Particle Filter[1]

Lecture 27

See Section 9.3.3 in
Reinhard Klette: Concise Computer Vision
Springer-Verlag, London, 2014

[1] See last slide for copyright information.

Agenda

Particles, Iterative Condensation, and Particle Weights

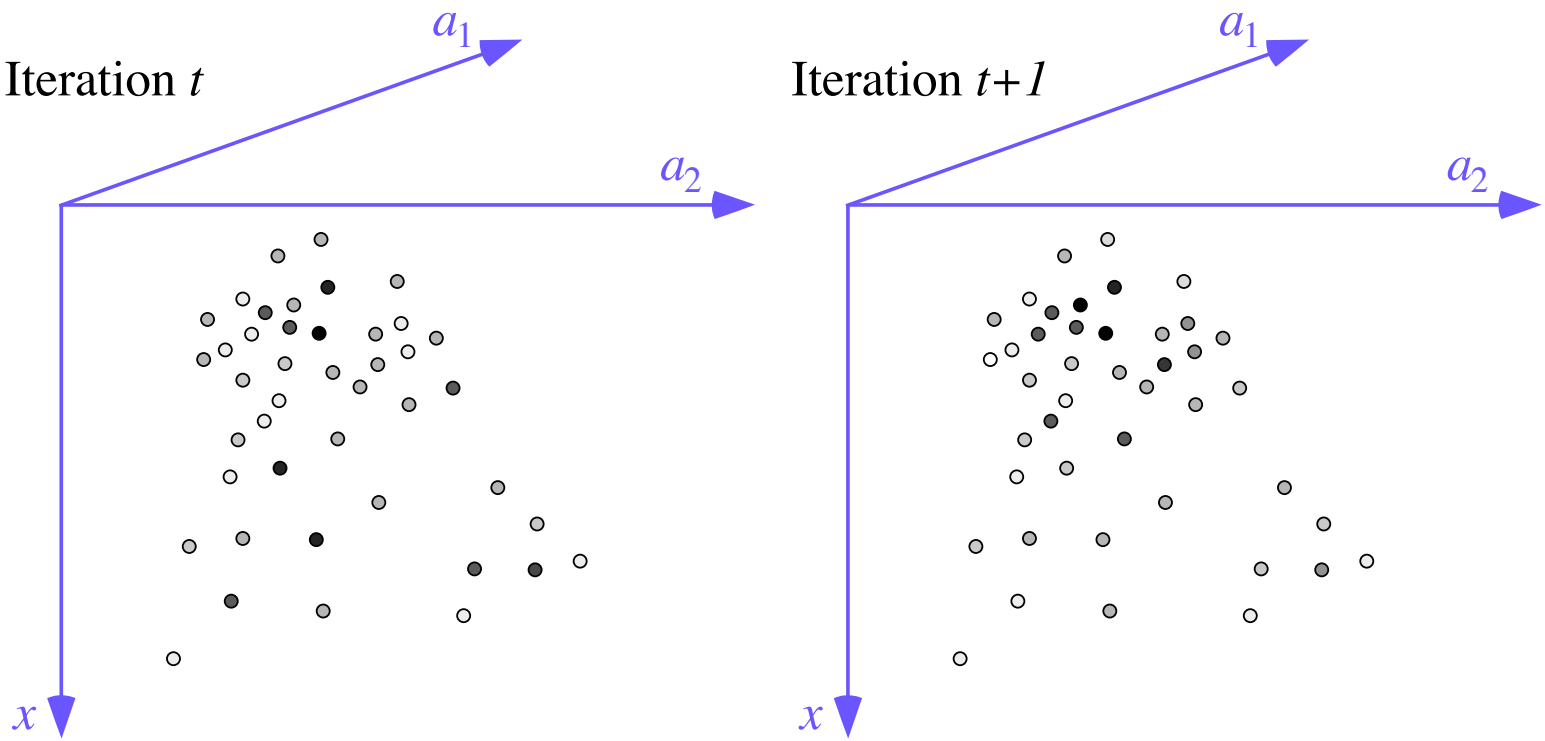
A *particle* is a point in a multi-dimensional space, the *particle space*

A particle may represent a feature (i.e. location and descriptor).

