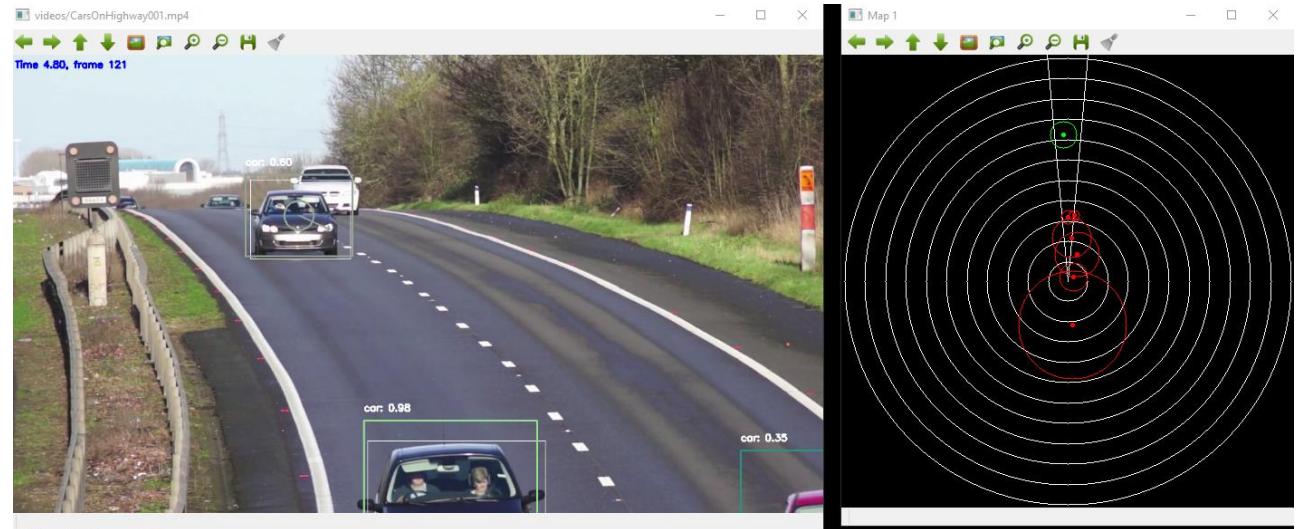
$$\mu_1(k) = A * \mu(k-1)$$

$$\Sigma_1(k) = A * \Sigma(k-1) * A^T + R$$

$$K(k) = \Sigma_1(k) * C^T (C * \Sigma_1(k) * C^T + Q)^{-1}$$

$$\mu(k) = \mu_1(k) + K(k) * (z(k) - C * \mu_1(k))$$

$$\Sigma(k) = (I - K(k) * C) * \Sigma_1(k)$$

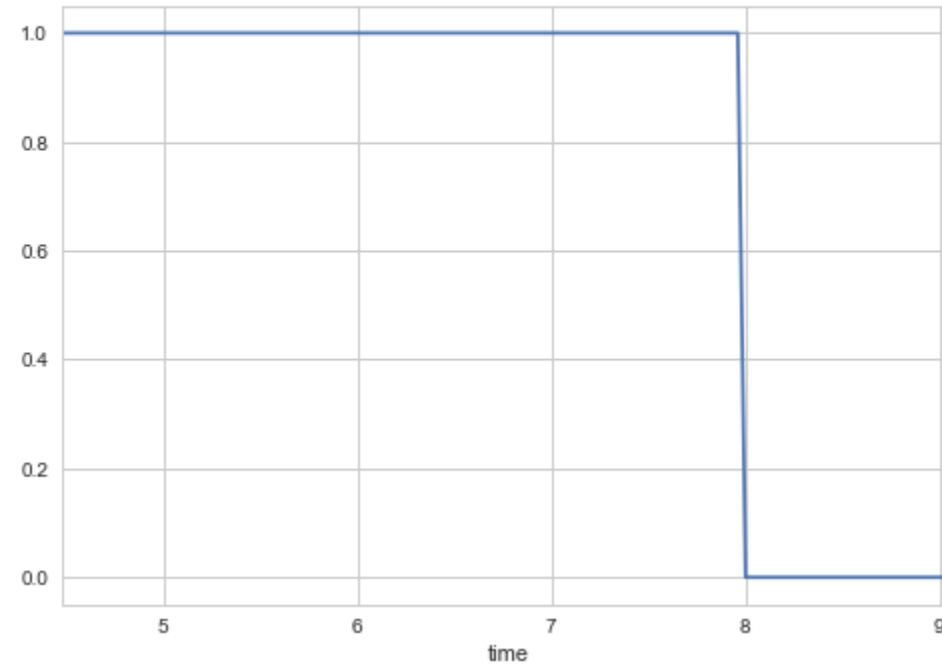


# Body Kalman Filtering

---

```
In [606]: data_one['status'].plot()
```

```
Out[606]: <matplotlib.axes._subplots.AxesSubplot at 0x1f5dbe24be0>
```



