



Image-Based Situation Awareness Audit 28.2.2018

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 algorithm, 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, roll)
- 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 have a diagonal edge that meets at the center of the slide.

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

arXiv.org e-Print archive

Cornell University Library

arXiv.org

Search or Article ID [] All papers [] (title | Advanced search)

Login

Open access to 1,358,453 e-prints in Physics, Mathematics, Computer Science, Quantitative Biology, Quantitative Finance, Statistics, Electrical Engineering and Systems Science, and Economics

Subject search and browse: [Physics] [Search] [Form Interface] [Catchup]

02 Jan 2018: 1991-2017 submission rate statistics are now available.
See cumulative "What's New" pages. Read robots beware before attempting any automated download

Physics

- Astrophysics (**astro-ph** new, recent, find)
Includes: Astrophysics of Galaxies; Cosmology and Nongalactic Astrophysics; Earth and Planetary Astrophysics; High Energy Astrophysical Phenomena; Instrumentation and Methods for Astrophysics; Solar and Stellar Astrophysics
- Condensed Matter (**cond-mat** new, recent, find)
Includes: Disordered Systems and Neural Networks; Materials Science; Mesoscale and Nanoscale Physics; Other Condensed Matter; Quantum Gases; Soft Condensed Matter; Statistical Mechanics; Strongly Correlated Electrons; Superconductivity
- General Relativity and Quantum Cosmology (**gr-qc** new, recent, find)
- High Energy Physics - Experiment (**hep-ex** new, recent, find)
- High Energy Physics - Lattice (**hep-lat** new, recent, find)
- High Energy Physics - Phenomenology (**hep-ph** new, recent, find)
- High Energy Physics - Theory (**hep-th** new, recent, find)
- Mathematical Physics (**math-ph** new, recent, find)
- Nonlinear Sciences (**nlin** new, recent, find)
Includes: Adaptation and Self-Organizing Systems; Cellular Automata and Lattice Gases; Chaotic Dynamics; Exactly Solvable and Integrable Systems; Pattern Formation and Solitons
- Nuclear Experiment (**nuc-ex** new, recent, find)
- Nuclear Theory (**nuc-th** new, recent, find)
- Physics (**physics** new, recent, find)
Includes: Accelerator Physics; Applied Physics; Atmospheric and Oceanic Physics; Atomic Physics; Atomic and Molecular Clusters; Biological Physics; Chemical Physics; Classical Physics; Computational Physics; Data Analysis; Statistics and Probability; Fluid Dynamics; General Physics; Geophysics; History and Philosophy of Physics; Instrumentation and Detectors; Medical Physics; Optics; Physics Education; Physics and Society; Plasma Physics; Popular Physics; Space Physics
- Quantum Physics (**quant-ph** new, recent, find)

Mathematics

- Mathematics (**math** new, recent, find)
Includes (see detailed description): Algebraic Geometry; Algebraic Topology; Analysis of PDEs; Category Theory; Classical Analysis and ODEs; Combinatorics; Commutative Algebra; Complex Variables; Differential Geometry; Dynamical Systems; Functional Analysis; General Mathematics; General Topology; Geometric Topology; Group Theory; History and Overview; Information Theory; K-Theory and Homology; Logic; Mathematical Physics; Metric Geometry; Number Theory; Numerical Analysis; Operator Algebras; Optimization and Control; Probability; Quantum Algebra; Representation Theory; Rings and Algebras; Spectral Theory; Statistics Theory; Symplectic Geometry

Computer Science

- Computing Research Repository (**CoRR** new, recent, find)
Includes (see detailed description): Artificial Intelligence; Computation and Language; Computational Complexity; Computational Engineering, Finance, and Science; Computational Geometry; Computer Science and Game Theory; Computer Vision and Pattern Recognition; Computers and Society; Cryptography and Security; Data Structures and Algorithms; Databases; Digital Libraries; Discrete Mathematics; Distributed, Parallel, and Cluster Computing; Emerging Technologies; Formal Languages and Automata Theory; General Literature; Graphics; Hardware Architecture; Human-Computer Interaction; Information Retrieval; Information Theory; Learning; Logic in Computer Science; Mathematical Software; Multimedia; Networking and Internet Architecture; Neural and Evolutionary Computing; Numerical Analysis; Operating Systems; Other Computer Science; Performance; Programming Languages; Robotics; Social and Information Networks; Software Engineering; Sound; Symbolic Computation; Systems and Control



arXiv.org e-Print archive

GitHub, Inc. [US] | https://github.com/SakariLampola/Thesis/tree/master/papers

This repository Search Pull requests Issues Marketplace Explore

SakariLampola / Thesis

Watch 0 Star 0 Fork 0

Code Issues Pull requests Projects Wiki Insights Settings

Branch: master Thesis / papers / Create new file Upload files Find file History

SakariLampola 20180213	Latest commit 9e4d839 20 hours ago	
--		
ComputerVision	20180213	20 hours ago
NLP	20180207b	7 days ago
Robotics	20180219	5 days ago

© 2018 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About

arXiv.org e-Print archive

GitHub, Inc. [US] | https://github.com/SakariLampola/Thesis/tree/master/papers/ComputerVision

This repository Search Pull requests Issues Marketplace Explore

SakariLampola / Thesis

Code Issues Pull requests Projects Wiki Insights Settings

Branch: master Thesis / papers / ComputerVision / Create new file Upload files Find file History

--		
AnswererToQuestionModelerGoalOrientedVisualDialogue.pdf	20180213	21 hours ago
DualRecurrentAttentionModelForVisualQuestionAnswering.pdf	20180207b	7 days ago
ExplainingFeatureImportanceModelingRecognizingFeatureImportanceApparentPersonalityFrom...	20180207b	7 days ago
FromBenedictCumberbatchToTheTechHulmeCharacterIdentificationWithTfIdF.pdf	20180207b	7 days ago
HowWouldYouSoundOnTheWeb.pdf	20180207b	7 days ago
ImageCaptioningWithIterativeSchemeOfEffectivelyExploitingSentiments.pdf	20180207b	7 days ago
ImprovingMultipleObjectTrackingWithOpticalFlowAndDeepProcessing.pdf	20180207b	7 days ago
MobileFeedForwardConvolutionalNeuralNetworkForMobileVisionApplications.pdf	20180207b	7 days ago
ObjectBasedFaceRecognitionQA.pdf	20180207b	7 days ago
ObjectDetectionInNoisyShortAndLongRangeObjectLinking.pdf	20180207b	7 days ago
Open3DModelLibraryFor3DObjectProcessing.pdf	20180207b	7 days ago
ParallelTrackingAndVerification.pdf	20180207b	7 days ago
SD2LongShortTermBoxDetector.pdf	20180207b	7 days ago
Size to Depth-A New Perspective for Single Image Estimation.pdf	20180207b	7 days ago
StructureTripleLearningWithFOTTagGuidedAttention.pdf	20180207b	7 days ago
TVLDeepDiffusionOfDeepLearningModelsForTimeSeriesAnalysis.pdf	20180219	5 days ago
TheChallengeOfSimultaneousObjectDetectionAndPoseEstimationComparison.pdf	20180207b	7 days ago
TrackingMultipleMovingObjectsUsingUncertaintyAwareFilteringTechniques.pdf	20180207b	7 days ago
WhatMakesGoodSyntheticLearningData.pdf	20180207b	7 days ago

Method Survey



Lecture Collection | Natural Language Processing with Deep Learning (Winter 2017)

19 videos • 342,803 views • Last updated on Apr 3, 2017










Stanford University School of Engineering

SUBSCRIBED 48K

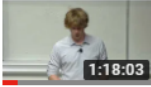


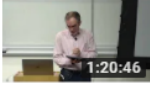










Natural language processing (NLP) deals with the key artificial intelligence technology of understanding complex human language communication. This lecture series provides a thorough introduction to the cutting-edge research in deep learning applied to NLP, an approach that has recently obtained very high performance across many different NLP tasks including question answering and machine translation.

Christoffer Manning & Richard Socher

-  **Lecture 1 | Natural Language Processing with Deep Learning**
Stanford University School of Engineering
1:11:41
-  **Lecture 2 | Word Vector Representations: word2vec**
Stanford University School of Engineering
1:18:17
-  **Lecture 3 | GloVe: Global Vectors for Word Representation**
Stanford University School of Engineering
1:18:40
-  **Lecture 4: Word Window Classification and Neural Networks**
Stanford University School of Engineering
1:16:43
-  **Lecture 5: Backpropagation and Project Advice**
Stanford University School of Engineering
1:18:20
-  **Lecture 6: Dependency Parsing**
Stanford University School of Engineering
1:23:07
-  **Lecture 7: Introduction to TensorFlow**
Stanford University School of Engineering
1:12:33

Method Survey

8	 Lecture 8: Recurrent Neural Networks and Language Models Stanford University School of Engineering	15	 Lecture 14: Tree Recursive Neural Networks and Constituency Parsing Stanford University School of Engineering
9	 Lecture 9: Machine Translation and Advanced Recurrent LSTMs and GRUs Stanford University School of Engineering	16	 Lecture 15: Coreference Resolution Stanford University School of Engineering
10	 Review Session: Midterm Review Stanford University School of Engineering	17	 Lecture 16: Dynamic Neural Networks for Question Answering Stanford University School of Engineering
11	 Lecture 10: Neural Machine Translation and Models with Attention Stanford University School of Engineering	18	 Lecture 17: Issues in NLP and Possible Architectures for NLP Stanford University School of Engineering
12	 Lecture 11: Gated Recurrent Units and Further Topics in NMT Stanford University School of Engineering	19	 Lecture 18: Tackling the Limits of Deep Learning for NLP Stanford University School of Engineering
13	 Lecture 12: End-to-End Models for Speech Processing Stanford University School of Engineering		
14	 Lecture 13: Convolutional Neural Networks Stanford University School of Engineering		

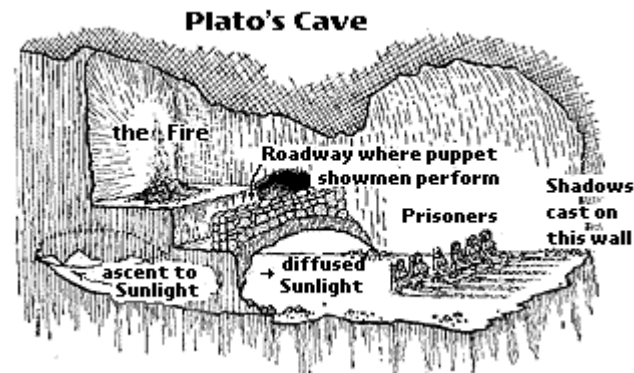
Goodfellow, Bengio, Courville: Deep Learning

Software Version 2.0

V2.0 Goal

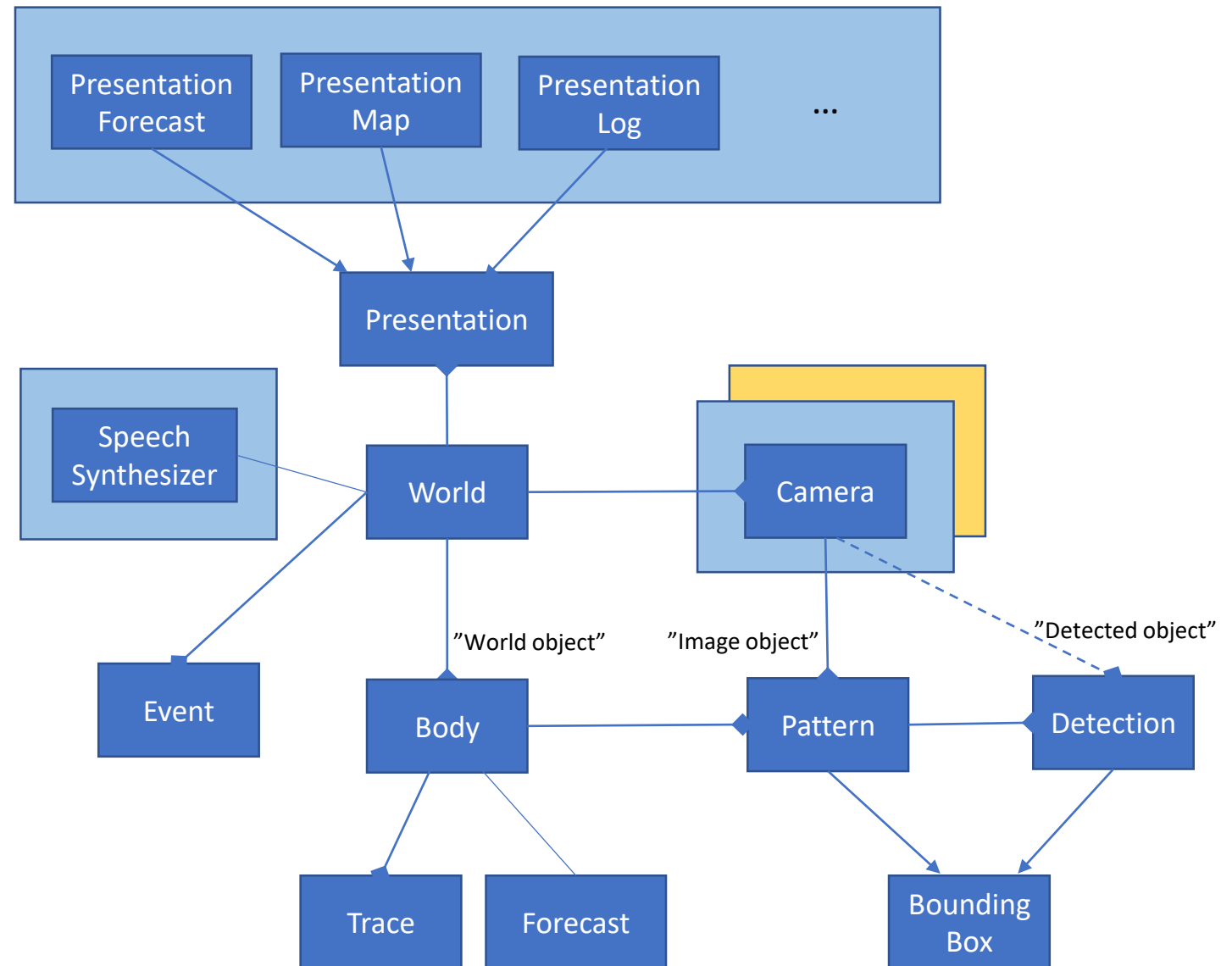
- Detected classes not hardcoded
- Object class may change
- Support for many cameras , rotations, movement
- Names less awkward, terminology fixed
- Cleaning
- Python style guide followed, excluding line length
- Code optimization
- One package
- Language (speech synthesizer)

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

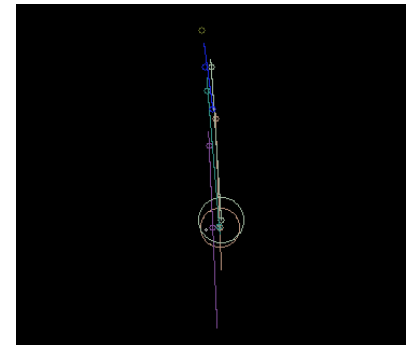
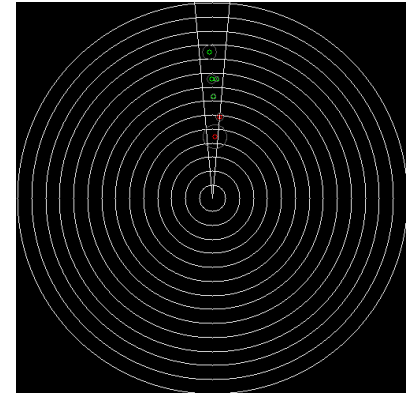
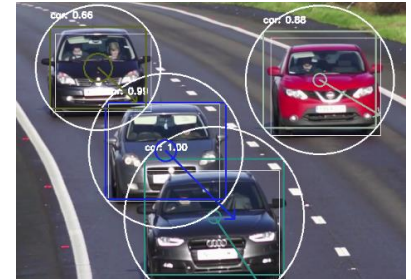
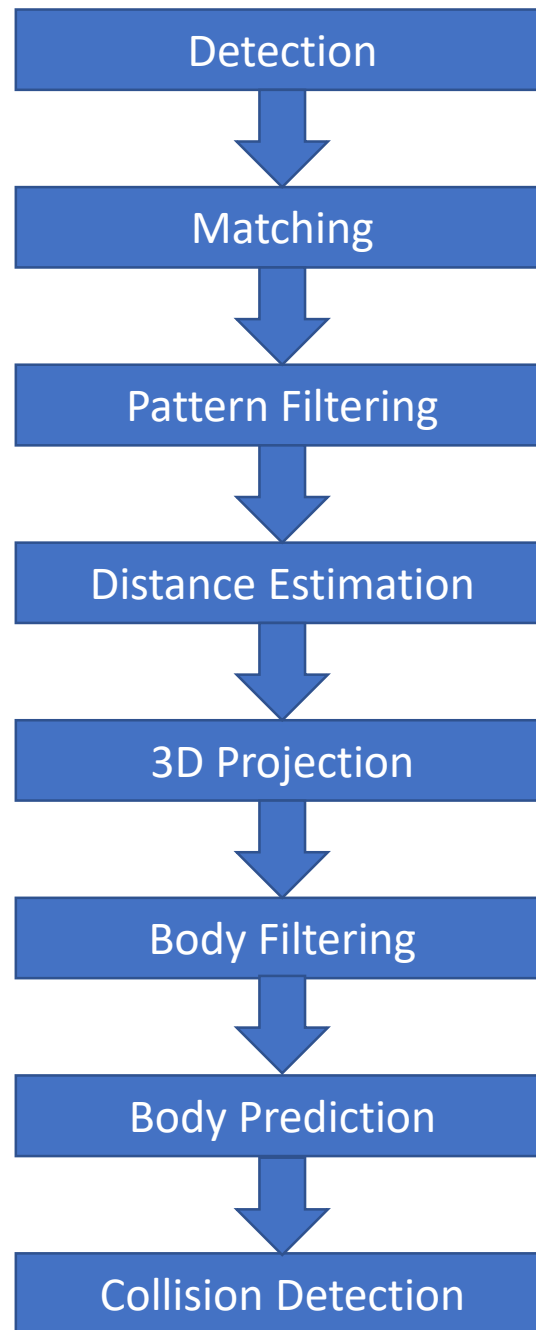


Software Version 2.0

Class Diagram

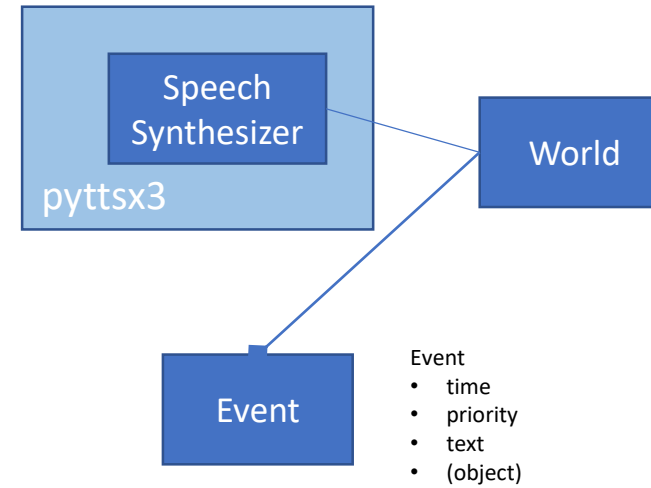


Software Version 2.0



Software Version 2.0

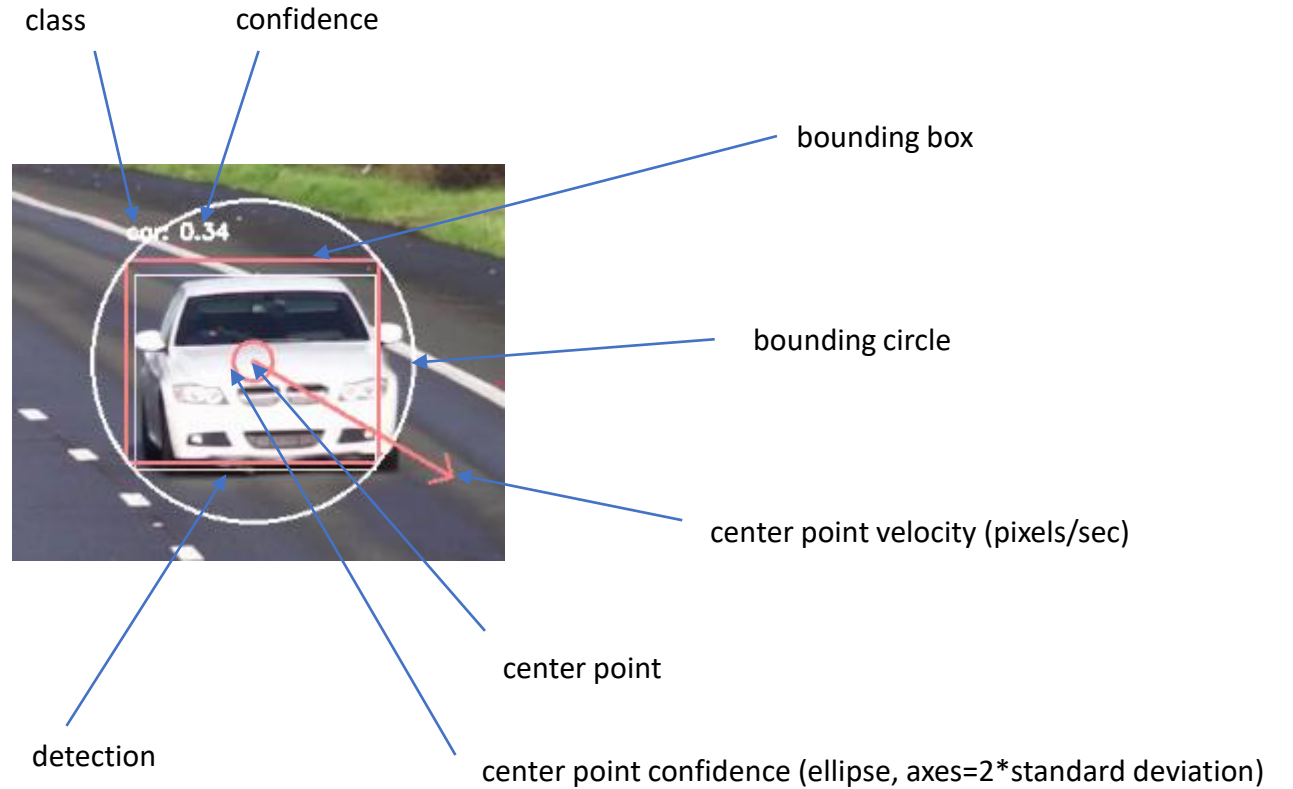
Language



- Speech synthesizer based on pytsx3 package
- Event is spelled out if priority ≤ 0
- Event will pause video for the duration of speech (to be changed later by using separate thread)
- Example events:
 - Body observed (1 sec after created)
 - Collision warning
- Speech recognition later