Example: Location such as (x, y) in the image, or just coordinate x on a specified row in the image, with descriptor values in a parameter vector $\mathbf{a} = [a_1, a_2, \ldots, a_m]^\top$ at the location, i.e. $m + 2$ dimensions for axes $x, y, a_1, \ldots, a_m$, i.e. an $(m + 2)$D particle space

A distribution of particles in particle space is transformed by *condensation* into a new distribution taking *weights* of particles into account

Example: 3D Particle Space



One spatial component x and two descriptor components a_1 and a_2

Condensation maps weighted particles from Iteration t to $t+1$

Gray values indicate weights; weights change in iteration steps

Winning Particle

Particle filter may be designed for tracking parameter vectors over time or within a space, based on evaluating consistency with a defined model

By evaluating the consistency of a particle with the model, we assign a weight (a non-negative real) to the particle

Condensation algorithm

Designed for analyzing a cluster of weighted particles for identifying a *winning particle*

Many different strategies: typically, weights are recalculated in iterations for the given cluster of weighted particles

Can also merge particles, change positions, or create new particles

When iteration stops then some kind of mean or local maxima is taken as a winning particle

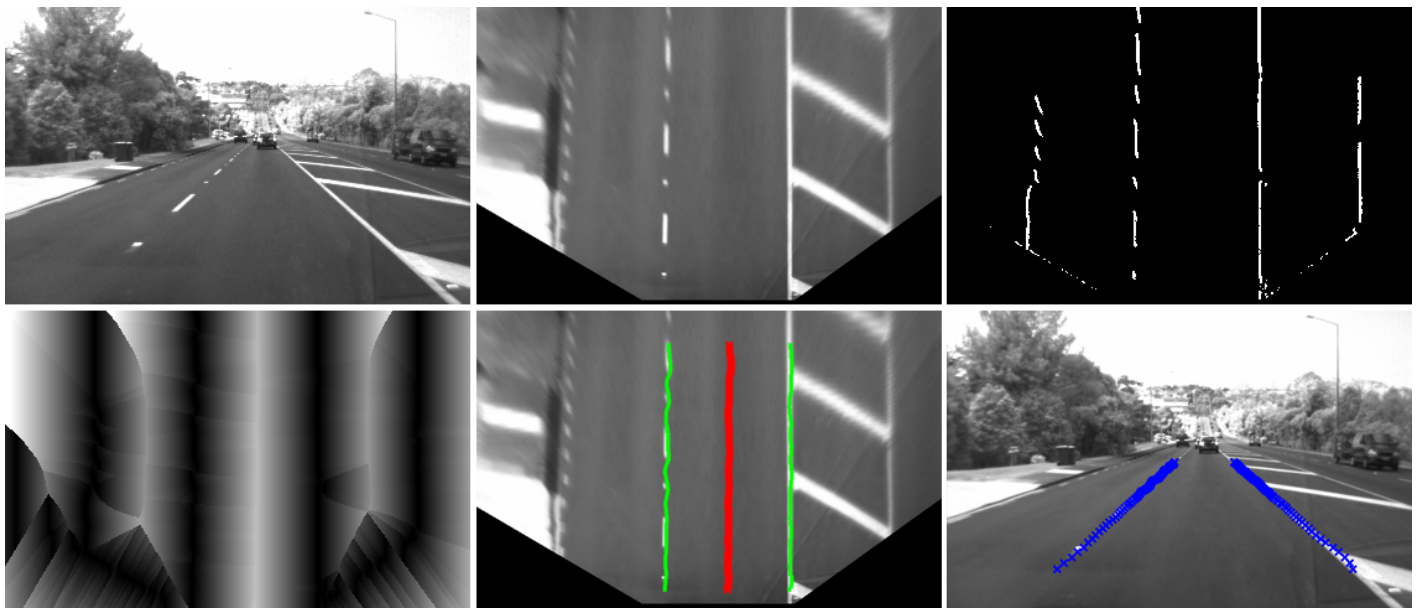
Agenda

Example: Lane Border Detection

A feature combines a left and a right point in the same image row (e.g. left and right lane border on the road)