Measurement status:

1=measurement

0=no measurement

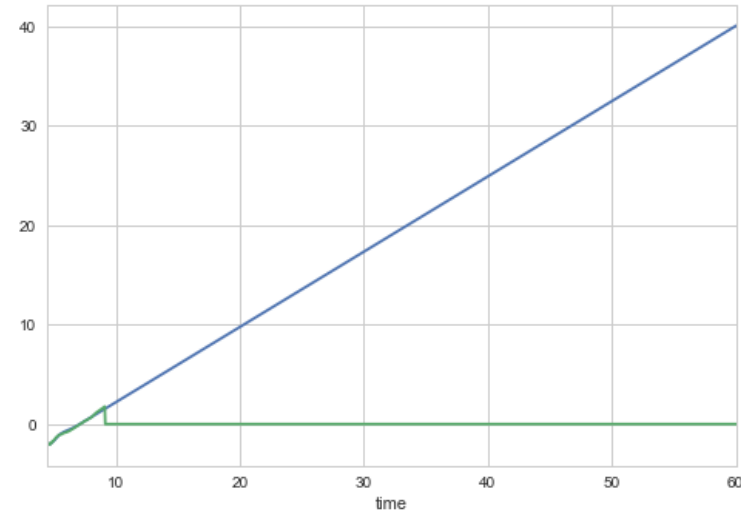


# Body Kalman Filtering

---

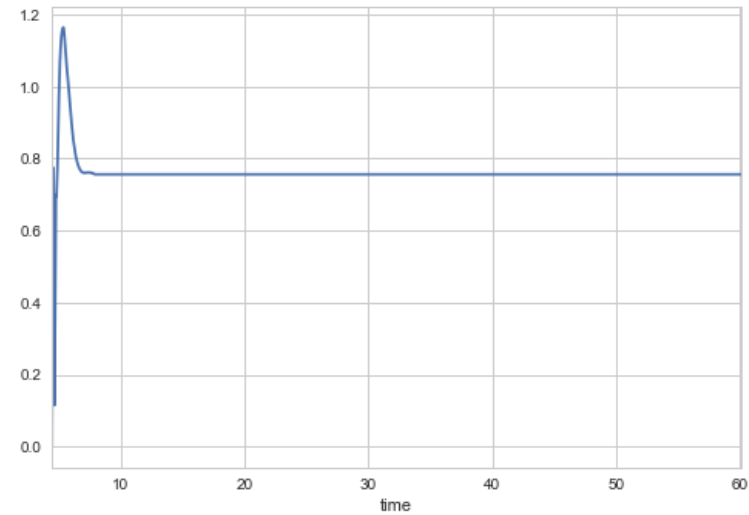
```
In [612]: data_one['x'].plot() # estimated, blue  
data_one['x_pattern'].plot() # measured, green
```

```
Out[612]: <matplotlib.axes._subplots.AxesSubplot at 0x1f5defad828>
```



```
In [613]: data_one['vx'].plot() # blue
```

```
Out[613]: <matplotlib.axes._subplots.AxesSubplot at 0x1f5dbf720f0>
```