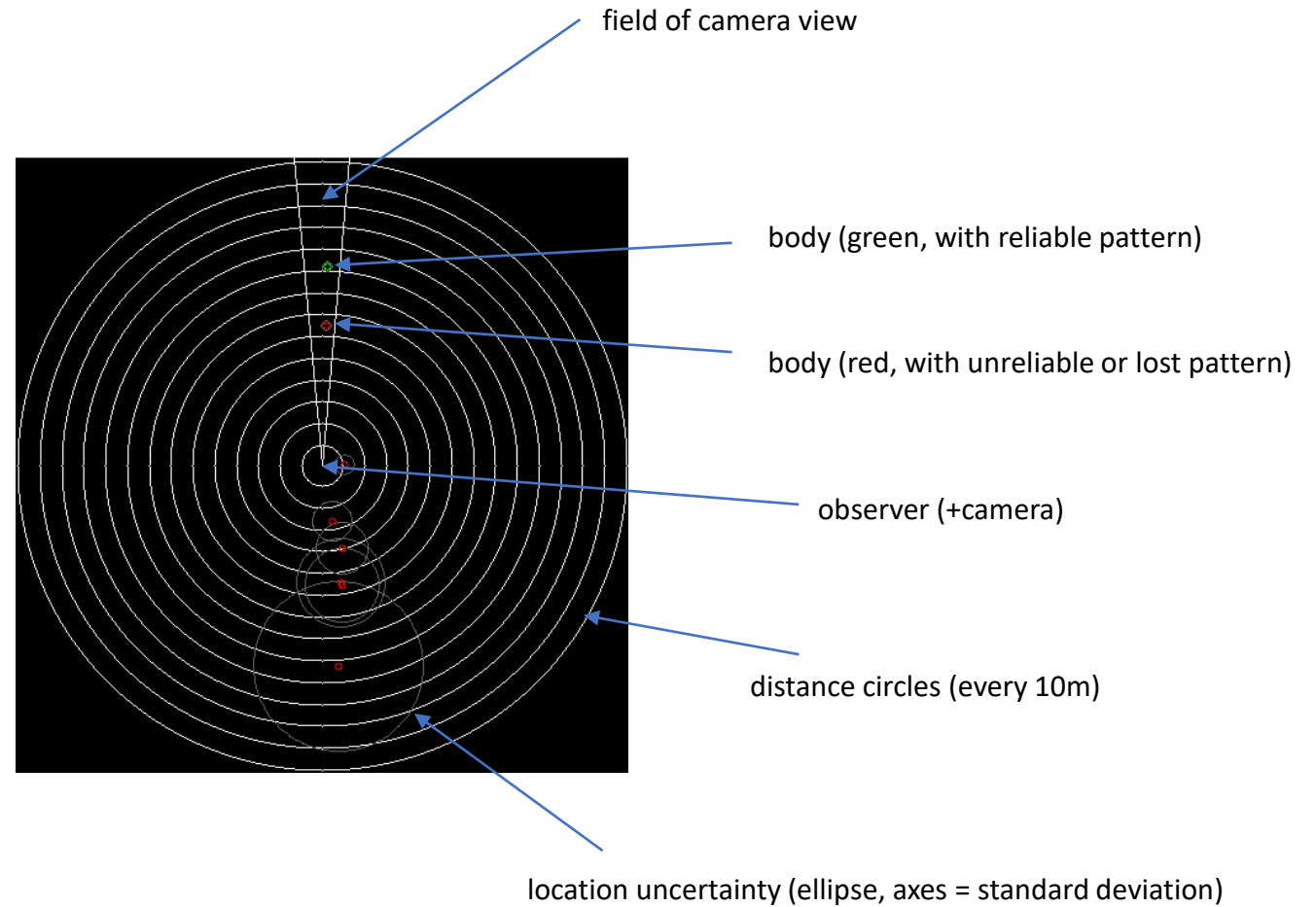
Software Version 2.0

Visual Presentation (pattern)



Software Version 2.0

Visual Presentation (body)



body size (circle, radius = mean class specific radius)

Software Version 2.0

Logging

```
Body.txt Detection.txt Event.txt Pattern.txt
1 time,id,class_id,x,y,z,vx,vy,vz,sigma_00,sigma_01,sigma_02,sigma_03,sigma_04,sigma_05,sigma_10
2 0.000,2061579034864,7,3.452,-0.145,-96.541,0.000,0.000,0.000,199.601,0.000,0.000,7.971,0.000,0
3 0.000,2061579035088,7,-0.226,-1.349,-89.454,0.000,0.000,0.000,199.601,0.000,0.000,7.971,0.000,0
4 0.040,2061579034864,7,3.507,-0.183,-96.872,0.612,-0.426,-3.685,128.569,0.000,0.000,1429.181,0.
5 0.040,2061579035088,7,-0.209,-1.372,-88.698,0.182,-0.255,8.404,128.569,0.000,0.000,1429.181,0.
6 0.080,2061579034864,7,3.579,-0.228,-97.404,1.181,-0.758,-8.294,128.173,0.000,0.000,1536.731,0.
7 0.080,2061579035088,7,-0.193,-1.397,-88.220,0.289,-0.429,10.111,128.173,0.000,0.000,1536.731,0
8 0.080,2061579036712,7,0.591,2.067,-118.685,0.000,0.000,0.000,199.601,0.000,0.000,7.971,0.000,0
9 0.120,2061579034864,7,3.636,-0.299,-97.312,1.271,-1.158,-4.132,121.951,0.000,0.000,1198.603,0.
10 0.120,2061579035088,7,-0.170,-1.424,-87.496,0.405,-0.530,13.249,121.951,0.000,0.000,1198.603,0
11 0.120,2061579036712,7,0.620,2.098,-121.664,0.325,0.348,-33.111,128.569,0.000,0.000,1429.181,0.
12 0.160,2061579034864,7,3.706,-0.386,-97.300,1.425,-1.481,-2.710,111.067,0.000,0.000,888.002,0.0
13 0.160,2061579035088,7,-0.137,-1.478,-87.432,0.535,-0.791,9.528,111.067,0.000,0.000,888.002,0.0
14 0.160,2061579036712,7,0.650,2.122,-123.217,0.530,0.466,-35.849,128.173,0.000,0.000,1536.731,0.
15 0.160,2061579038560,7,-1.793,0.215,-107.421,0.000,0.000,0.000,199.601,0.000,0.000,7.971,0.000,
16 0.200,2061579034864,7,3.791,-0.455,-97.492,1.616,-1.550,-3.271,99.964,0.000,0.000,666.108,0.00
17 0.200,2061579035088,7,-0.103,-1.548,-87.545,0.624,-1.047,6.234,99.964,0.000,0.000,666.108,0.00
18 0.200,2061579036712,7,0.678,2.116,-122.864,0.597,0.220,-18.293,121.951,0.000,0.000,1198.603,0.
19 0.200,2061579038560,7,-1.802,0.192,-107.854,-0.108,-0.258,-4.809,128.569,0.000,0.000,1429.181,
20 0.240,2061579034864,7,3.848,-0.515,-97.046,1.572,-1.538,0.010,90.081,0.000,0.000,512.455,0.000
21 0.240,2061579035088,7,-0.066,-1.624,-87.436,0.688,-1.241,5.433,90.081,0.000,0.000,512.455,0.00
22 0.240,2061579036712,7,0.716,2.084,-121.887,0.710,-0.101,-4.624,111.067,0.000,0.000,888.002,0.0
23 0.240,2061579038560,7,-1.797,0.181,-107.885,0.003,-0.262,-2.878,128.173,0.000,0.000,1536.731,0
24 0.280,2061579034864,7,3.904,-0.567,-96.519,1.537,-1.489,2.623,81.625,0.000,0.000,404.376,0.000
25 0.280,2061579035088,7,-0.025,-1.708,-87.195,0.758,-1.408,5.551,81.625,0.000,0.000,404.376,0.00
26 0.280,2061579036712,7,0.769,2.066,-121.510,0.873,-0.200,-0.880,99.964,0.000,0.000,666.108,0.00
27 0.280,2061579038560,7,-1.769,0.157,-107.973,0.283,-0.392,-2.613,121.951,0.000,0.000,1198.603,0
28 0.320,2061579034864,7,3.953,-0.616,-95.888,1.480,-1.443,4.926,74.449,0.000,0.000,326.355,0.000
29 0.320,2061579035088,7,0.022,-1.804,-86.974,0.830,-1.582,5.549,74.449,0.000,0.000,326.355,0.000
30 0.320,2061579036712,7,0.817,2.006,-119.710,0.947,-0.494,9.559,90.081,0.000,0.000,512.455,0.000
31 0.320,2061579038560,7,-1.721,0.126,-107.255,0.571,-0.520,3.968,111.067,0.000,0.000,888.002,0.0
32 0.360,2061579034864,7,4.001,-0.664,-95.265,1.438,-1.405,6.601,68.343,0.000,0.000,268.528,0.000
33 0.360,2061579035088,7,0.071,-1.908,-86.738,0.890,-1.744,5.602,68.343,0.000,0.000,268.528,0.000
34 0.360,2061579036712,7,0.859,1.938,-117.821,0.968,-0.734,17.022,81.625,0.000,0.000,404.376,0.00
35 0.360,2061579038560,7,-1.646,0.099,-105.881,0.925,-0.559,12.064,99.964,0.000,0.000,666.108,0.0
36 0.400,2061579034864,7,4.041,-0.712,-94.469,1.374,-1.377,8.493,63.113,0.000,0.000,224.623,0.000
37 0.400,2061579035088,7,0.117,-1.992,-86.240,0.928,-1.796,6.576,63.113,0.000,0.000,224.623,0.000
38 0.400,2061579036712,7,0.904,1.869,-116.026,0.994,-0.907,21.908,74.449,0.000,0.000,326.355,0.00
39 0.400,2061579038560,7,-1.589,0.174,-104.160,1.036,-0.006,19.110,90.081,0.000,0.000,512.455,0.0
40 0.440,2061579034864,7,4.064,-0.759,-93.474,1.272,-1.349,10.624,58.597,0.000,0.000,190.567,0.00
41 0.440,2061579035088,7,0.164,-2.066,-85.548,0.961,-1.802,7.971,58.597,0.000,0.000,190.567,0.00
42 0.440,2061579036712,7,0.963,1.810,-114.711,1.072,-0.997,23.628,68.343,0.000,0.000,268.528,0.00
43 0.440,2061579038560,7,-1.524,0.192,-102.873,1.154,0.087,21.695,81.625,0.000,0.000,404.376,0.00
44 0.480,2061579034864,7,4.068,-0.806,-92.157,1.129,-1.329,13.295,54.666,0.000,0.000,163.653,0.00
45 0.480,2061579035088,7,0.214,-2.135,-84.872,0.995,-1.793,9.042,54.666,0.000,0.000,163.653,0.00
46 0.480,2061579036712,7,1.026,1.753,-113.527,1.142,-1.056,24.481,63.113,0.000,0.000,224.623,0.00
```

Software Version 2.0

How to use?

```
def run_application():
    """
    Example application
    """
    test_video = 5

    world = World()

    world.add_camera(Camera(world, focal_length=TEST_FOCAL_LENGTHS[test_video],
                             sensor_width=0.0359, sensor_height=0.0240,
                             x=0.0, y=0.0, z=0.0,
                             yaw=0.0, pitch=0.0, roll=0.0,
                             videofile=TEST_VIDEOS[test_video]))

    world.add_presentation(PresentationMap(world, map_id=1, height_pixels=500,
                                           width_pixels=500,
                                           extent=TEST_EXTENTS[test_video]))

    world.add_presentation(PresentationForecast(world, map_id=2,
                                                height_pixels=500,
                                                width_pixels=500,
                                                extent=TEST_EXTENTS[test_video]))

    world.add_presentation(PresentationLog(world, "Detection", "Detection.txt"))
    world.add_presentation(PresentationLog(world, "Pattern", "Pattern.txt"))
    world.add_presentation(PresentationLog(world, "Body", "Body.txt"))
    world.add_presentation(PresentationLog(world, "Event", "Event.txt"))

    world.run()

if __name__ == "__main__":
    run_application()
```

Demo

- 3-5 videos
- Program code

Pattern Filtering

Why pattern filtering?

- Reduces object detection (bounding box) noise
- Provides prediction for pattern location in the next frame (matching easier)
- Predicts pattern location when detection is missing

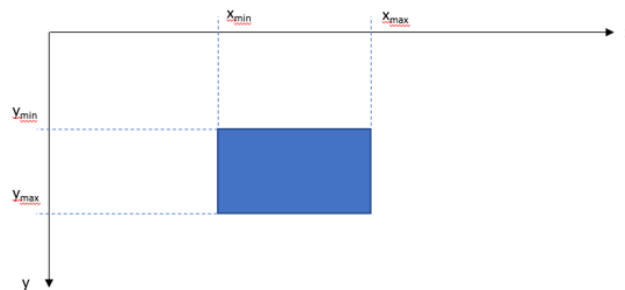
Hyperparameters:

```
52 #
53 PATTERN_ALFA = 200.0 # Pattern initial location error variance
54 PATTERN_BETA = 10000.0 # Pattern initial velocity error variance
55 PATTERN_C = np.array([[1.0, 0.0]]) # Pattern measurement matrix
56 PATTERN_Q = np.array([200.0]) # Pattern measurement variance
57 PATTERN_R = np.array([[0.1, 0.0],
58                        [0.0, 1.0]]) # Pattern state equation covariance
59 #
```

Pattern Filtering

Pattern Kalman Filtering

Bounding box edge coordinates



Pattern location (bounding box) is determined by four edge coordinates: x_{min} , x_{max} , y_{min} and y_{max} . vx_{min} , vx_{max} , vy_{min} and vy_{max} are corresponding velocities.

Each edge coordinate is filtered separately and identically. x_{min} is used here as an example.

State equation in differential form:

$$\frac{d}{dt} \begin{bmatrix} x_{min}(t) \\ vx_{min}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} * \begin{bmatrix} x_{min}(t) \\ vx_{min}(t) \end{bmatrix} + \epsilon(t)$$

State equation in difference form:

$$\begin{bmatrix} x_{min}(k+1) \\ vx_{min}(k+1) \end{bmatrix} = A * \begin{bmatrix} x_{min}(k) \\ vx_{min}(k) \end{bmatrix} + \epsilon(k)$$

$$A = \begin{bmatrix} 1 & \Delta \\ 0 & 1 \end{bmatrix}$$

where Δ is the time increment and ϵ Gaussian noise with covariance R:

$$R = \begin{bmatrix} 0.1 & 0 \\ 0 & 1.0 \end{bmatrix}$$

Measurement equation

$$z(k) = C * \begin{bmatrix} x_{min}(k) \\ vx_{min}(k) \end{bmatrix} + \delta(k)$$

$$C = [1 \quad 0]$$

where δ is Gaussian noise with covariance matrix Q:

$$Q = [200.0]$$

Kalman filter initialization:

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

where $x_{min}(0)$ is the first location measurement.

$$\Sigma(0) = \begin{bmatrix} 200.0 & 0 \\ 0 & 10000.0 \end{bmatrix}$$

Kalman filter update:

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

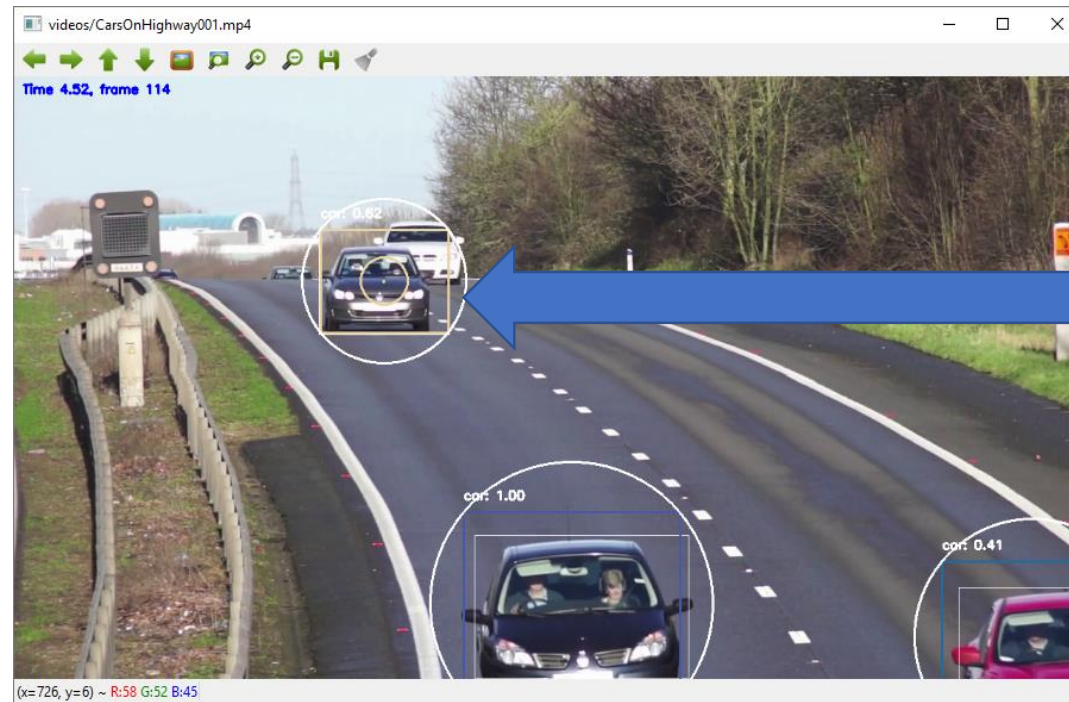
$$\Sigma_1(k) = A * \Sigma(k-1) * A^T + R$$

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

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

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

Pattern Filtering



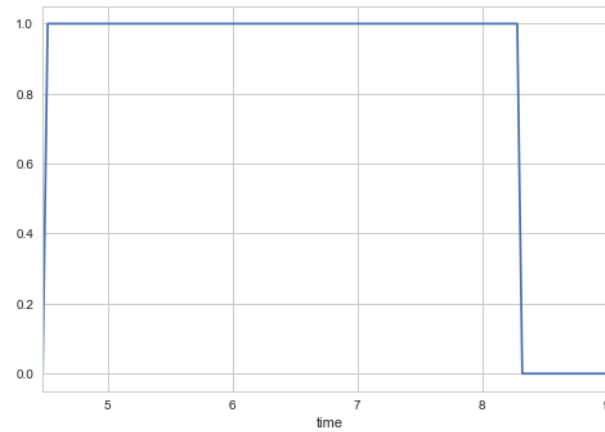
```
Event.txt
630 4.440,3,2061577943248,Detection created
631 4.440,2,2061579199488,Pattern removed
632 4.480,3,2061577756008,Detection created
633 4.480,3,2061577754272,Detection created
634 4.480,3,2061577755112,Detection created
635 4.480,1,2061577797816,Body created
636 4.480,2,2061577776432,Pattern created
637 4.520,3,2061577844944,Detection created
```