Such a feature needs to be tracked in the the same image, say from one image row to the next in bottom-up direction

Translational movement of both contributing points to one feature can (slightly) differ by changes in lane width and position



Caption

Figure on slide before illustrates a possible chain of subprocesses for lane border detection

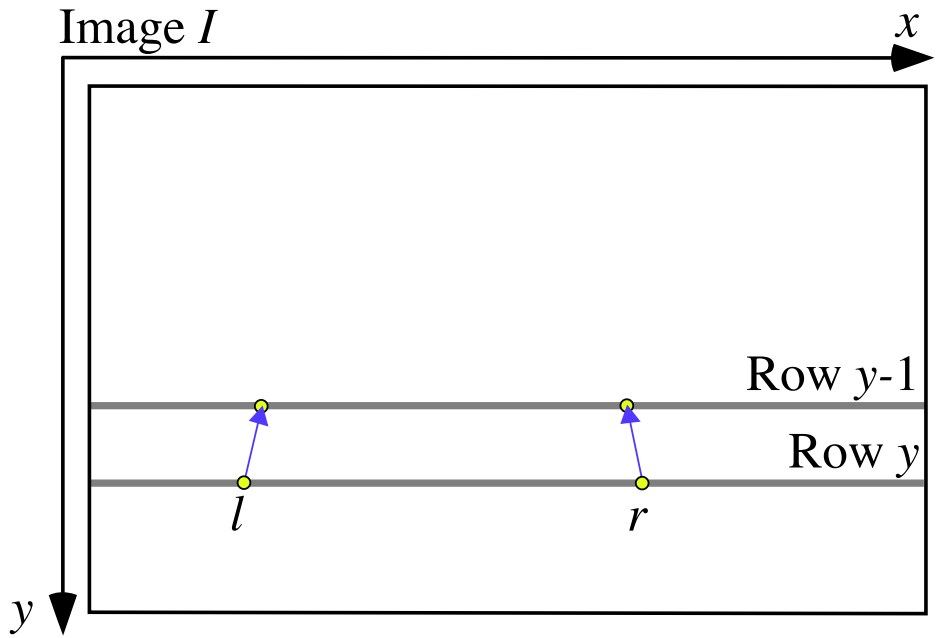
Top, left to right:
Input frame
Bird's-eye view
Detected vertical edges

Bottom, left to right:
Row components of Euclidean distance transform (EDT), shown as absolute values
Detected center of lane and lane borders
Lane borders projected back into the perspective view of the recorded frames

Model About Particle Movement

Features (each defined by keypoint and descriptor) are to be tracked. We have a general model about expected movement of those features

Example: Tracking of features row by row, bottom-up in one image



Expected movement might be: no motion, features expected to remain at same positions in the next row

Dynamic Model and Descriptors

From Row y to Row $y-1$: a feature, defined by coordinates l and r and further parameters combined into descriptors $\mathbf{a}_l$ and $\mathbf{a}_r$, moves into a new location, defined by updated values of l and r

The expected move (the *dynamic model*) can be characterised by a matrix. If no particular move is expected, this can simply be the identity matrix.

Descriptors of features are defined by a specific, application-dependent model, not by a generic model of having (e.g.) just a disk of influence

The model allows some *weighting* due to consistencies between descriptor values and model assumptions

A particle filter is appropriate for such a feature tracking scenario

Agenda

Lane Borders

Task: Detection of lanes of a road in video data recorded in a driving ego-vehicle; brief outline of (a possible) general workflow:

1. Map recorded video frames into a bird's-eye view
2. Detect vertical edges in bird's-eye view, remove edge-artifacts
3. Perform Euclidean distance transform (EDT) for calculating minimum distances between pixel locations $p = (x, y)$ to those edge pixels

   [Use *signed row components* $x - x_{edge}$ of calculated distances $\sqrt{(x - x_{edge})^2 + (y - y_{edge})^2}$ for identifying centers of lanes at places where signs are changing, and distance values about half of the expected lane width]