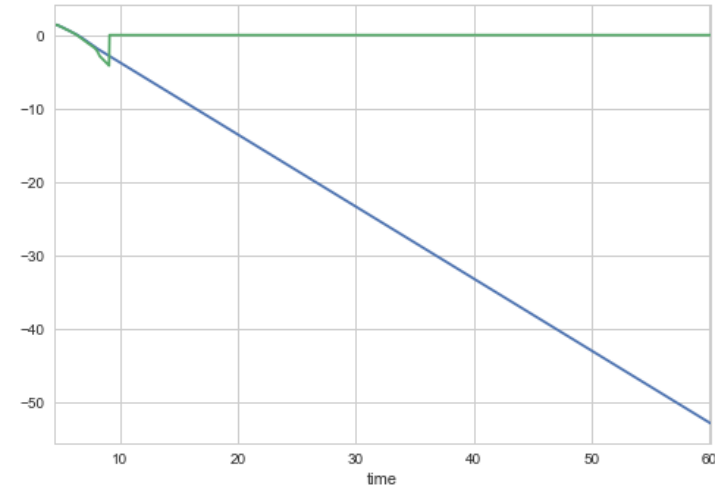


# Body Kalman Filtering

---

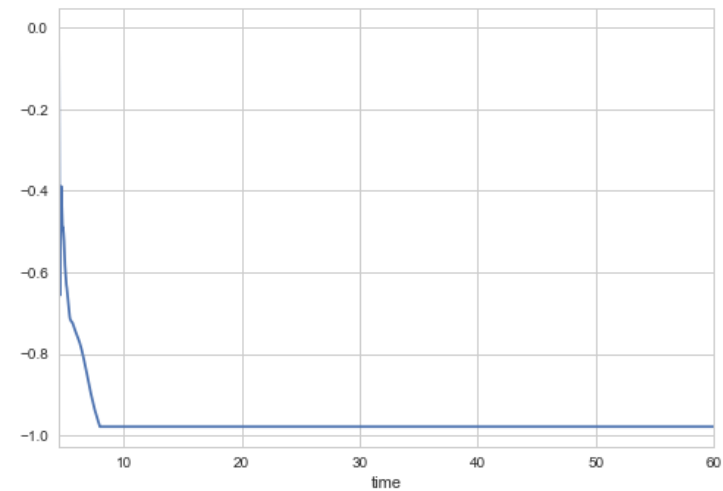
```
In [614]: data_one['y'].plot() # estimated, blue  
data_one['y_pattern'].plot() # measured, green
```

```
Out[614]: <matplotlib.axes._subplots.AxesSubplot at 0x1f5e003d0b8>
```



```
In [615]: data_one['vy'].plot() # blue
```

```
Out[615]: <matplotlib.axes._subplots.AxesSubplot at 0x1f5e00cd908>
```