Pattern Filtering

Matched status, coordinate and velocity

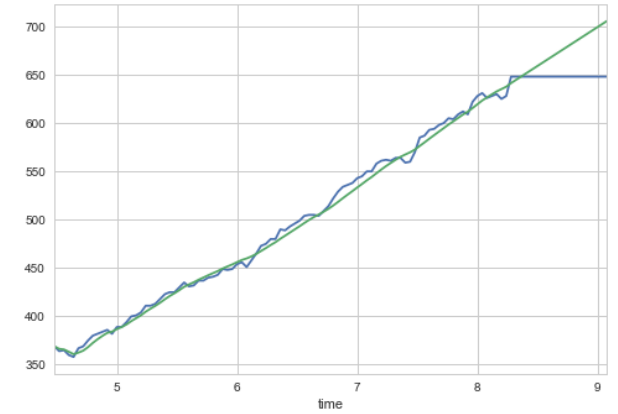
```
In [9]: data_one['matched'].plot()
```

```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1b3759afe80>
```



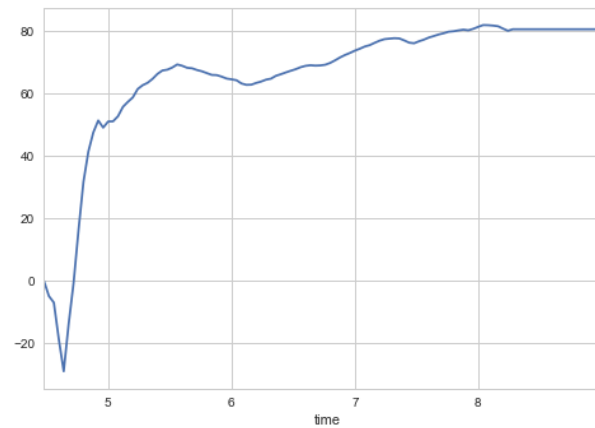
```
In [10]: data_one['x_min_d'].plot() # blue, measurement  
data_one['x_min'].plot() # green, filtered + predicted
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1b376ccbe48>
```



```
In [11]: data_one['vx_min'].plot()
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1b376e50f98>
```



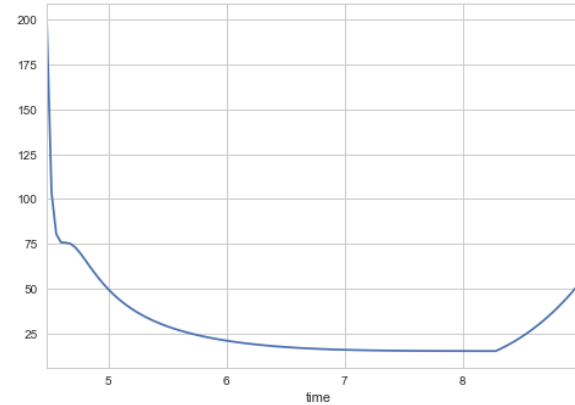
Pattern Filtering

Covariance matrix

0=location, 1=velocity

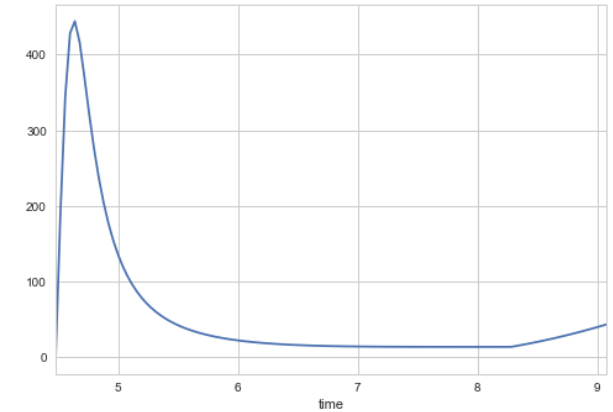
```
In [16]: data_one['sigma_x_min_00'].plot()
```

```
Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x1b3771859b0>
```



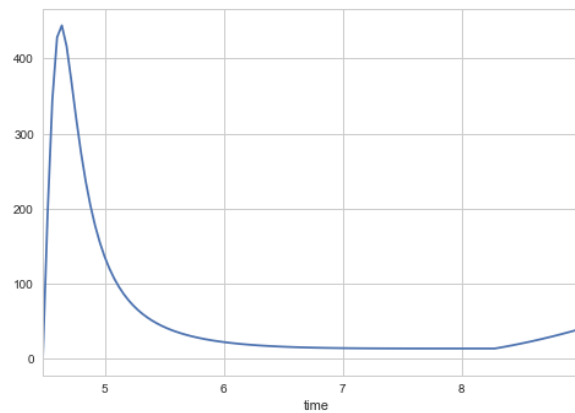
```
In [17]: data_one['sigma_x_min_01'].plot()
```

```
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1b37726b5c0>
```



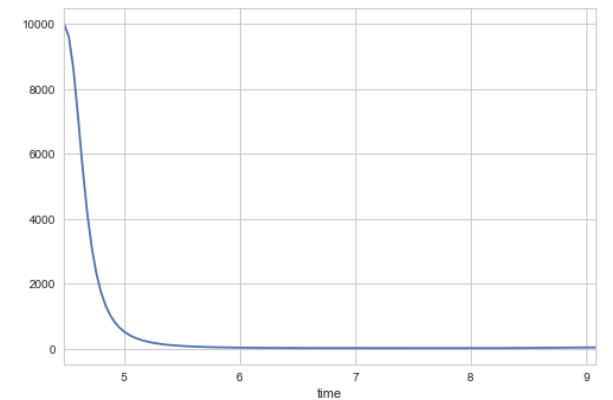
```
In [18]: data_one['sigma_x_min_10'].plot()
```

```
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1b3772b9da0>
```



```
In [19]: data_one['sigma_x_min_11'].plot()
```

```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1b377347358>
```



Matching / Confidence Level

Minimum confidence level to create a pattern (and body) was varied between 0 and 1. The number of bodies created was compared to the true number of objects in the video.

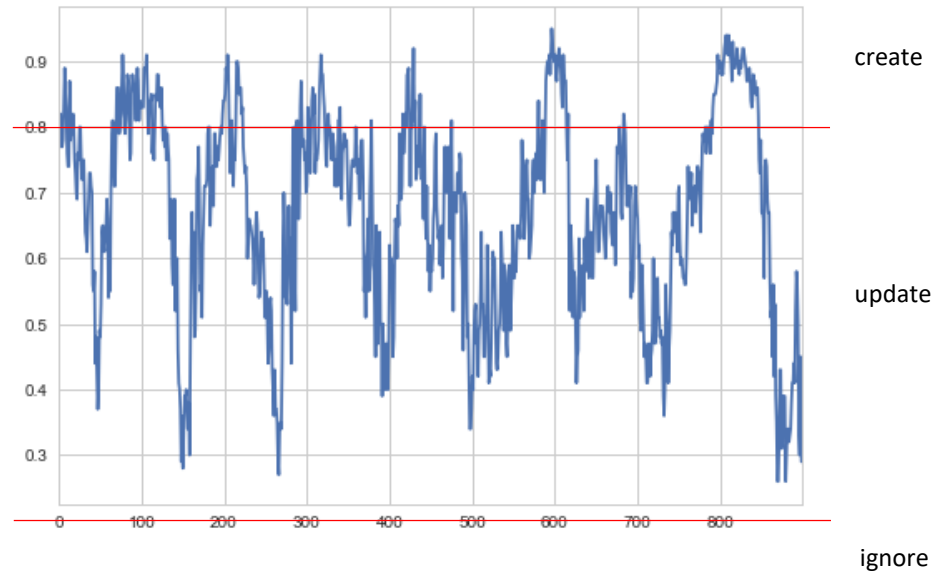
	A	B	C	D	E	F	G	H	I	J
1	Objects detected		Confidence level							
2	Video	Correct	0,00	0,20	0,40	0,60	0,80	0,90	0,95	1,00
3	CarsOnHighway001.mpg	39	49	49	39	36	34	32	32	0
4	Calf-2679.mp4	1	2	2	2	2	1	1	1	0
5	Dunes-7238.mp4	1	7	7	6	5	2	2	2	0
6	Sofa-11294.mp4	1	2	2	1	1	1	1	1	0
7	Cars133.mp4	5	9	9	6	5	5	5	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	1	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 pattern is between 0.8 and 0.9.

Matching / Confidence Level



Confidence level has dynamics



Different levels for creating and updating image object.

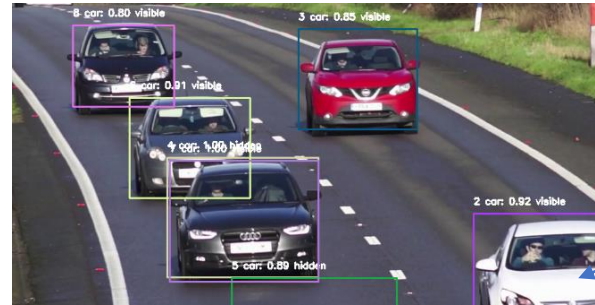
Hyperparameters:

- CONFIDENCE_LEVEL_CREATE (0.8)
- CONFIDENCE_LEVEL_UPDATE (0.2)

Matching / Border Behaviour

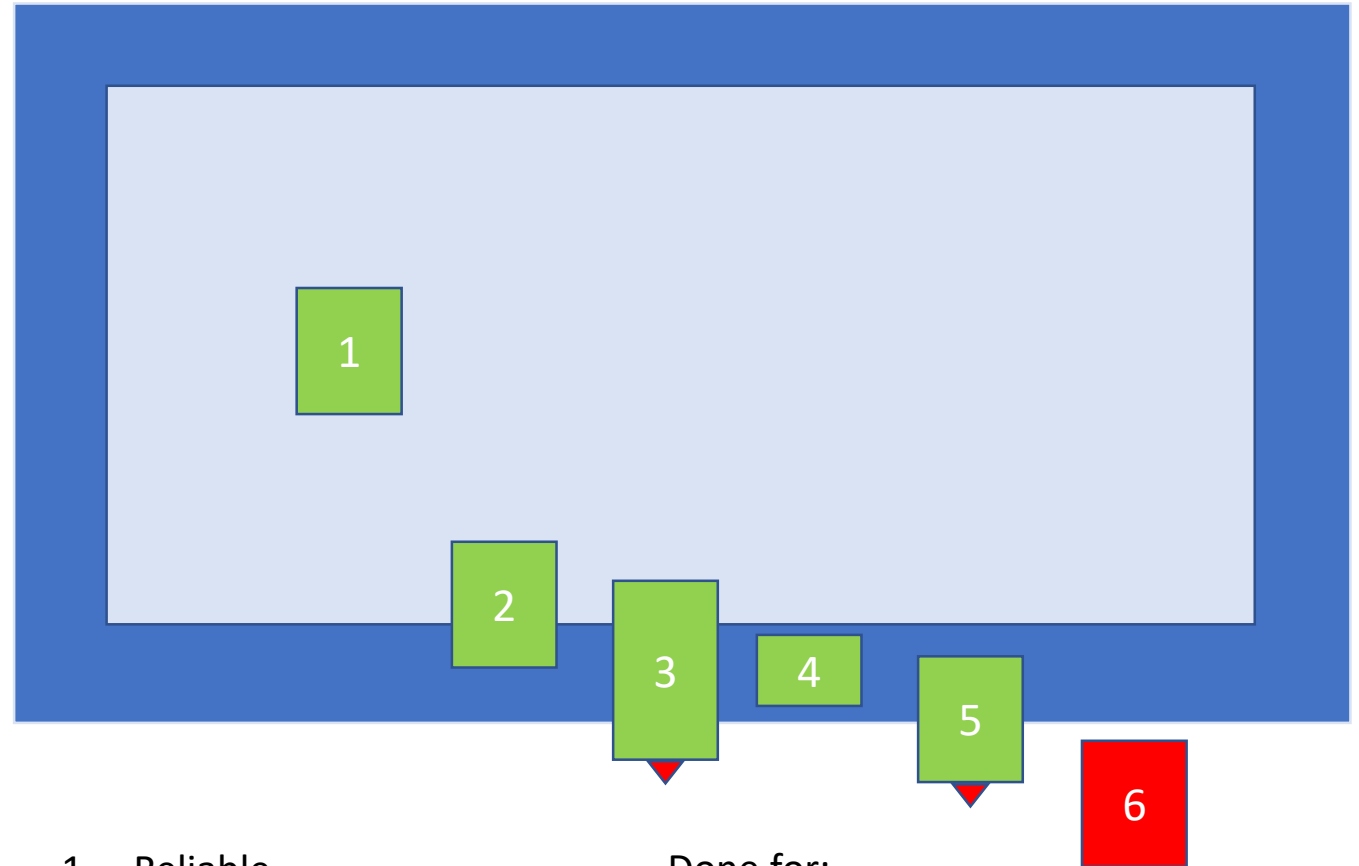
The problem:

Bounding box size and form are distorted near edges



Hyperparameter BORDER_WIDTH (30)

Matching / Border Behaviour



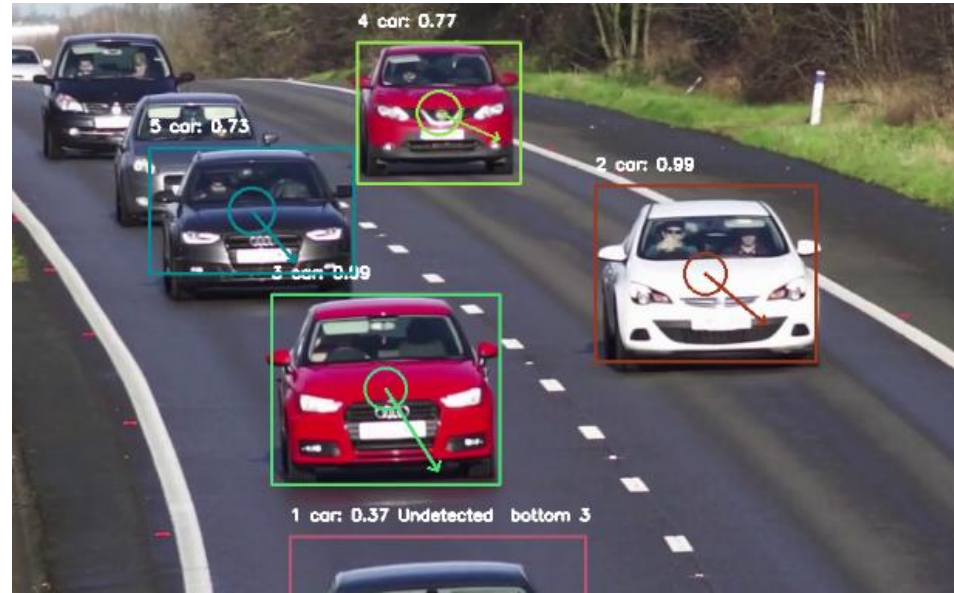
1. Reliable
2. Reliable
3. Unreliable
4. Reliable, not created
5. Unreliable, not created
6. Unreliable, removed

Done for:

- left
- right
- top
- bottom

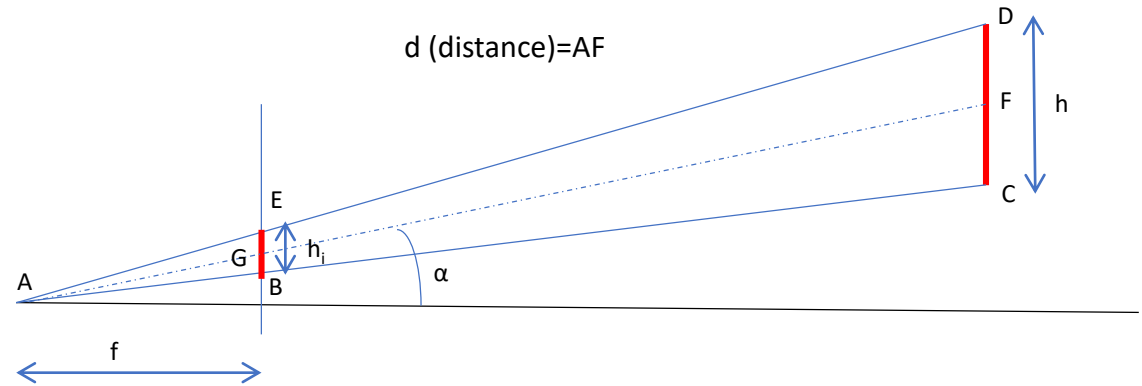
If a pattern touches 3 borders, it is removed
Reliable = body information updated

Matching / Pattern Retention



Patterns are removed if not detected in RETENTION_COUNT_MAX (30) successive frames.

Distance Estimation



Similar triangles AGE and AFD:

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

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

Distance Estimation

Similar equations for horizontal direction (β =azimuth):

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

s_w = sensor width (m)
 s_h = sensor height (m)
 p_w = image width (pixels)
 p_h = image height (pixels)
 l_i = object length (pixels)
 l = object length (m)
 f = focal length (m)
 α = altitude (rad)
 β = azimuth (rad)

Assumptions:

- equal vertical/horizontal pixel spacing
- optical axis in image center

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
 α = altitude (rad) = 0.0
 β = azimuth (rad) = 0.0

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

Distance Estimation

Question: How to compare pattern and body sizes for distance estimation?

Height or width alone might be misleading.

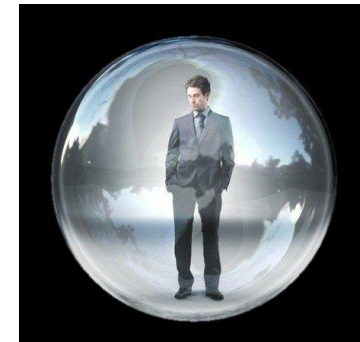
Some form of (3D) spatial simplification is needed, like

- cube
- rectangular prism
- cylinder
- **sphere** (probably the easiest math)

Uncertainty should be modeled.

Solution:

pattern circle <---> body sphere



Distance Estimation

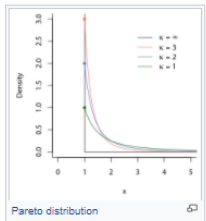
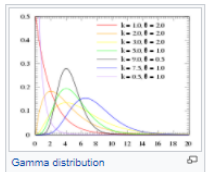
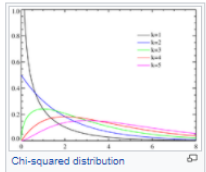
Body radius distribution

Distribution should

- be defined in $[0, \infty]$
- mode > 0
- simple
- skew controllable

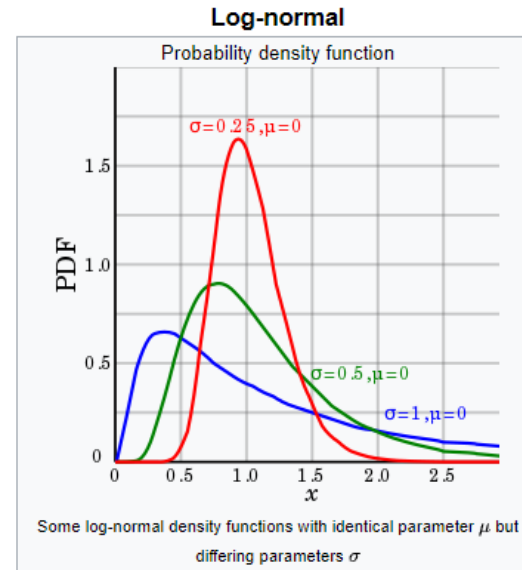
Supported on semi-infinite intervals, usually $[0, \infty)$ [\[edit\]](#)

- The Beta prime distribution
- The Birnbaum–Saunders distribution, also known as the fatigue life distribution, is a probability distribution used extensively in reliability applications to model failure times.
- The chi distribution
 - The noncentral chi distribution
- The chi-squared distribution, which is the sum of the squares of n independent Gaussian random variables. It is a special case of the Gamma distribution, and it is used in [goodness-of-fit tests in statistics](#).
 - The inverse-chi-squared distribution
 - The noncentral chi-squared distribution
 - The Scaled inverse chi-squared distribution
- The Dagum distribution
- The exponential distribution, which describes the time between consecutive rare random events in a process with no memory.
- The Exponential-logarithmic distribution
- The F-distribution, which is the distribution of the ratio of two (normalized) chi-squared-distributed random variables, used in the [analysis of variance](#). It is referred to as the [beta prime distribution](#) when it is the ratio of two chi-squared variates which are not normalized by dividing them by their numbers of degrees of freedom.
 - The noncentral F-distribution
- The folded normal distribution
- The Fréchet distribution
- The Gamma distribution, which describes the time until n consecutive rare random events occur in a process with no memory.
 - The Erlang distribution, which is a special case of the gamma distribution with integral shape parameter, developed to predict waiting times in [queueing systems](#)
 - The inverse-gamma distribution
- The Generalized gamma distribution
- The generalized Pareto distribution
- The Gamma/Gompertz distribution
- The Gompertz distribution
- The half-normal distribution
- Hotelling's T-squared distribution
- The inverse Gaussian distribution, also known as the Wald distribution
- The Lévy distribution
- The log-Cauchy distribution
- The log-Laplace distribution
- The log-logistic distribution
- The log-normal distribution, describing variables which can be modelled as the product of many small independent positive variables.
- The Lomax distribution
- The Mittag-Leffler distribution
- The Nakagami distribution
- The Pareto distribution, or "power law" distribution, used in the analysis of financial data and critical behavior.
- The Pearson Type III distribution
- The Phase-type distribution, used in [queueing theory](#)
- The [phased bi-exponential distribution](#) is commonly used in [pharmokinetics](#)
- The [phased bi-Weibull distribution](#)
- The Rayleigh distribution
- The Rayleigh mixture distribution
- The Rice distribution
- The shifted Gompertz distribution
- The type-2 Gumbel distribution
- The Weibull distribution or Rosin Rammler distribution, of which the [exponential distribution](#) is a special case, is used to model the lifetime of technical devices and is used to describe the [particle size distribution](#) of particles generated by grinding, [milling](#) and [crushing](#) operations.