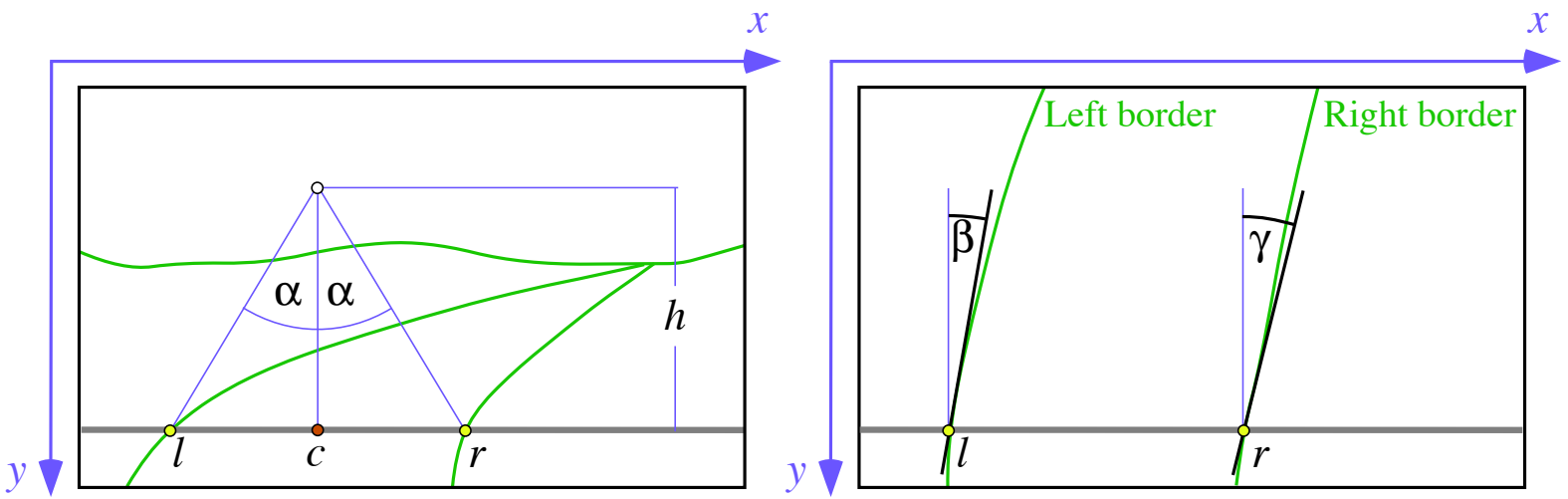
4. Apply a particle filter for propagating detected lane border pixels bottom-up, row by row, such that we have the most likely pixels as lane border pixels again in the next row

Generating a Bird's-Eye View

Step 1 on Slide 12 can be done by

1. *inverse perspective mapping* using the calibration data for the used camera (comment: more work-intensive; requires calibration) or
2. simply by marking four pixels in the image, supposed to be corners of a rectangle in the plane of the road (thus appearing as a trapezoid in the perspective image), and by applying a *homography* which maps those marked four points into a rectangle, and at the same time the perspective view into a bird's-eye view

Used Lane-Border Model



Left: Model visualization for the perspective image: Center point c defines at fixed height h angle α which identifies left and right lane border points at l and r

Right: Bird's-eye view: Model parameters defined for this view; at detected points l and r, we have tangent angles β and γ to the left and right lane border, detected as dominantly-vertical edges

4D Particle Space

The lane-border model of Slide 14 defines a 4D particle space

For a detected center of a lane, where positive and negative row components of the EDT meet (with absolute values which differ by 1 at most) in the bird's-eye view, we apply a fixed height h, and have (applying the positive and negative row components) angle α which identifies x-coordinates l and r

Local approximations of tangents at l and r to detected lane borders define angles β and γ

Height h and angle α is shown in the perspective view; coordinates l and r are actually defined in the bird's-eye view

Feature: combines two points (l, y) and (r, y) in row y on the left and right lane border [or one *keypoint* (c, y)] together with angles β and γ, defined by one vector $\mathbf{a} = [c, \alpha, \beta, \gamma]^\top$ (i.e. a particle in 4D space)

Agenda

Initialization of the Tracking Process

Having a sequence of frames, results from previous frame can be used for initializing a feature in a row close to the bottom of the current frame