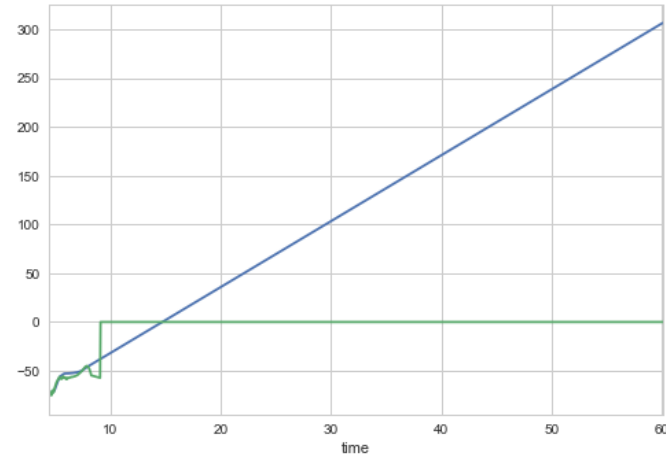


# Body Kalman Filtering

---

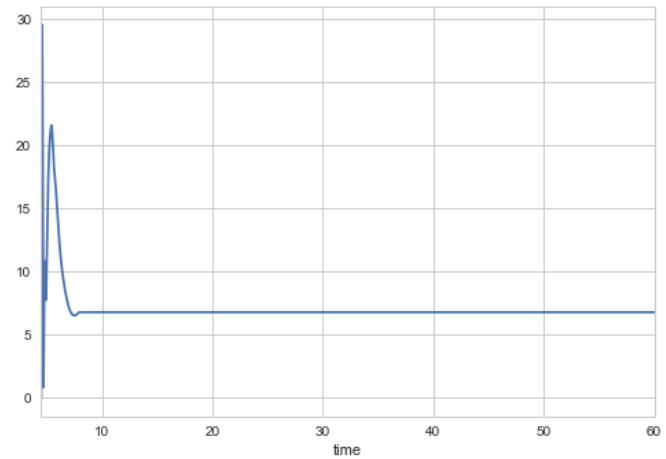
```
In [616]: data_one['z'].plot() # estimated, blue  
data_one['z_pattern'].plot() # measured, green
```

```
Out[616]: <matplotlib.axes._subplots.AxesSubplot at 0x1f5def60780>
```



```
In [617]: data_one['vz'].plot() # blue
```

```
Out[617]: <matplotlib.axes._subplots.AxesSubplot at 0x1f5d827f0b8>
```

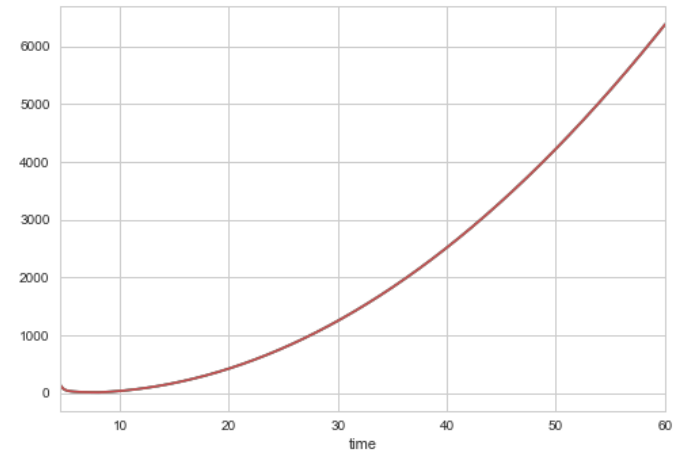


# Body Kalman Filtering

---

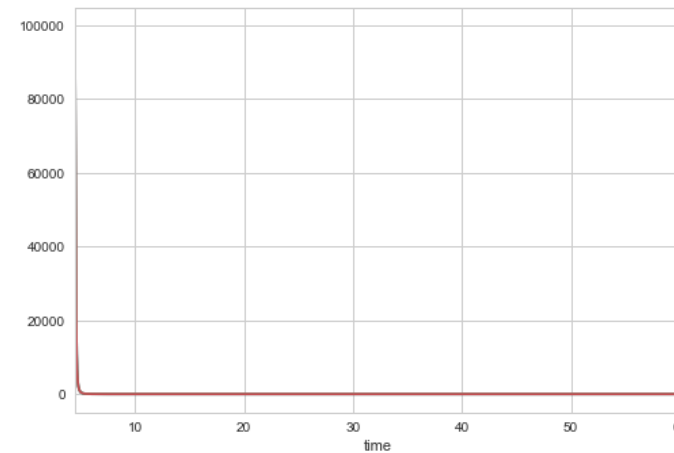
```
In [621]: data_one['sigma_00'].plot() # x variance, blue  
data_one['sigma_11'].plot() # y variance, green  
data_one['sigma_22'].plot() # y variance, green
```

Out[621]: <matplotlib.axes.\_subplots.AxesSubplot at 0x1f5e0280d68>



```
In [622]: data_one['sigma_33'].plot() # vx variance, blue  
data_one['sigma_44'].plot() # vy variance, green  
data_one['sigma_55'].plot() # vy variance, green
```