Distance Estimation

Log-normal distribution for body radius



Used in the context of describing human height distribution

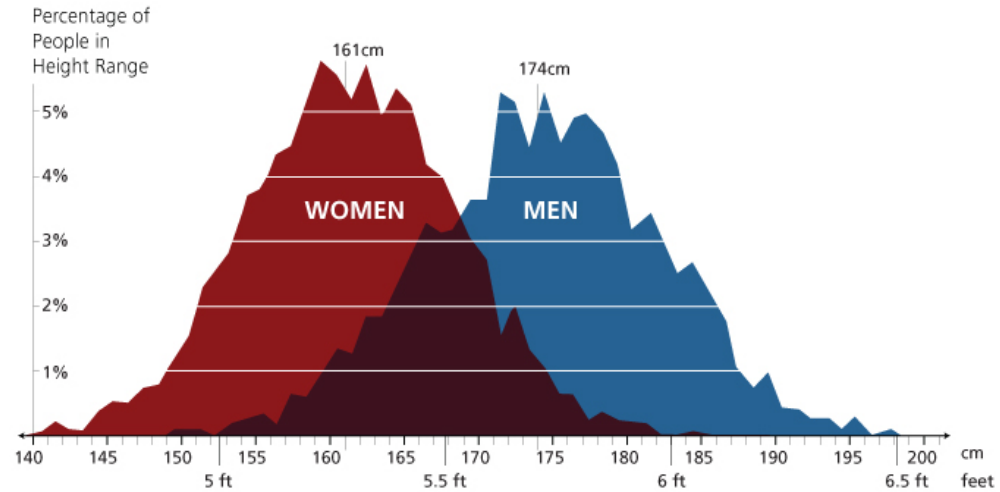
Notation	$\text{Lognormal}(\mu, \sigma^2)$
Parameters	$\mu \in (-\infty, +\infty)$, $\sigma > 0$
Support	$x \in (0, +\infty)$
PDF	$\frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}$
CDF	$\frac{1}{2} + \frac{1}{2} \operatorname{erf}\left[\frac{\ln x - \mu}{\sqrt{2}\sigma}\right]$
Mean	$\exp\left(\mu + \frac{\sigma^2}{2}\right)$
Median	$\exp(\mu)$
Mode	$\exp(\mu - \sigma^2)$
Variance	$[\exp(\sigma^2) - 1] \exp(2\mu + \sigma^2)$
Skewness	$(e^{\sigma^2} + 2)\sqrt{e^{\sigma^2} - 1}$
Ex. kurtosis	$\exp(4\sigma^2) + 2\exp(3\sigma^2) + 3\exp(2\sigma^2) - 6$
Entropy	$\log(\sigma e^{\mu + \frac{1}{2}\sigma^2} \sqrt{2\pi})$
MGF	defined only for numbers with a non-positive real part, see text
CF	representation $\sum_{n=0}^{\infty} \frac{(it)^n}{n!} e^{n\mu + n^2\sigma^2/2}$ is asymptotically divergent but sufficient for numerical purposes
Fisher information	$\begin{pmatrix} 1/\sigma^2 & 0 \\ 0 & 1/2\sigma^4 \end{pmatrix}$

Distance Estimation

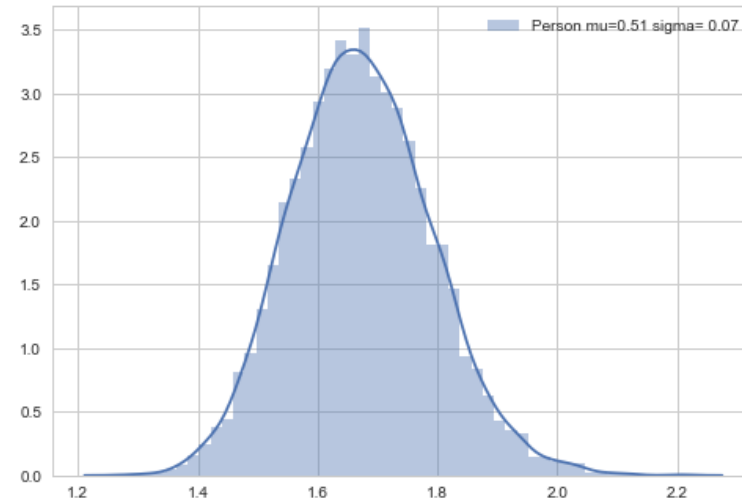
Example: Person

Height of Adult Women and Men

Within-group variation and between-group overlap are significant

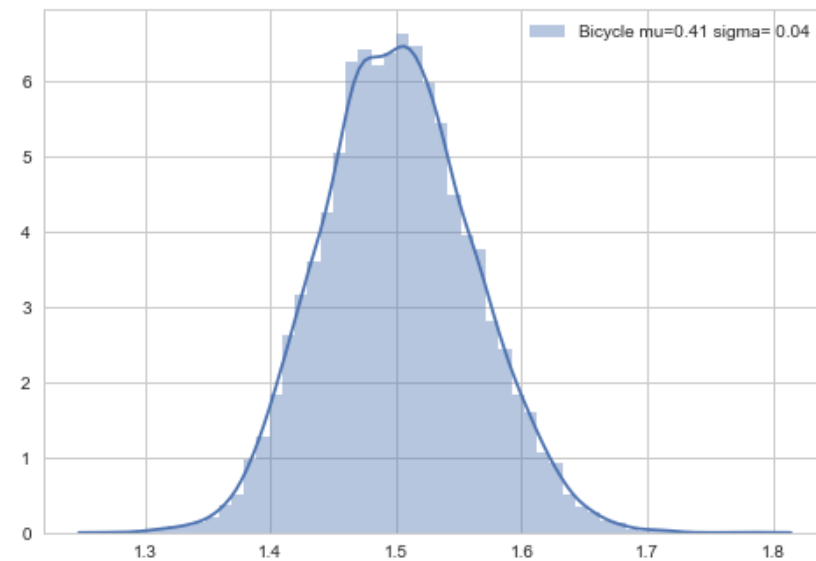
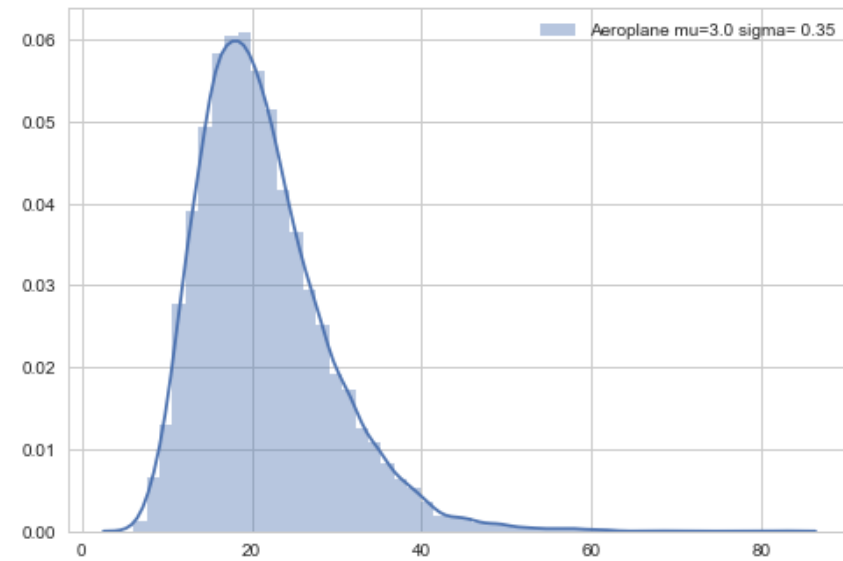


Data from U.S. CDC, adults ages 18-86 in 2007



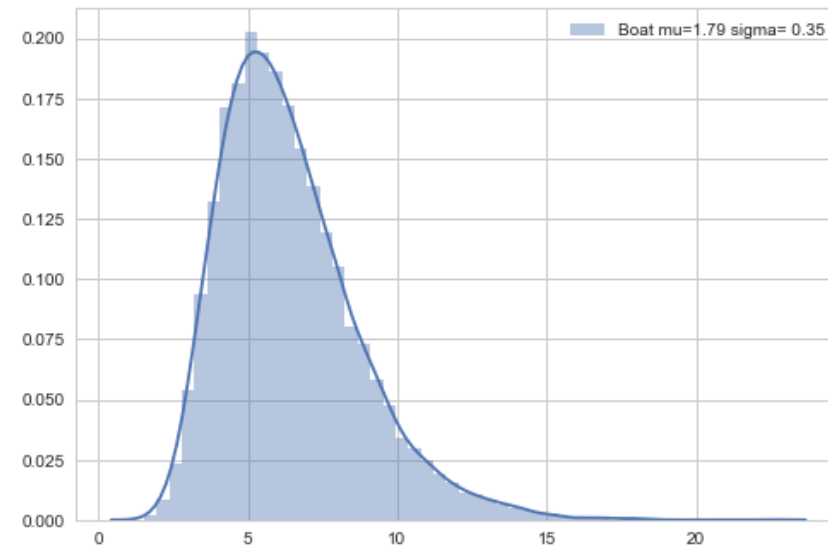
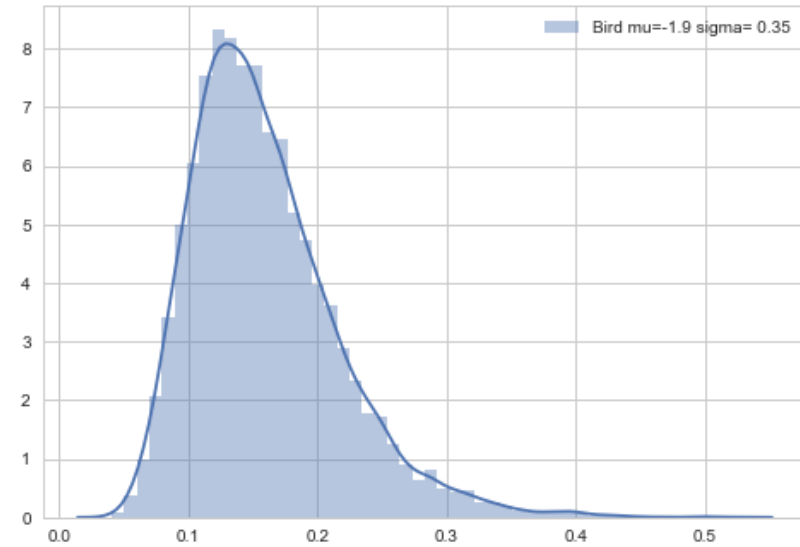
Distance Estimation

Bubble diameters ($2 \times \text{radius}$)



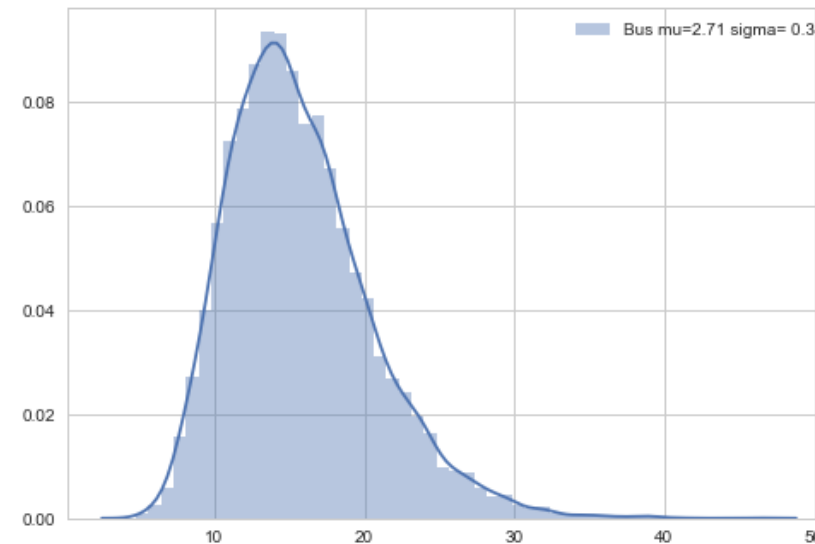
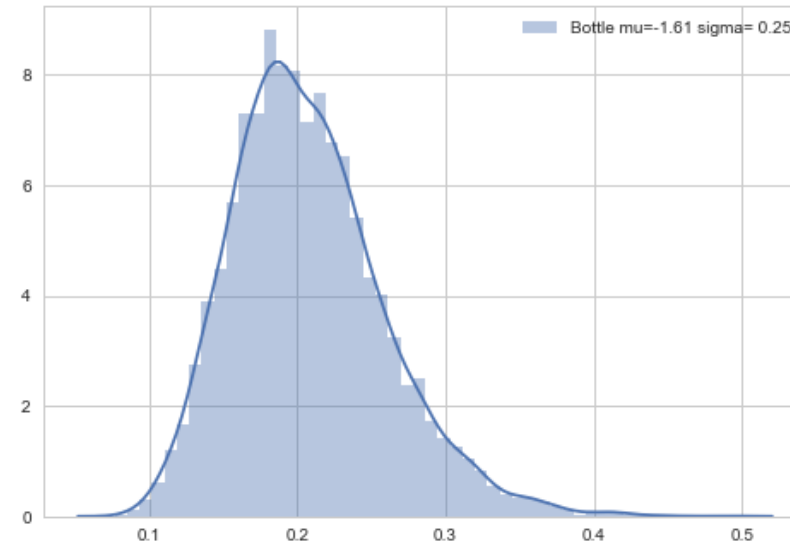
Distance Estimation

Bubble diameters ($2 \times \text{radius}$)



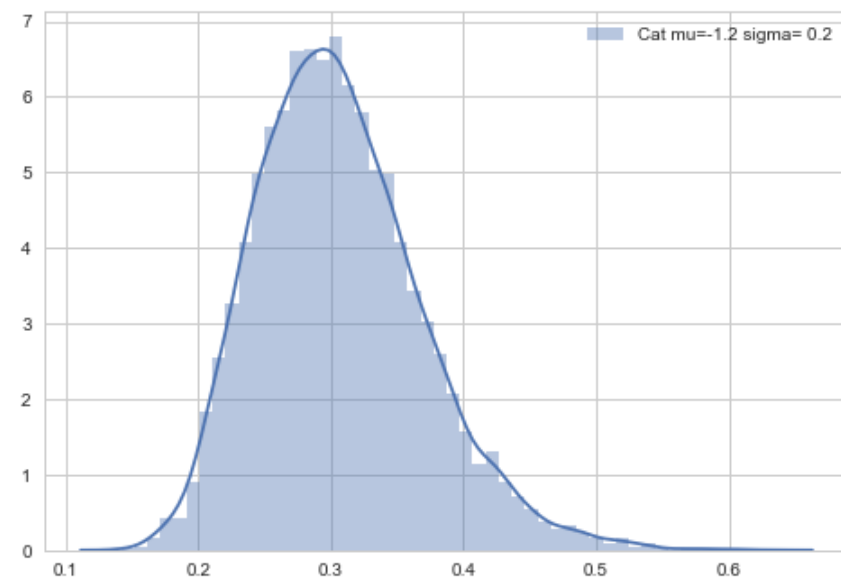
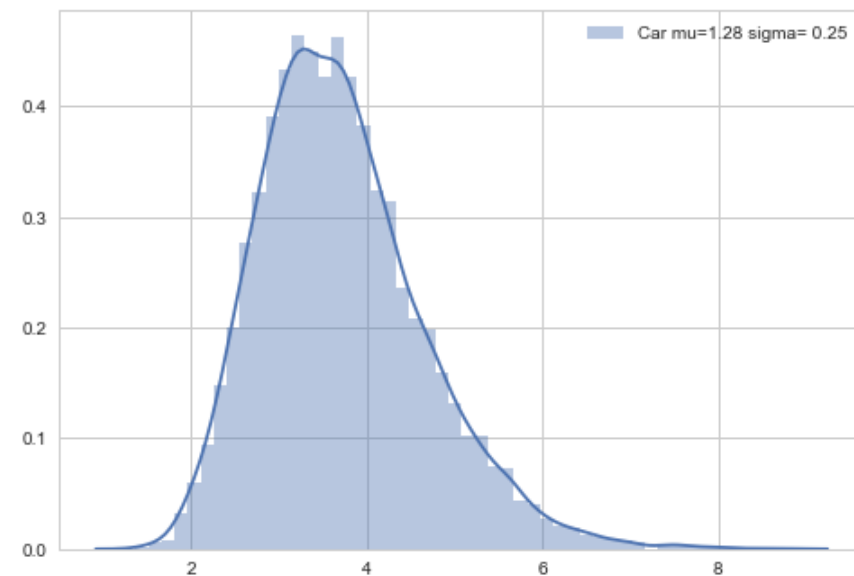
Distance Estimation

Bubble diameters ($2 \times \text{radius}$)



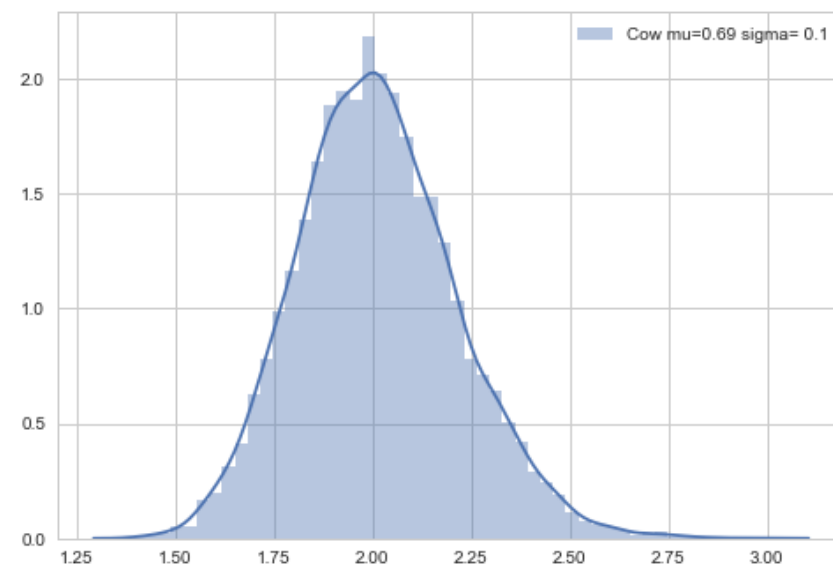
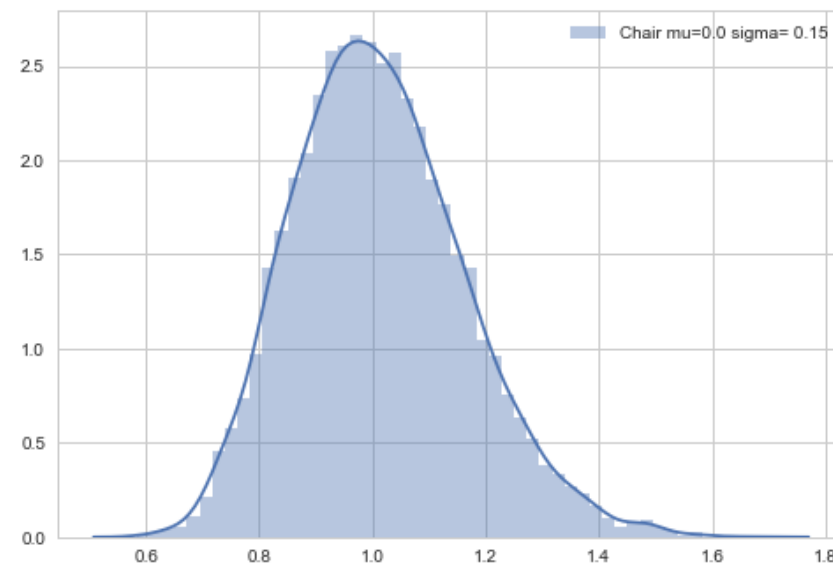
Distance Estimation

Bubble diameters ($2 \times \text{radius}$)



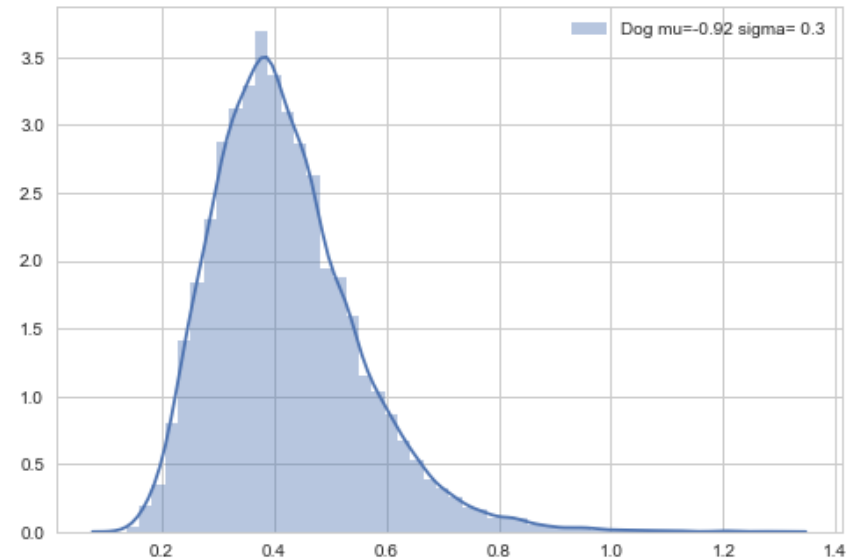
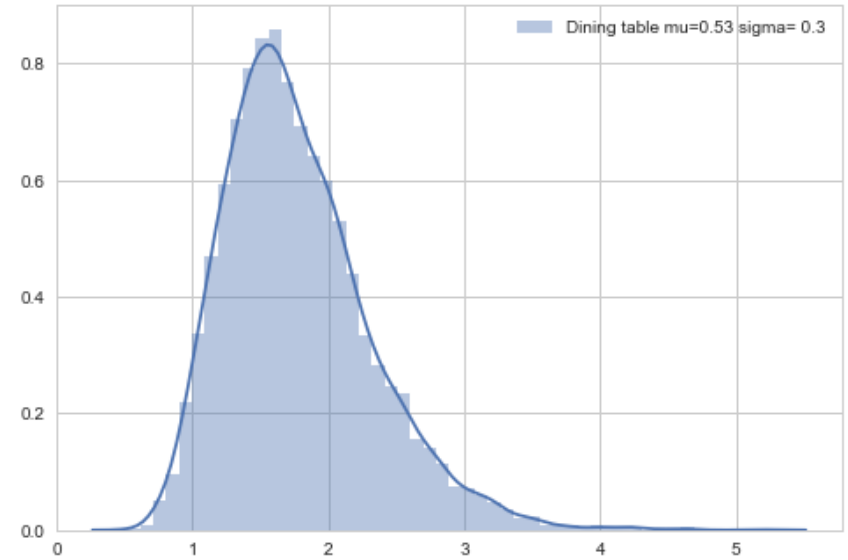
Distance Estimation

Bubble diameters ($2 \times \text{radius}$)



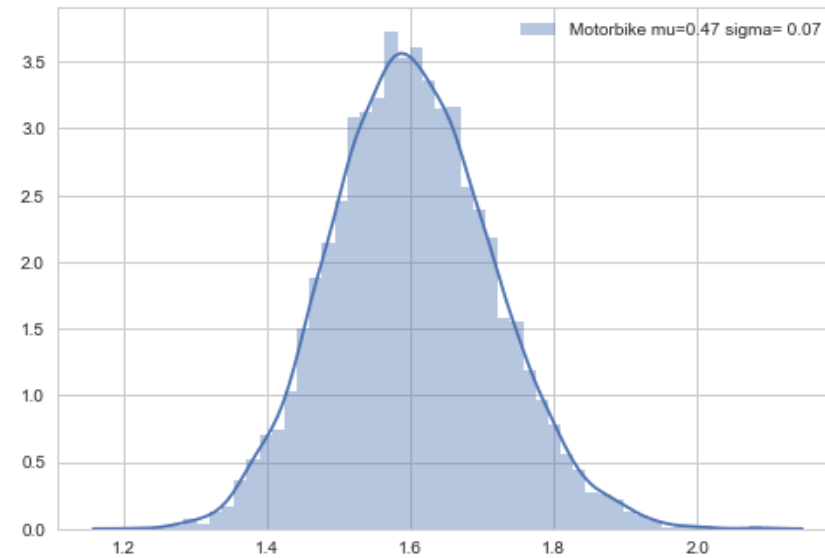
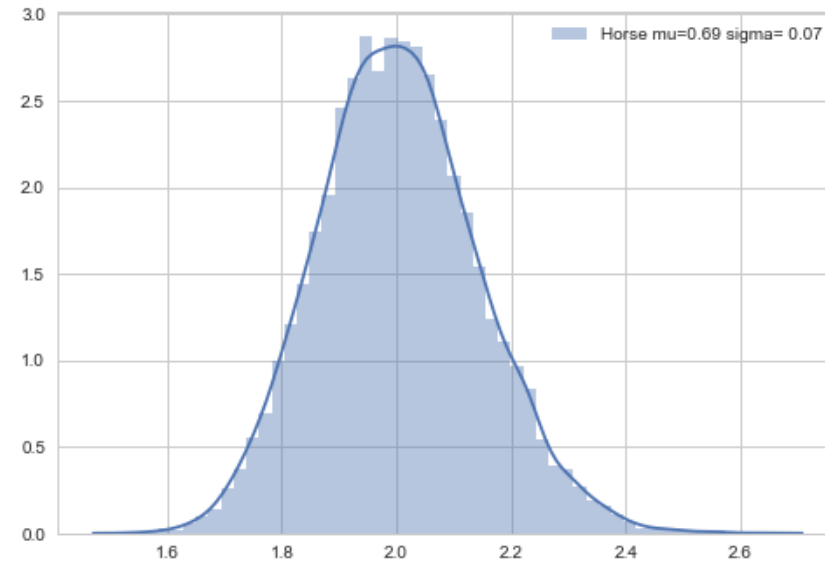
Distance Estimation

Bubble diameters ($2 \times \text{radius}$)



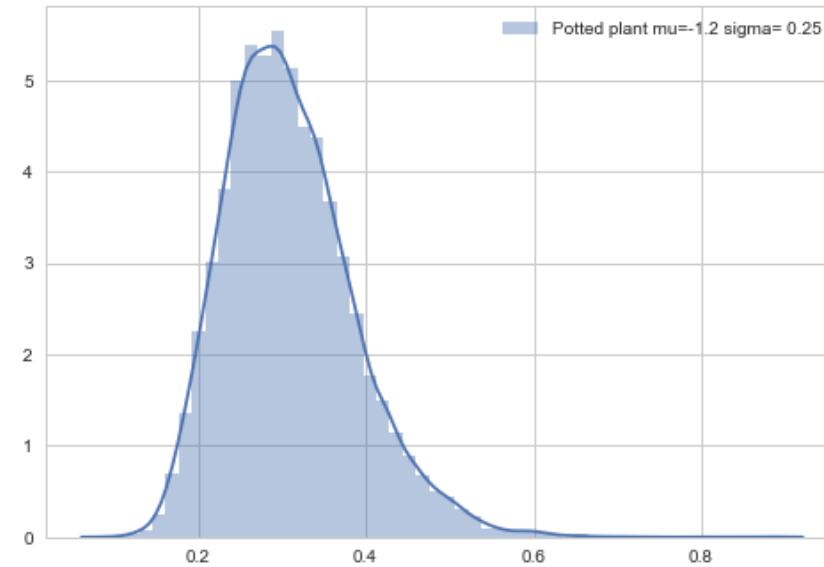
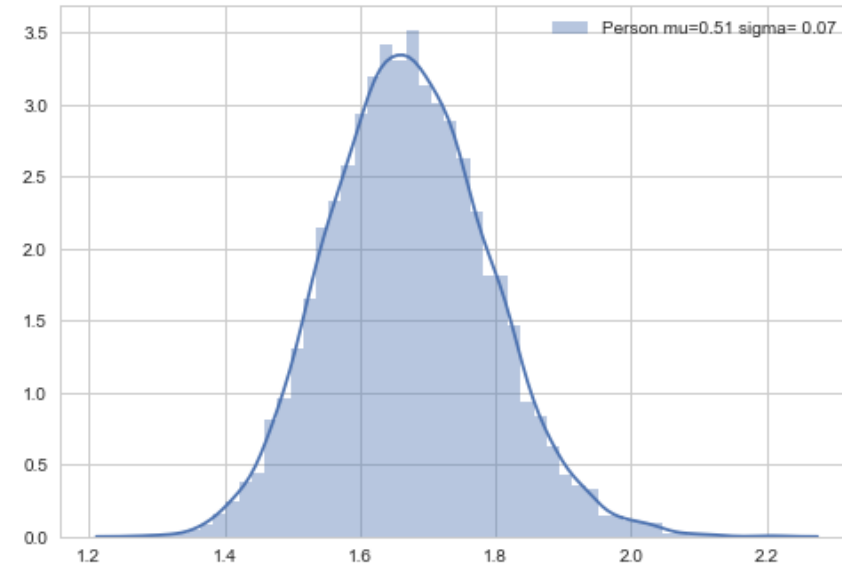
Distance Estimation

Bubble diameters ($2 \times \text{radius}$)



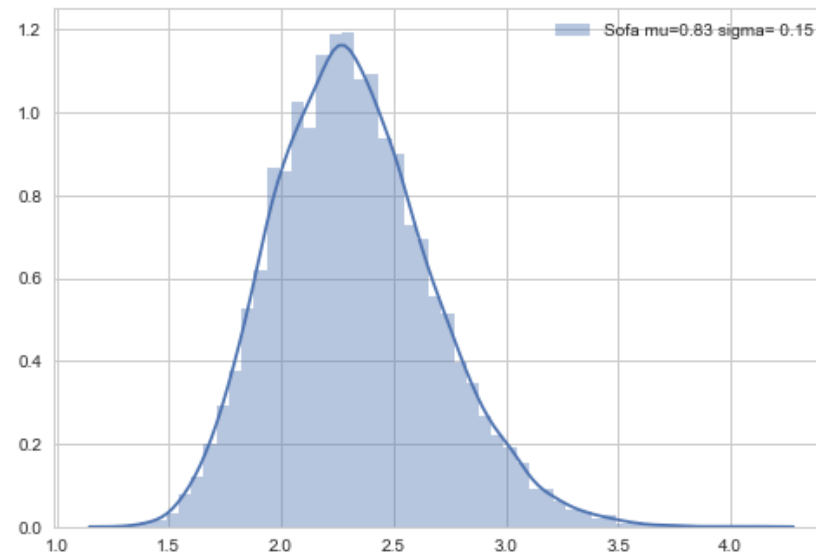
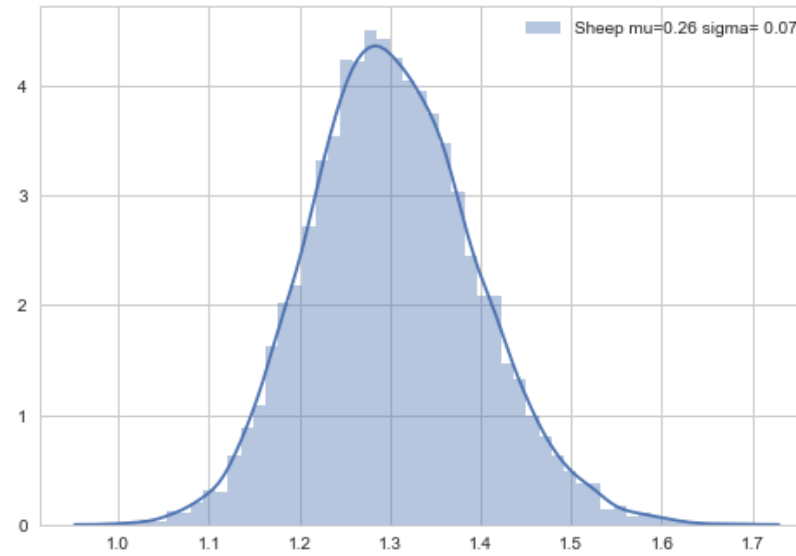
Distance Estimation

Bubble diameters ($2 \times \text{radius}$)



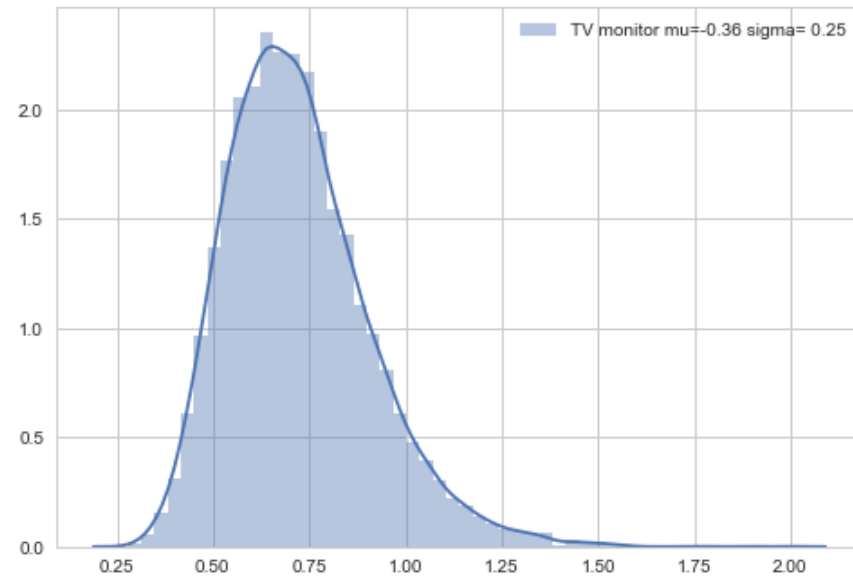
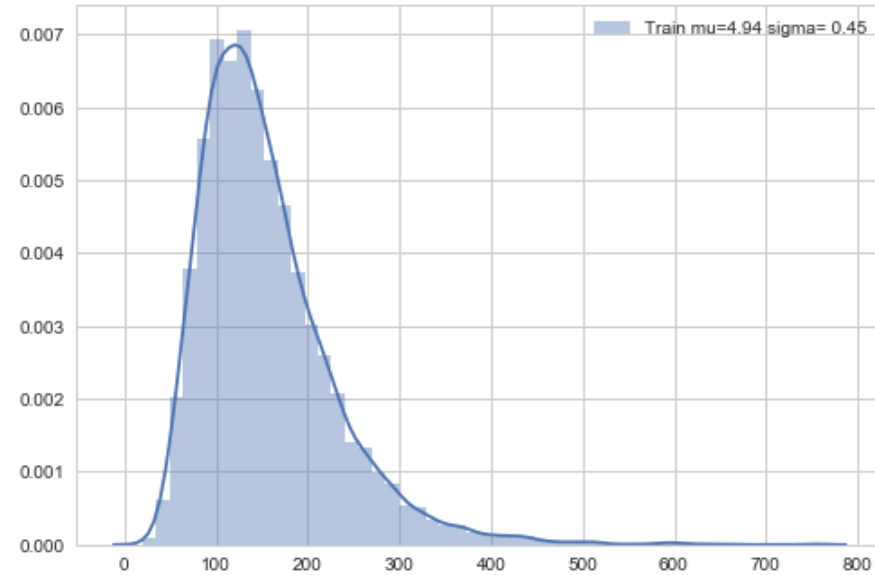
Distance Estimation

Bubble diameters ($2 \times \text{radius}$)



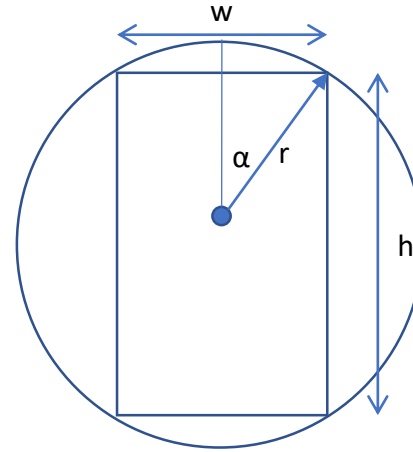
Distance Estimation

Bubble diameters ($2 \times \text{radius}$)



Distance Estimation

Radius of enclosing circle will be used for pattern



From bounding box coordinates to radius:

$$r = \sqrt{\left(\frac{w}{2}\right)^2 + \left(\frac{h}{2}\right)^2}$$

$$h = (ymax - ymin) \quad c_y = (ymax + ymin)/2$$

$$w = (xmax - xmin) \quad c_x = (xmax + xmin)/2$$

Distance Estimation

Distance estimation using pattern circle and body sphere

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

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)

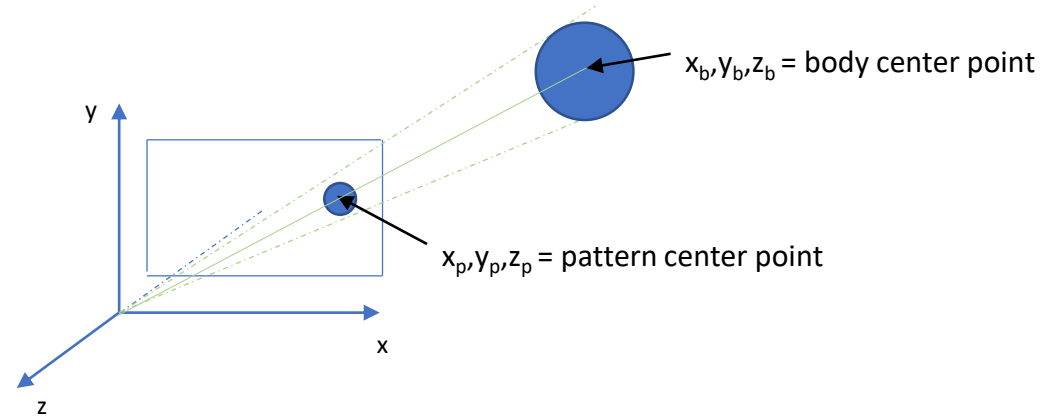
β = azimuth (rad)

Distance Estimation

Remarks:

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

3D Projection



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

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

Body center will be on the line:

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

The distance to the body is:

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

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

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

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)
 α = altitude (rad)

3D Projection

So: $t^2 * (x_p^2 + y_p^2 + z_p^2) = d^2$

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

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

Where:

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

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

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

3D Projection

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
 r_i = pattern radius (pixels) = 100
 r = body radius (m) = 1.0 m
 f = focal length (m) = 0.050 m
 x_p = 1200
 y_p = 2000



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

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

$$\alpha = \arctan(y_p/f) = 0.0445 \quad \beta = \arctan(x_p/f) = -0.2374$$

$$d = \frac{f * h}{\cos(\alpha) * \cos(\beta) * h_i * s_h/p_h} = \frac{0.050 * 1}{\cos(0.0445) * \cos(-0.2374) * 100 * 0.0240/4912} = 105.39$$

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

$$(x_b, y_b, z_b) = t * (x_p, y_p, z_p) = 2.0468e+03 * (-0.0121, 0.0022, -0.0500) = (-24.7593, 4.5602, -102.3389)$$

Body Filtering

General

- Enables prediction, including collision detection
- 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

Body Filtering

Body Kalman Filtering

Body center point location

State vector s :

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

where

(x, y, z) = location of the body center point

(v_x, v_y, v_z) = velocity of the body

State equation in differential form:

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

Body Filtering

State equation in difference form:

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

$$A = \begin{bmatrix} 1 & 0 & 0 & \Delta & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where Δ is the time increment and ϵ Gaussian noise with covariance R:

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

Measurement equation:

$$z(k) = C * s(k) + \delta(k)$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Where δ is Gaussian noise with covariance matrix Q:

$$Q = \begin{bmatrix} 200 & 0 & 0 \\ 0 & 200 & 0 \\ 0 & 0 & 200 \end{bmatrix}$$

Body Filtering

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.

$$\Sigma(0) = \begin{bmatrix} 100\,000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100\,000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100\,000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100\,000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100\,000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100\,000 \end{bmatrix}$$

Kalman filter update:

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

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

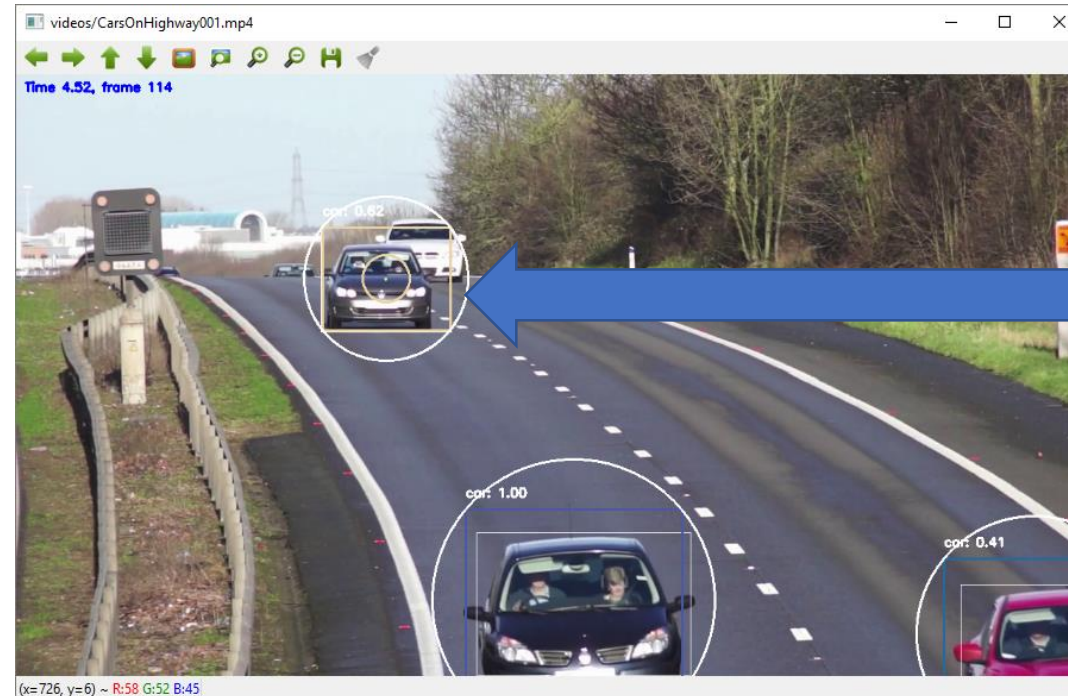
$$K(k) = \Sigma_1(k) * C^T (C * \Sigma_1(k) * C^T + Q)^{-1}$$