Out[622]: <matplotlib.axes.\_subplots.AxesSubplot at 0x1f5e03d60b8>



# Body Kalman Filtering

```
16 # Hyperparameters-----
17 BODY_ALFA = 100000.0 # Body initial location error variance 200
18 BODY_BETA = 100000.0 # Body initial velocity error variance 10000
19 BODY_C = np.array([[1.0, 0.0, 0.0, 0.0, 0.0, 0.0],
20                    [0.0, 1.0, 0.0, 0.0, 0.0, 0.0],
21                    [0.0, 0.0, 1.0, 0.0, 0.0, 0.0]
22                    ]) # Body measurement matrix
23 BODY_DATA_COLLECTION_COUNT = 30 # How many frames until notification
24 BODY_Q = np.array([[200.0, 0.0, 0.0],
25                    [0.0, 200.0, 0.0],
26                    [0.0, 0.0, 200.0]]) # Body measurement variance 200
27 BODY_R = np.array([[0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
28                    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
29                    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
30                    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
31                    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
32                    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
33                    ]) # Body state equation covariance
```

Kalman filter update:

$$\mu_1(k) = A * \mu(k-1)$$

$$\Sigma_1(k) = A * \Sigma(k-1) * A^T + R$$

$$K(k) = \Sigma_1(k) * C^T (C * \Sigma_1(k) * C^T + Q)^{-1}$$

$$\mu(k) = \mu_1(k) + K(k) * (z(k) - C * \mu_1(k))$$

$$\Sigma(k) = (I - K(k) * C) * \Sigma_1(k)$$

- Filtering requires several seconds to achieve reliable results
- Kalman filter parameters are not optimized
- Optimization will be done later

# Paper

---

- One solution for location estimation and movement prediction for video detected object (nearly) solved
- Work needed:
  - Kalman filter parameter adjustment\*
  - Experiments in the wild (see previous slides)
  - Other tests? (using existing (tracking) dataset)
- Where to publish?
  - ECCV 2018 (8-14.9, Munich, deadline **14.3.2018, too soon**)
  - CVPR 2019 (6/2019, Long Beach, deadline 11/2018)
  - ICCV 2019 (29.10.-3.11.2019, Seoul, deadline 1/2019)

\*Locations and especially velocities take too much time to settle at the moment.

# Paper

## Image-based situation awareness: Estimating location and velocity using single camera object detection

### Contents:

- 1. Abstract
  - 2. Introduction
  - 3. Related work
  - 4. Our approach
    - 1. Definitions
    - 2. Object identity
    - 3. Distance estimation
    - 4. Location and velocity estimation
    - 5. Movement prediction
  - 5. Experiments
    - 1. Test setup
    - 2. Evaluation metric
    - 3. Results
    - 4. Other experiments
  - 6. Conclusion
- 
- Topics and their connections:
- ????? (Red text) points to 3. Related work
  - detection, pattern... points to 4. Our approach
  - Hungarian algorithm, distance metrics, movement prediction points to 4. Our approach
  - geometry points to 4. Our approach
  - Kalman filtering points to 4. Our approach
  - Difference equation, uncertainties points to 4. Our approach
  - In the wild. See previous slides. More? (Red text) points to 5. Experiments
  - Existing datasets? Which? (Red text) points to 6. Conclusion



Work in Progress



# Perception

---

“The first step in achieving SA is to perceive the status, attributes, and dynamics of relevant elements in the environment. Thus, Level 1 SA, the most basic level of SA, involves the processes of monitoring, cue detection, and simple recognition, which lead to an awareness of multiple situational elements (objects, events, people, systems, environmental factors) and their current states (locations, conditions, modes, actions).”



# Next Steps

# Next steps

---

## **Comprehension:**

1. Closing the open questions
2. 2d -> 3d transformation
3. World object state estimation



To Be Discussed

# To Be Discussed

---

- Activity recognition?
- Emotion recognition?
- Turning camera, estimation by background movement?

# Thank you!

[lampola@student.tut.fi](mailto:lampola@student.tut.fi)

<https://github.com/SakariLampola/Thesis>