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

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

Body Filtering

Example 1

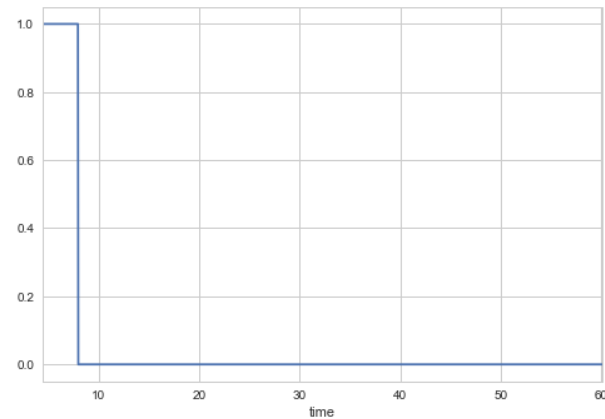


```
Event.txt
630 4.440,3,2061577943248,Detection created
631 4.440,2,2061579199488,Pattern removed
632 4.480,3,2061577756008,Detection created
633 4.480,3,2061577754272,Detection created
634 4.480,3,2061577755112,Detection create
635 4.480,1,2061577797816,Body created
636 4.480,2,206157776432,Pattern created
637 4.520,3,2061577844944,Detection created
```

Body Filtering

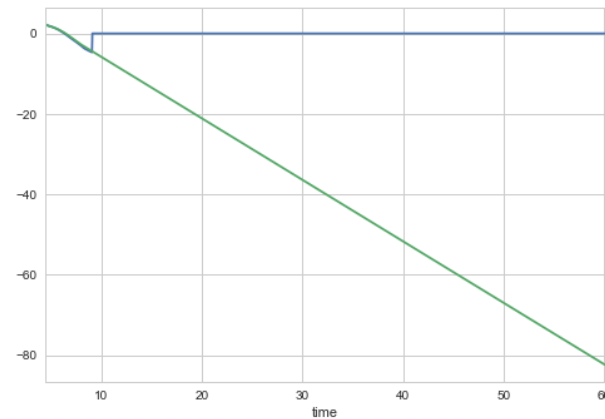
```
In [5]: data_one['status'].plot()
```

```
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x2e5b3fa4e10>
```



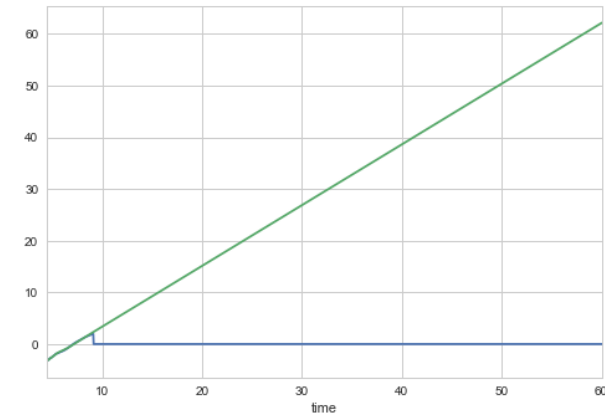
```
In [8]: data_one['y_pattern'].plot() # blue, measurement  
data_one['y'].plot() # green, filtered + predicted
```

```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x2e5b4125470>
```



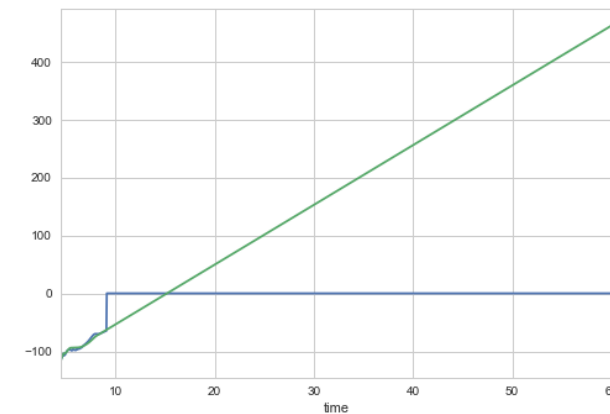
```
In [7]: data_one['x_pattern'].plot() # blue, measurement  
data_one['x'].plot() # green, filtered + predicted
```

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2e5b4111c50>
```



```
In [9]: data_one['z_pattern'].plot() # blue, measurement  
data_one['z'].plot() # green, filtered + predicted
```

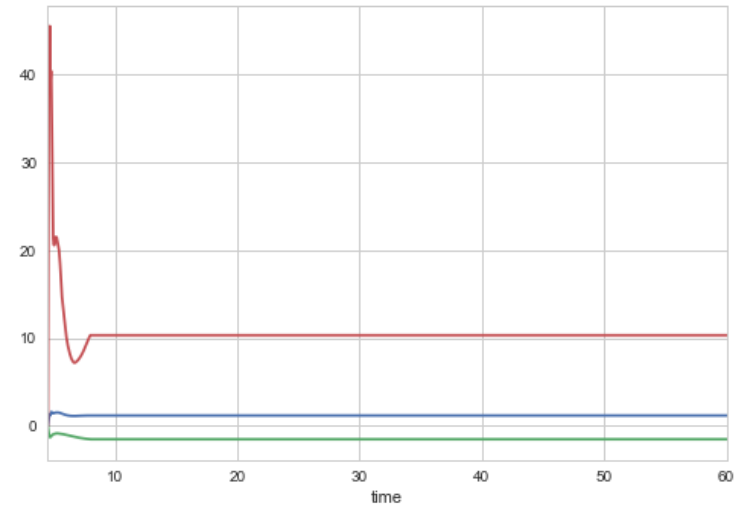
```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x2e5b41ab6d8>
```



Body Filtering

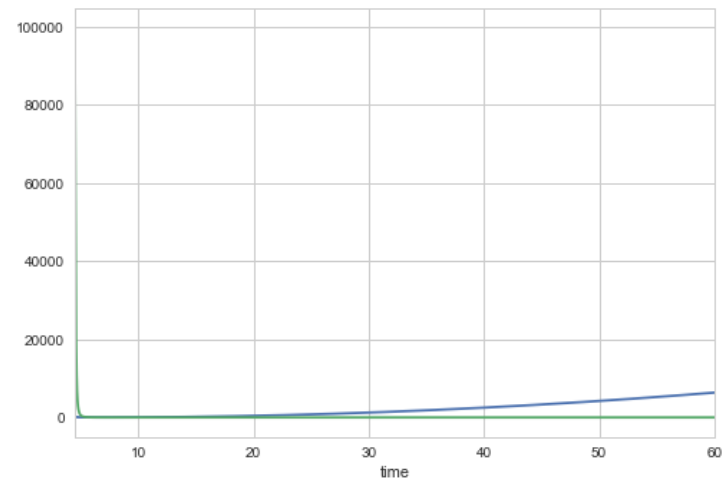
```
In [10]: data_one['vx'].plot() # blue  
data_one['vy'].plot() # green  
data_one['vz'].plot() # red
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x2e5b424cac8>



```
In [11]: data_one['sigma_00'].plot() # blue, x location variance  
data_one['sigma_33'].plot() # green, x velocity variance
```

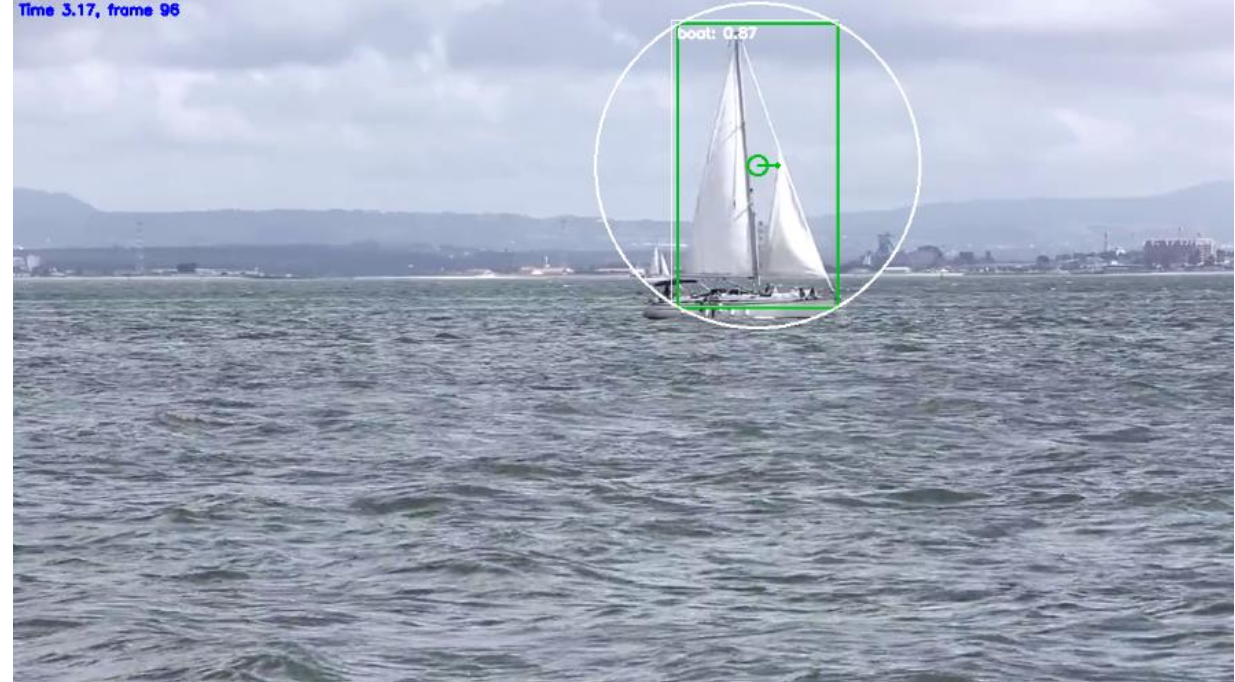
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x2e5b55b6ac8>



Body Filtering

Example 2

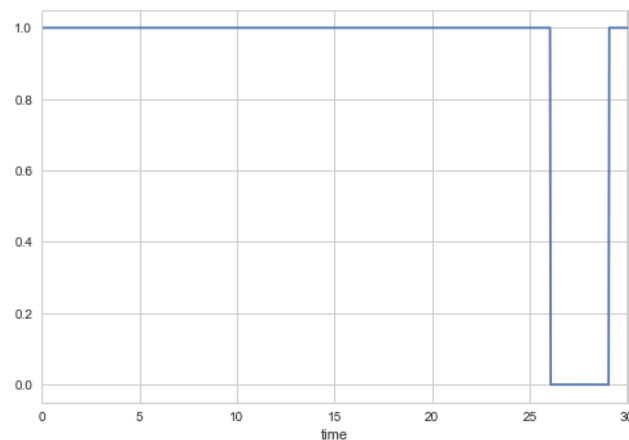
Time 3.17, frame 96



Body Filtering

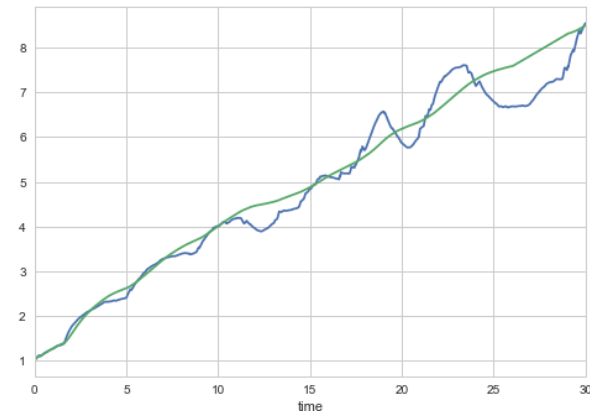
```
In [5]: data_one['status'].plot()
```

```
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x1a52aca5438>
```



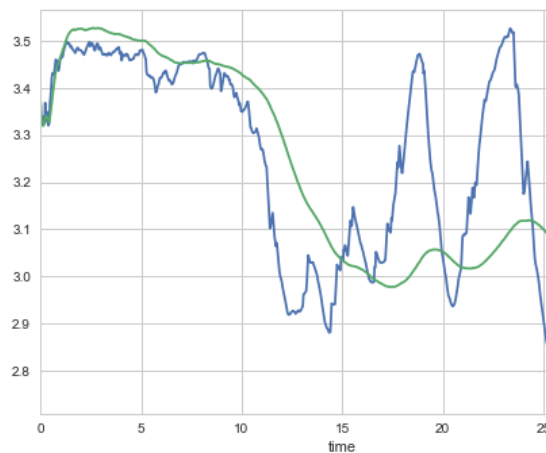
```
In [6]: data_one['x_pattern'].plot() # blue, measurement  
data_one['x'].plot() # green, filtered + predicted
```

```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1a52bf61470>
```



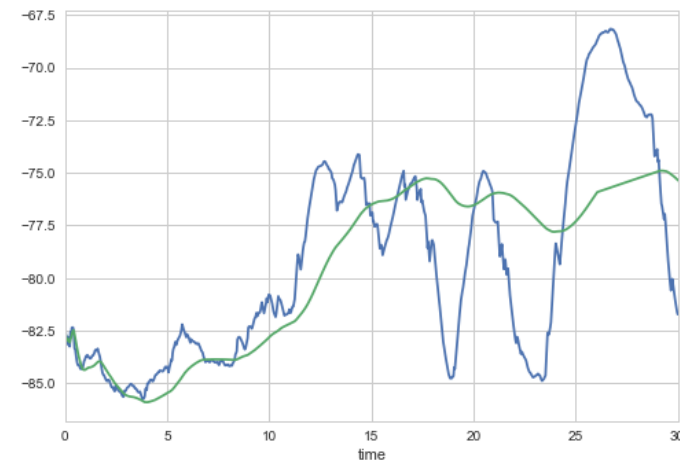
```
In [7]: data_one['y_pattern'].plot() # blue, measurement  
data_one['y'].plot() # green, filtered + predicted
```

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1a52ad49>
```



```
In [8]: data_one['z_pattern'].plot() # blue, measurement  
data_one['z'].plot() # green, filtered + predicted
```

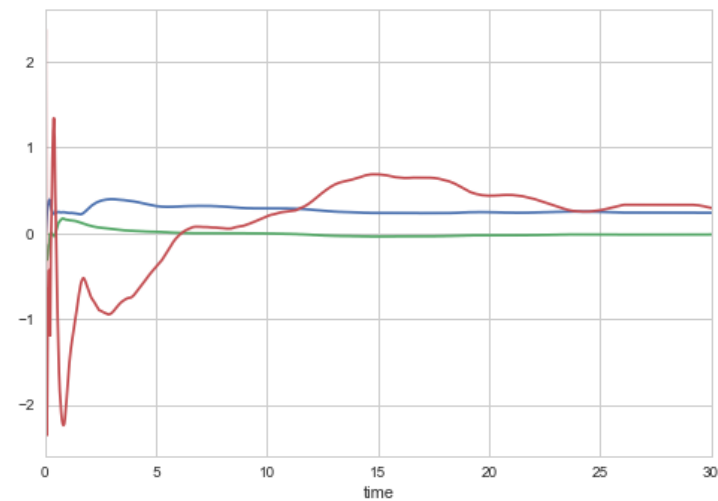
```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1a52c097cf8>
```



Body Filtering

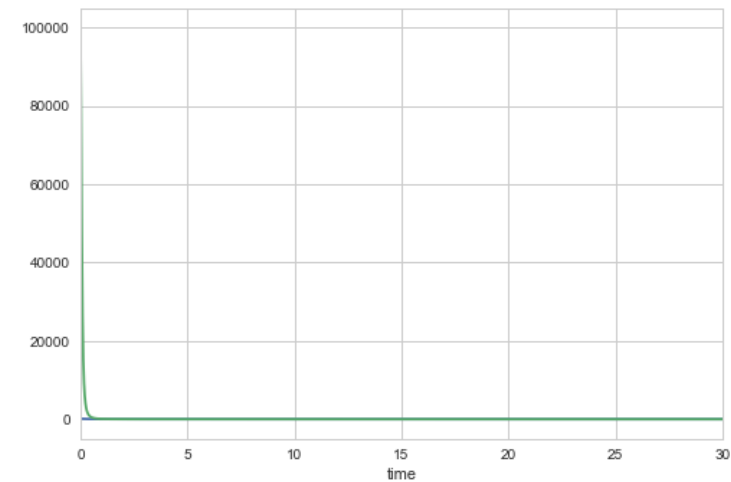
```
In [9]: data_one['vx'].plot() # blue  
data_one['vy'].plot() # green  
data_one['vz'].plot() # red
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1a52708b4a8>

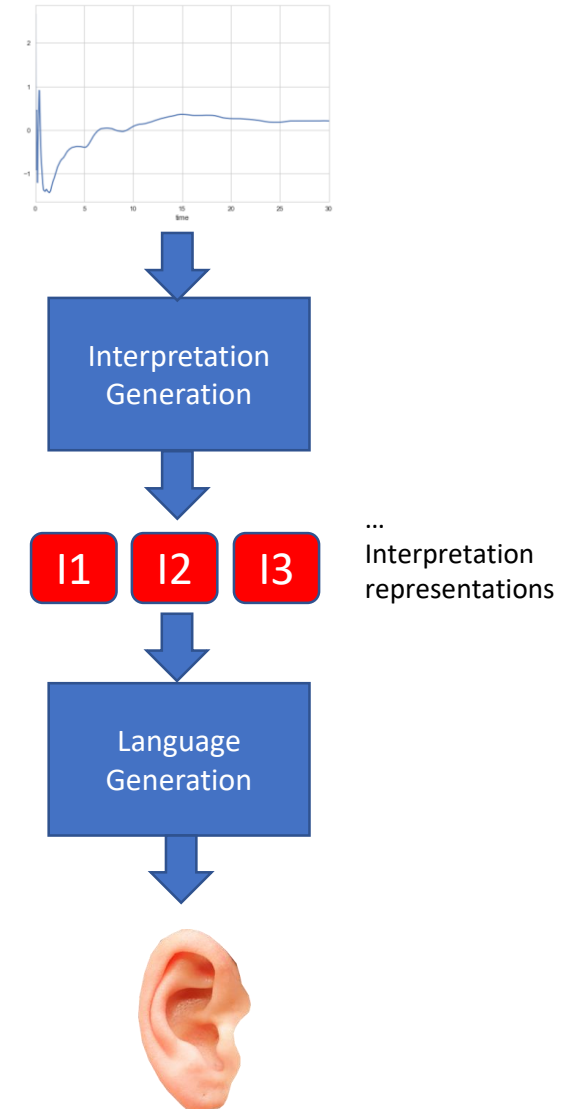
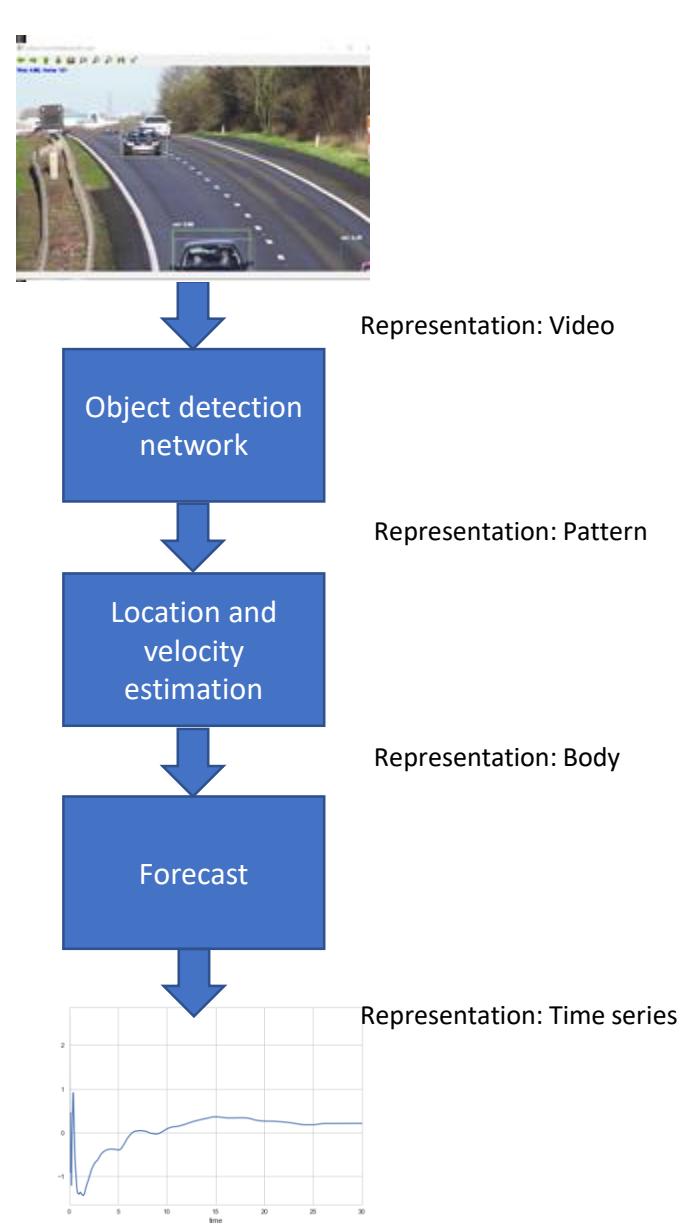


```
In [10]: data_one['sigma_00'].plot() # blue, x location variance  
data_one['sigma_33'].plot() # green, x velocity variance
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1a52c184438>

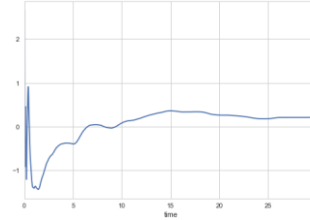


Representations for Interpretation



Representations for Interpretation

Example for interpretation representation: collision detection



Time series forecast for body locations



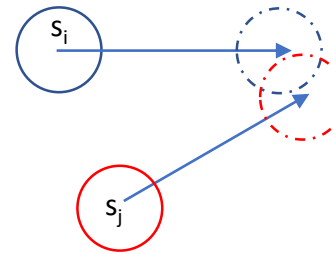
Interpretation
Generation



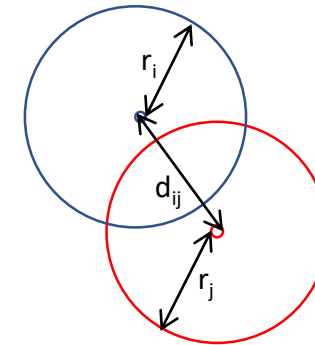
$$\begin{bmatrix} 0 & p_{12} & p_{13} & p_{14} & \dots & p_{1n} \\ p_{21} & 0 & p_{23} & p_{24} & \dots & p_{2n} \\ p_{31} & p_{32} & 0 & p_{34} & \dots & p_{3n} \\ p_{41} & p_{42} & p_{43} & 0 & \dots & p_{4n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ p_{n1} & p_{n2} & p_{n3} & p_{n4} & \dots & 0 \end{bmatrix}$$

(Symmetric) collision probability matrix
 p_{nm} = probability that bodies n and m will collide

Collision Detection



$$\begin{bmatrix} x_i(t) \\ y_i(t) \\ z_i(t) \end{bmatrix} = \begin{bmatrix} x_i(0) \\ y_i(0) \\ z_i(0) \end{bmatrix} + t * \begin{bmatrix} v_{i,x}(0) \\ v_{i,y}(0) \\ v_{i,z}(0) \end{bmatrix}$$



Collision C_{ij} if $d_{ij} < r_i + r_j$

$$d_{ij}(t) = \sqrt{(x(t)_i - x(t)_j)^2 + (y(t)_i - y(t)_j)^2 + (z(t)_i - z(t)_j)^2}$$

Random variable

$$r = \begin{bmatrix} x_1(0) \\ y_1(0) \\ z_1(0) \\ v_{1,x}(0) \\ v_{1,y}(0) \\ v_{1,z}(0) \\ r_1 \\ \dots \\ x_n(0) \\ y_n(0) \\ z_n(0) \\ v_{n,x}(0) \\ v_{n,y}(0) \\ v_{n,z}(0) \\ r_n \\ t \end{bmatrix}$$

t: uniform distribution on $[0, t_{\text{end}}]$

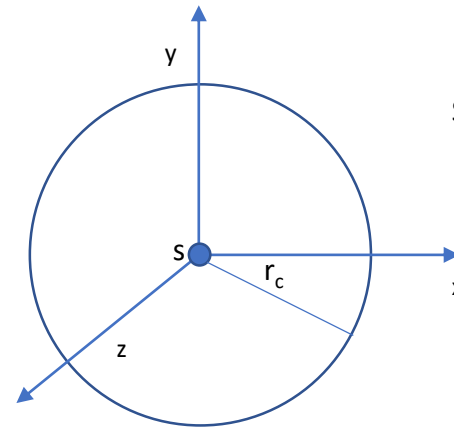
Sampling: $p_{ij} = E\{C_{ij}\} = \sum_{k=1}^m \frac{\delta(C_{i,j})}{m}$

Open question:

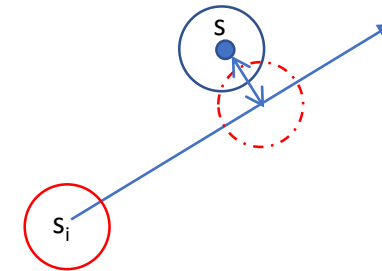
- 1) Too many dimensions?
- 2) More efficient sampling with MCMC / Metropolis-Hastings?

Collision Detection

Simpler case: Collision with the observer



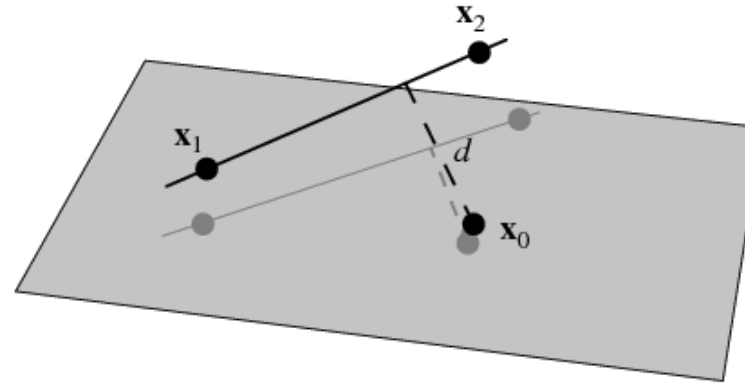
$S=[0,0,0,0,0,0]^T$, deterministic, $r = r[\text{class person}]$



$$r = \begin{bmatrix} x_i(0) \\ y_i(0) \\ z_i(0) \\ v_{i,x}(0) \\ v_{i,y}(0) \\ v_{i,z}(0) \\ r_i \\ r \end{bmatrix}$$

For each body i , a random vector r is sampled and minimum distance to the observer calculated. If the distance is less than r_1+r , collision occurred. The proportion on collisions to all cases is the probability estimate for the body/observer collision.

Collision Detection



$$t = - \frac{(\mathbf{x}_1 - \mathbf{x}_0) \cdot (\mathbf{x}_2 - \mathbf{x}_1)}{|\mathbf{x}_2 - \mathbf{x}_1|^2}$$

$$d = \frac{|(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_1 - \mathbf{x}_0)|}{|\mathbf{x}_2 - \mathbf{x}_1|}$$

$$= \frac{|(\mathbf{x}_0 - \mathbf{x}_1) \times (\mathbf{x}_0 - \mathbf{x}_2)|}{|\mathbf{x}_2 - \mathbf{x}_1|}$$

In our case:

$$\mathbf{x}_1 = P(0)$$

$$\mathbf{x}_2 = P(0) + V(0) \text{ (loc after one sec)}$$

$$\mathbf{x}_0 = 0$$

$$t = \frac{(P(0) - 0) \cdot (P(0) + V(0) - P(0))}{|P(0) + V(0) - P(0)|^2} = \frac{P(0) \cdot V(0)}{|V(0)|^2}$$

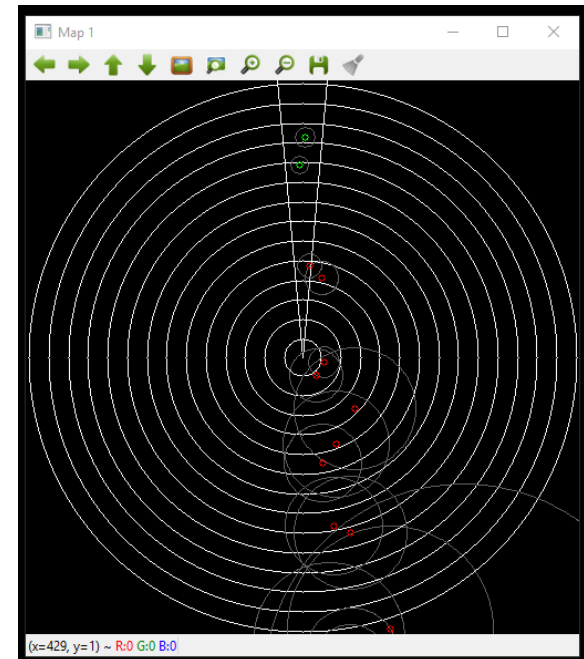
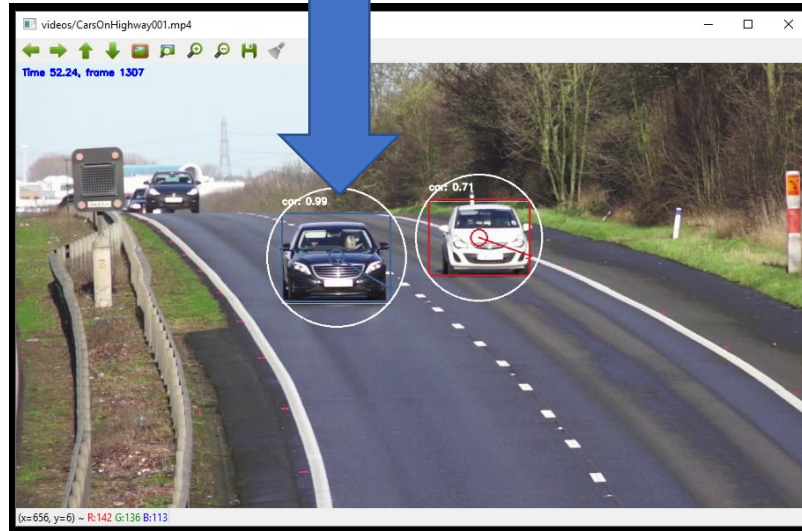
- ?

If $t \leq 0$, nearest point is $P(0)$

$$d = |P(0) + t * V(0)|$$

Collision Detection

Example

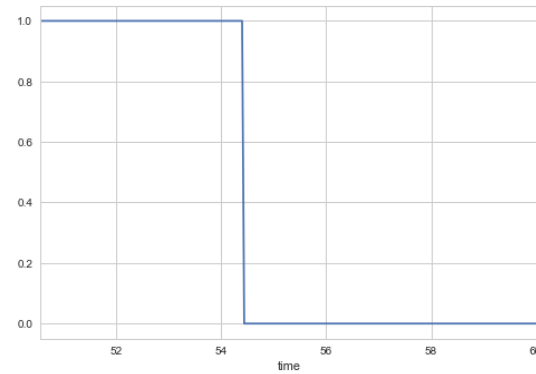


Collision Detection

Example

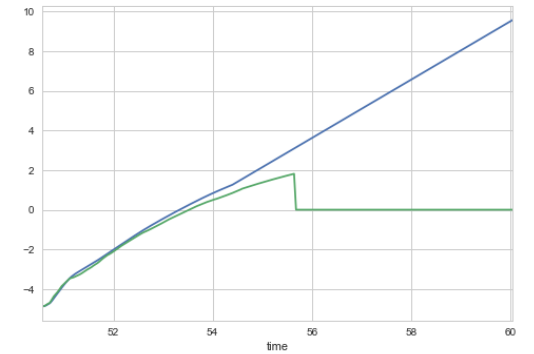
```
In [177]: data_one['status'].plot()
```

```
Out[177]: <matplotlib.axes._subplots.AxesSubplot at 0x1b238f06588>
```



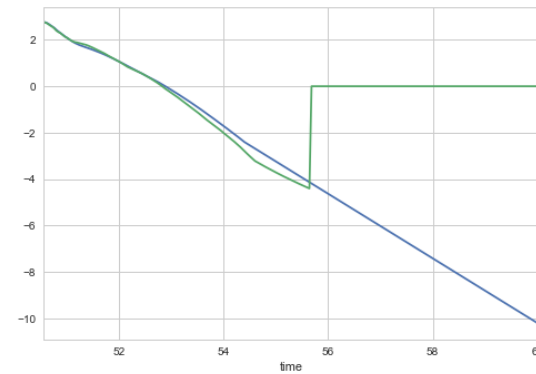
```
In [446]: data_one['x'].plot() # estimated, blue  
data_one['x_pattern'].plot() # measured, green
```

```
Out[446]: <matplotlib.axes._subplots.AxesSubplot at 0x1b241357278>
```



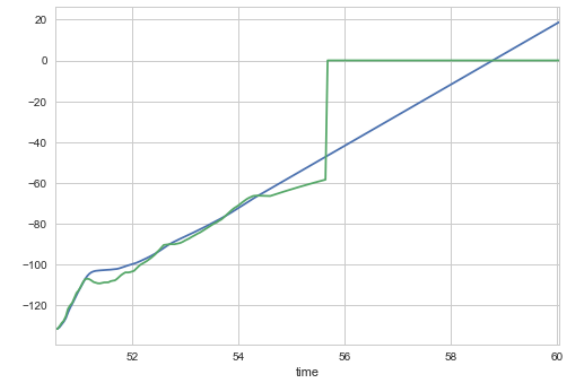
```
In [448]: data_one['y'].plot() # estimated, blue  
data_one['y_pattern'].plot() # measured, green
```

```
Out[448]: <matplotlib.axes._subplots.AxesSubplot at 0x1b24111898>
```



```
In [449]: data_one['z'].plot() # estimated, blue  
data_one['z_pattern'].plot() # measured, green
```

```
Out[449]: <matplotlib.axes._subplots.AxesSubplot at 0x1b2390bf0f0>
```

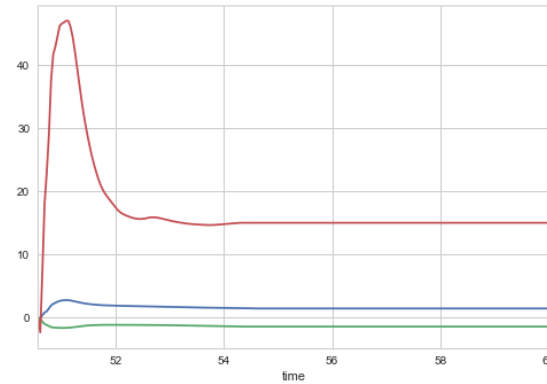


Collision Detection

Example

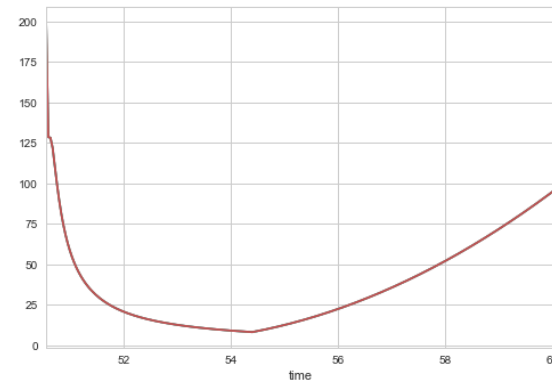
```
In [450]: data_one['vx'].plot() # estimated, blue  
data_one['vy'].plot() # estimated, green  
data_one['vz'].plot() # estimated, red
```

Out[450]: <matplotlib.axes._subplots.AxesSubplot at 0x1b2391b7c50>



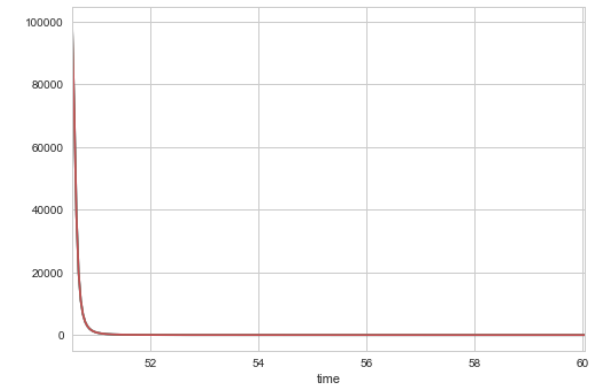
```
In [451]: data_one['sigma_00'].plot() # x, estimated, blue  
data_one['sigma_11'].plot() # y, estimated, green  
data_one['sigma_22'].plot() # z, estimated, red
```

Out[451]: <matplotlib.axes._subplots.AxesSubplot at 0x1b2391a0e10>



```
In [452]: data_one['sigma_33'].plot() # vx, estimated, blue  
data_one['sigma_44'].plot() # vy, estimated, green  
data_one['sigma_55'].plot() # vz, estimated, red
```

Out[452]: <matplotlib.axes._subplots.AxesSubplot at 0x1b2392fb780>



Collision Detection

Example

T=50.56 sec

```
In [506]: mu # mean for Location and velocity
```

```
Out[506]: array([ -4.868,   2.767, -131.242,   0. ,   0. ,   0. ])
```

```
In [508]: sigma # covariance for Location and velocity
```

```
Out[508]: array([[ 1.99601000e+02,  0.00000000e+00,  0.00000000e+00,
                    7.97100000e+00,  0.00000000e+00,  0.00000000e+00],
                  [ 0.00000000e+00,  1.99601000e+02,  0.00000000e+00,
                    0.00000000e+00,  7.97100000e+00,  0.00000000e+00],
                  [ 0.00000000e+00,  0.00000000e+00,  1.99601000e+02,
                    0.00000000e+00,  0.00000000e+00,  7.97100000e+00],
                  [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                    9.98405740e+04,  0.00000000e+00,  0.00000000e+00],
                  [ 7.97100000e+00,  0.00000000e+00,  0.00000000e+00,
                    0.00000000e+00,  7.97100000e+00,  0.00000000e+00],
                  [ 0.00000000e+00,  7.97100000e+00,  0.00000000e+00,
                    0.00000000e+00,  9.98405740e+04,  0.00000000e+00],
                  [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                    0.00000000e+00,  0.00000000e+00,  7.97100000e+00],
                  [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                    0.00000000e+00,  0.00000000e+00,  9.98405740e+04]])
```

```
In [537]: plt.scatter(x,z,alpha=1.0,s=2) # blue, start point
           plt.scatter(x_end,z_end,alpha=1.0,s=2) # green, end point
           plt.scatter(xo,zo,alpha=1.0,s=220) # red, observer
```

```
Out[537]: <matplotlib.collections.PathCollection at 0x1b243a07a20>
```



P=0.000

Collision Detection

Example

T=51.00 sec

```
In [547]: mu # mean for location and velocity
```

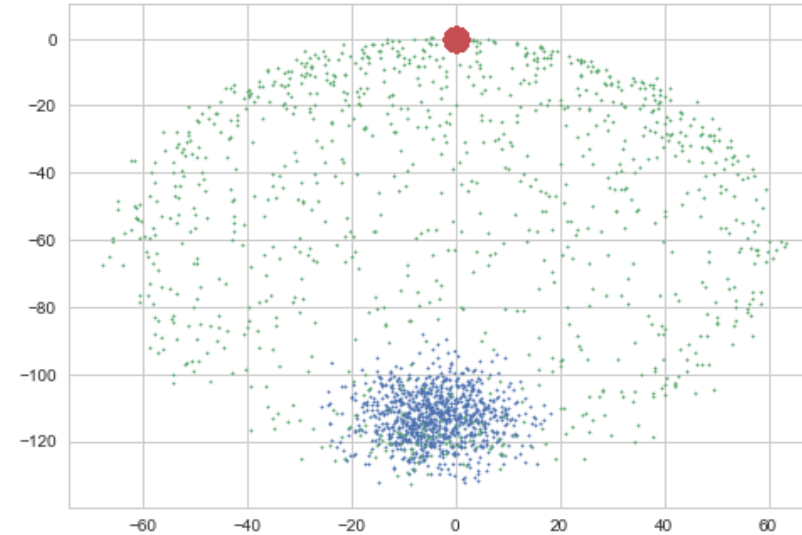
```
Out[547]: array([-3.806,  2.093, -112.909,  2.742, -1.606,  46.543])
```

```
In [549]: sigma # covariance for location and velocity
```

```
Out[549]: array([[ 58.597,  0.   ,  0.   , 190.567,  0.   ,  0.   ],
 [  0.   , 58.597,  0.   ,  0.   , 190.567,  0.   ],
 [  0.   ,  0.   , 58.597,  0.   ,  0.   , 190.567],
 [190.567,  0.   ,  0.   , 866.044,  0.   ,  0.   ],
 [  0.   , 190.567,  0.   ,  0.   , 866.044,  0.   ],
 [  0.   ,  0.   , 190.567,  0.   ,  0.   , 866.044]])
```

```
In [605]: plt.scatter(x,z,alpha=1.0,s=2) # blue, start point
plt.scatter(x_end,z_end,alpha=1.0,s=2) # green, end point
plt.scatter(xo,zo,alpha=1.0,s=220) # red, observer
```

```
Out[605]: <matplotlib.collections.PathCollection at 0x1b24597a780>
```



P=0.000

Collision Detection

Example

T=52.00 sec

```
In [615]: mu # mean for location and velocity
```

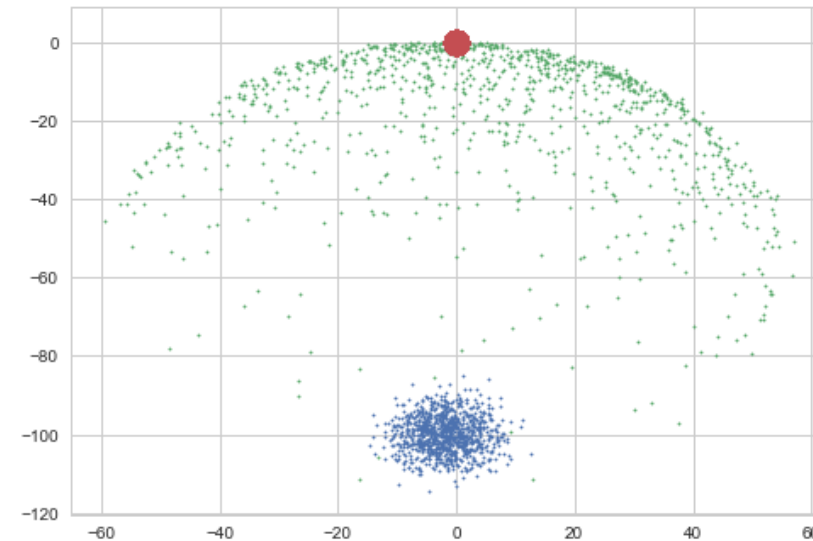
```
Out[615]: array([-2.018,  1.065, -99.876,  1.916, -1.118, 17.396])
```

```
In [617]: sigma # covariance for location and velocity
```

```
Out[617]: array([[ 20.762,  0.    ,  0.    , 21.328,  0.    ,  0.    ],
 [  0.    , 20.762,  0.    ,  0.    , 21.328,  0.    ],
 [  0.    ,  0.    , 20.762,  0.    ,  0.    , 21.328],
 [ 21.328,  0.    ,  0.    , 29.621,  0.    ,  0.    ],
 [  0.    , 21.328,  0.    ,  0.    , 29.621,  0.    ],
 [  0.    ,  0.    , 21.328,  0.    ,  0.    , 29.621]])
```

```
In [639]: plt.scatter(x,z,alpha=1.0,s=2) # blue, start point
plt.scatter(x_end,z_end,alpha=1.0,s=2) # green, end point
plt.scatter(xo,zo,alpha=1.0,s=220) # red, observer
```

```
Out[639]: <matplotlib.collections.PathCollection at 0x1b245cb0eb8>
```



P=0.003

Collision Detection

Example

T=54.00 sec

```
In [661]: mu # mean for location and velocity
```

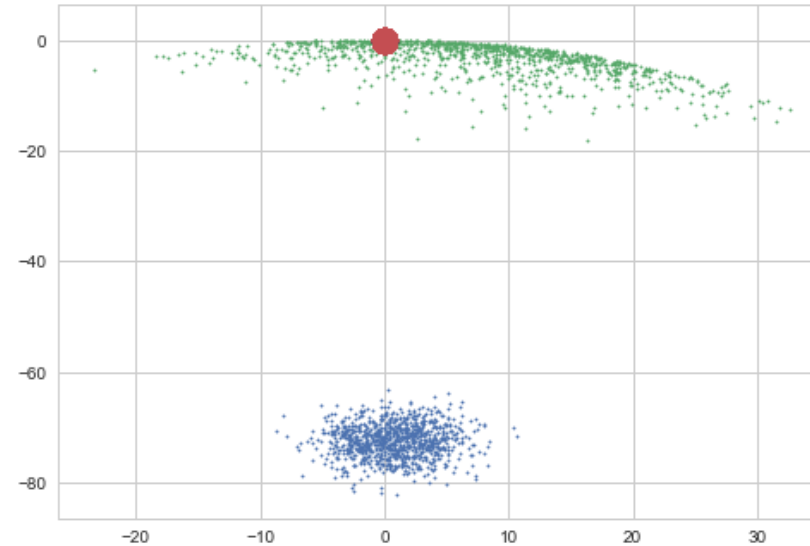
```
Out[661]: array([ 0.83 , -1.716, -72.372,  1.544, -1.332, 14.808])
```

```
In [663]: sigma # covariance for location and velocity
```

```
Out[663]: array([[ 9.038,  0.   ,  0.   ,  3.918,  0.   ,  0.   ],
 [ 0.   ,  9.038,  0.   ,  0.   ,  3.918,  0.   ],
 [ 0.   ,  0.   ,  9.038,  0.   ,  0.   ,  3.918],
 [ 3.918,  0.   ,  0.   ,  2.278,  0.   ,  0.   ],
 [ 0.   ,  3.918,  0.   ,  0.   ,  2.278,  0.   ],
 [ 0.   ,  0.   ,  3.918,  0.   ,  0.   ,  2.278]])
```

```
In [685]: plt.scatter(x,z,alpha=1.0,s=2) # blue, start point
plt.scatter(x_end,z_end,alpha=1.0,s=2) # green, end point
plt.scatter(xo,zo,alpha=1.0,s=220) # red, observer
```

```
Out[685]: <matplotlib.collections.PathCollection at 0x1b2455e9f28>
```



P=0.019

Collision Detection

Example

T=56.00 sec

```
In [695]: mu # mean for location and velocity
```

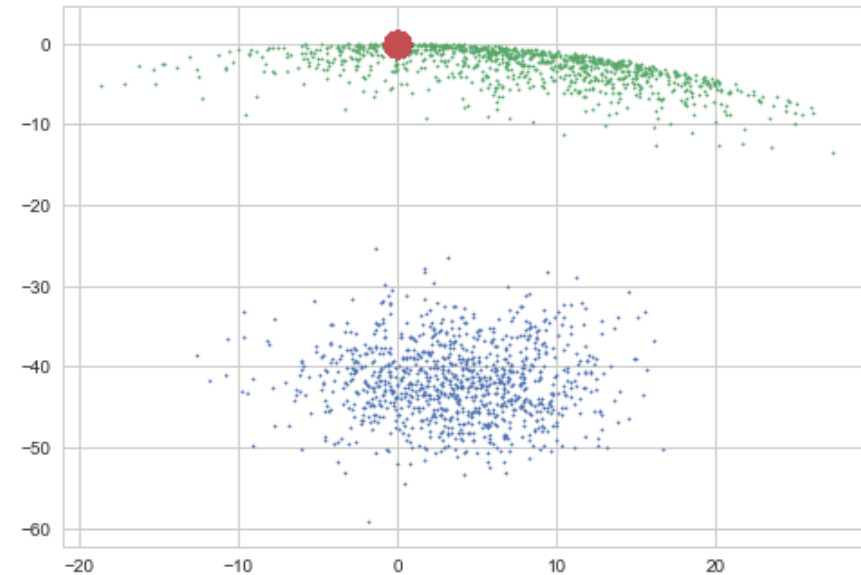
```
Out[695]: array([ 3.616, -4.639, -41.828, 1.473, -1.393, 15.026])
```

```
In [697]: sigma # covariance for location and velocity
```

```
Out[697]: array([[ 22.427,  0.    ,  0.    ,  5.785,  0.    ,  0.    ],
 [  0.    ,  22.427,  0.    ,  0.    ,  5.785,  0.    ],
 [  0.    ,  0.    ,  22.427,  0.    ,  0.    ,  5.785],
 [  5.785,  0.    ,  0.    ,  1.644,  0.    ,  0.    ],
 [  0.    ,  5.785,  0.    ,  0.    ,  1.644,  0.    ],
 [  0.    ,  0.    ,  5.785,  0.    ,  0.    ,  1.644]])
```

```
In [719]: plt.scatter(x,z,alpha=1.0,s=2) # blue, start point
plt.scatter(x_end,z_end,alpha=1.0,s=2) # green, end point
plt.scatter(xo,zo,alpha=1.0,s=220) # red, observer
```

```
Out[719]: <matplotlib.collections.PathCollection at 0x1b245e5a7f0>
```



P=0.027

Collision Detection

Example

T=58.00 sec

```
In [729]: mu # mean for location and velocity
```

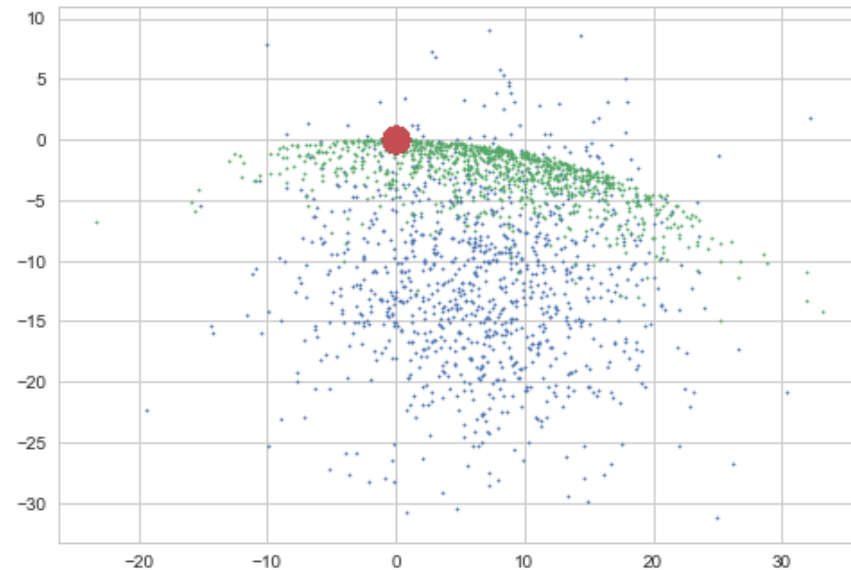
```
Out[729]: array([ 6.562, -7.425, -11.777, 1.473, -1.393, 15.026])
```

```
In [731]: sigma # covariance for location and velocity
```

```
Out[731]: array([[ 52.143,  0.   ,  0.   ,  9.073,  0.   ,  0.   ],
 [  0.   ,  52.143,  0.   ,  0.   ,  9.073,  0.   ],
 [  0.   ,  0.   ,  52.143,  0.   ,  0.   ,  9.073],
 [  9.073,  0.   ,  0.   ,  1.644,  0.   ,  0.   ],
 [  0.   ,  9.073,  0.   ,  0.   ,  1.644,  0.   ],
 [  0.   ,  0.   ,  9.073,  0.   ,  0.   ,  1.644]])
```

```
In [753]: plt.scatter(x,z,alpha=1.0,s=2) # blue, start point
plt.scatter(x_end,z_end,alpha=1.0,s=2) # green, end point
plt.scatter(xo,zo,alpha=1.0,s=220) # red, observer
```

```
Out[753]: <matplotlib.collections.PathCollection at 0x1b24716e4a8>
```



P=0.019

Collision Detection

Open questions:

- How to make uncertainty smaller?
- Collision matrix reachable?

Collision Detection

Distance estimation using object radius:

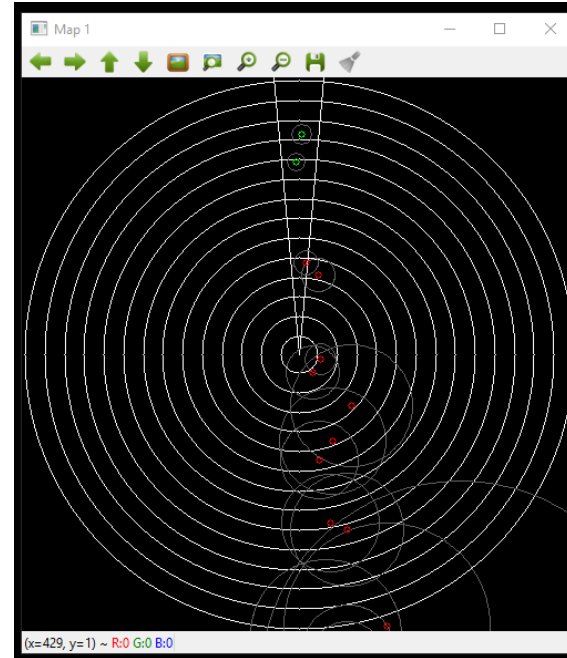
$$t = \frac{d}{\sqrt{x_c^2 + y_c^2 + z_c^2}}$$

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

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

Prediction

Path chosen - a simpler solution which provides a lot more information



Kalman filter update:

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

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

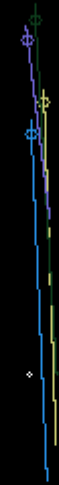
Update is done n times

$$A = \begin{bmatrix} 1 & 0 & 0 & \Delta & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

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

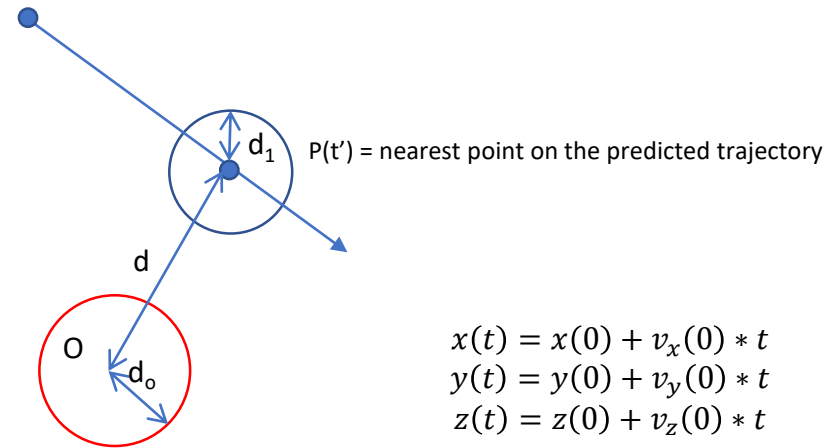
Δ has to be smaller than $1/\text{fps}$. If $\text{fps}=25$, $1/\text{fps} = 0.04$ sec. A car driving 120 km/h will proceed 1.33 meters and a collision with an observer might not be detected well enough. A value of $\Delta = 0.01$ corresponds to the movement of 33 cm for an object moving at 120 km/h. This will generate 100 predictions per second. If we want the prediction horizon to be 10 secs, we have 1000 predictions per object. Predicting for every frame will generate $\text{fps} * 1000 \approx 25\,000$ predictions per second per object. This is too much, so only current prediction is saved.

Prediction



Collision Detection

Collision with the observer



$$\begin{aligned}x(t) &= x(0) + v_x(0) * t \\y(t) &= y(0) + v_y(0) * t \\z(t) &= z(0) + v_z(0) * t\end{aligned}$$

$$d^2(t) = x(t)^2 + y(t)^2 + z(t)^2$$

$$d^2 = (x(0) + v_x(0) * t)^2 + (y(0) + v_y(0) * t)^2 + (z(0) + v_z(0) * t)^2$$

$$\frac{d(d(t)^2)}{dt} = 2 * (x(0) + v_x(0) * t) * v_x(0) + 2 * (y(0) + v_y(0) * t) * v_y(0) + 2 * (z(0) + v_z(0) * t) * v_z(0) = 0$$

$$t' = - \frac{x(0)*v_x(0)+y(0)*v_y(0)+z(0)*v_z(0)}{v_x(0)^2+v_y(0)^2+v_z(0)^2}$$



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).”



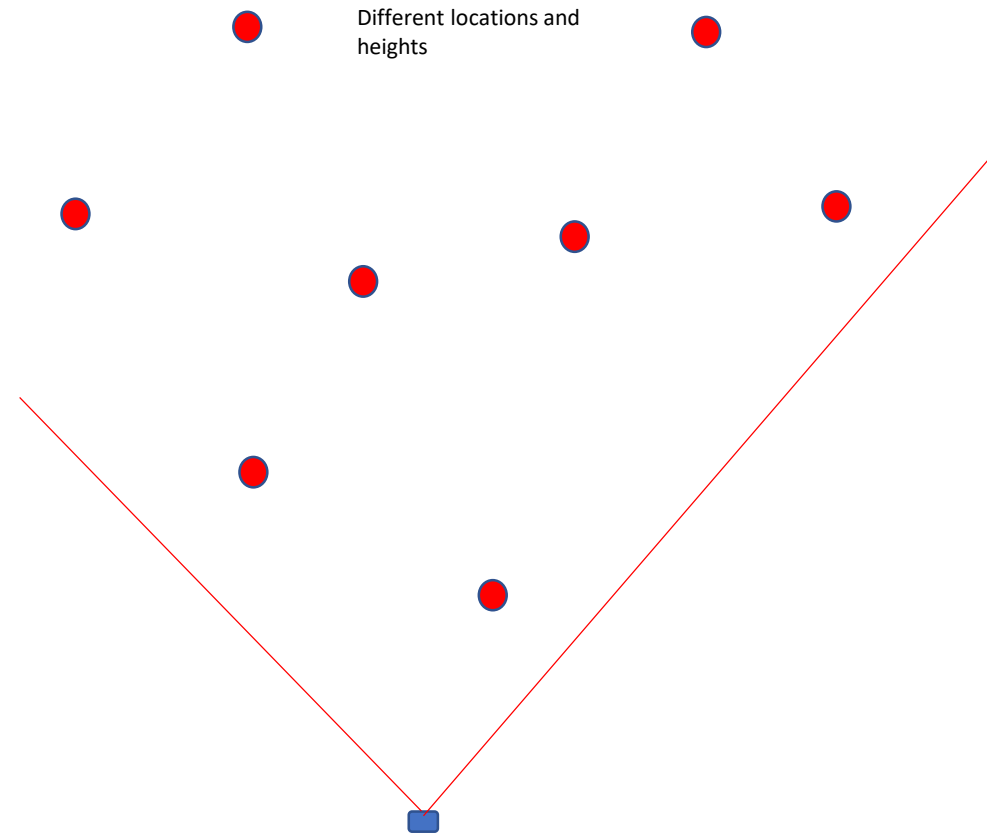
Next Steps

Next steps

- Kalman filter parameter adjustment
- Experiments in wild
- Paper

Experiment In Wild

Experiment 1 in the wild (locations)

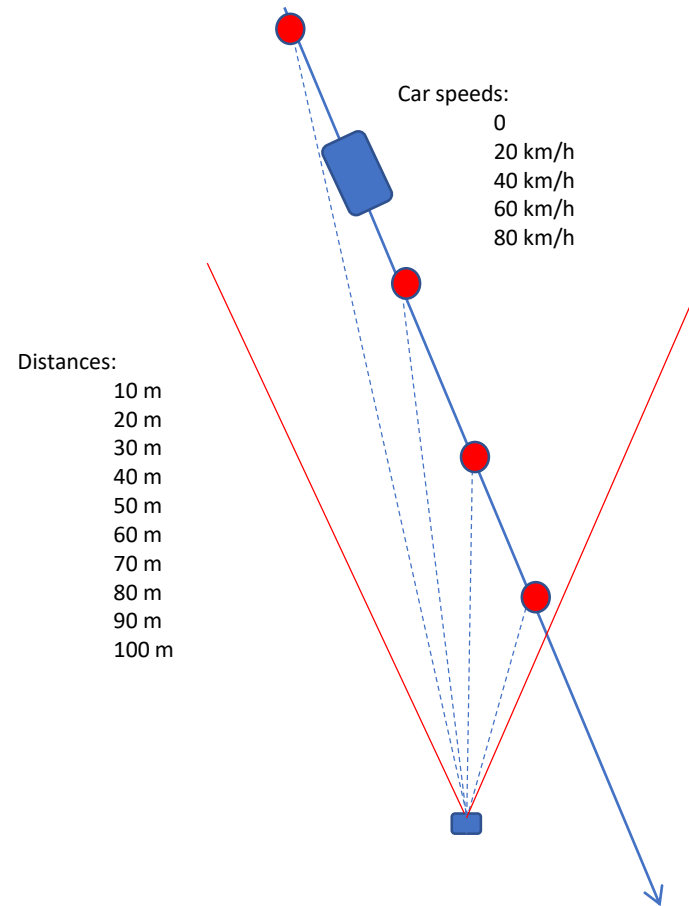


Nikon D800E:

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) = 0.020 m
 $fov = 2 * \arctan(\frac{s_w}{2*f}) = 83.81^\circ$

Experiment In Wild

Experiment 2 in the wild (moving car)



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
 $fov = 2 * \arctan\left(\frac{s_w}{2*f}\right) = 39.49^\circ$

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
-
- ?????
- detection, pattern...
- Hungarian algorithm, distance metrics, movement prediction
- geometry
- Kalman filtering
- Difference equation, uncertainties
- In the wild. See previous slides. **More?**
- Existing datasets? Which?

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.



To Be Discussed

To Be Discussed

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

Thank you!

lampola@student.tut.fi

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