Now: at the very first frame of the sequence, or at a frame after lane borders had been lost in the previous frame

Start row y_0 is close to the bottom of the first frame

- Option (1): Search for pixel (c, y) with a positive row-distance value, having an adjacent pixel in the same row with a negative row-distance value; possibly we need to move up to the next row until we have a proper initial value c
- Option (2): Run special detectors for points (l, y) and (r, y)

For the start row, initial values define the first feature vector (particle)

$$\mathbf{a}_0 = [c_0, \alpha_0, \beta_0, \gamma_0]^\top$$

Updating the Feature Vector of the Particle Filter

Track feature vector $\mathbf{a} = [c, \alpha, \beta, \gamma]^\top$ (a particle) from start row upward, up to a row which defines the upper limit for expected lane borders (there are no lanes in the sky)

Row parameter y is calculated incrementally by applying a fixed increment Δ, starting at y_0 in the birds-eye image

Row at Step n is identified by $y_n = y_0 + n \cdot \Delta$

In the update process, a particle filter applies (in general) two *update models*

Two Update Models

The Dynamic Model

A dynamic model matrix $\mathbf{A}$ defines default motion of particles (here: in the image)

Let p_n be the keypoint in Step n, expressed as a vector

Prediction value $\hat{p}_n$ generated from p_{n-1} by $\hat{p}_n = \mathbf{A} \cdot p_{n-1}$

A general and simple choice: $\mathbf{A} = \mathbf{I}$ (also used for our example)

The Observation Model

Determines a particle's weight during resampling

Observation Model for Given Example

Points (c_n, y_n) are assumed to have large absolute row-distance values

Let L_n and R_n be short digital line segments, centered at (l_n, y_n) and (r_n, y_n), representing the tangential lines at those two points in the bird's-eye view

It is assumed that these two line segments are formed by pixels which have absolute row-distance values (in EDT) close to 0

We assume that L_n and R_n are formed by an 8-path of length $2k + 1$, with points (l_n, y_n) or (r_n, y_n) at the middle position

Agenda

Generation of Random Particles

In each step when going forward to the next row, we generate $N_{part} > 0$ particles randomly around the predicted parameter vector (following the dynamic model) in mD particle space

For better results use a larger number of generated particles (e.g. $N_{part} = 500$)

The figure on Slide 4 illustrates 43 particles in a 3D particle space

Let

$$\hat{\mathbf{a}}_n^i = [\hat{c}_n^i, \hat{\alpha}_n^i, \hat{\beta}_n^i, \hat{\gamma}_n^i]^\top$$

be the i^{th} particle generated in Step n, for $1 \leq i \leq N_{part}$

Particle Generation Using a Uniform Distribution

Apply a uniform distribution for generated particles in mD particle space

For the discussed example:

For c we assume an interval $[c - 10, c + 10]$, and we select uniformly random values for the c-component in this interval

This process is independent from the other components of a vector $\hat{\mathbf{a}}_n^i$

For the second component, we assume an interval (say) of $[\alpha - 0.1, \alpha + 0.1]$, for the third $[\beta - 0.5, \beta + 0.5]$, and similar for γ

Particle Generation Using a Gauss Distribution

We use a Gauss (or normal) distribution for the generated particles in mD particle space

For the discussed example:

A zero-mean distribution produces values around the predicted value

For the individual components we assume a standard deviation $\sigma > 0$ such that we generate values about in the same intervals as specified for the uniform distribution on the slide before

For example, we use $\sigma = 10$ for the c-component

Agenda

Particle Weights

We define a weight for the ith particle $\hat{\mathbf{a}}_n^i$

In the discussed example:

Left position equals

$$l_n^i = \hat{c}_n^i - h \cdot \tan \hat{\alpha}_n^i$$

Sum of absolute row-distance values $d_r(x, y) = |x - x_{edge}|$, as provided by the EDT in the bird's-eye view, along line segment L (assumed to be an 8-path of length $2k + 1$) equals

$$S_L^i = \sum_{j=-k}^{k} \left| d_r \left(l_n^i + j \cdot \sin \hat{\beta}_n^i,\ y_n + j \cdot \cos \hat{\beta}_n^i \right) \right|$$

Calculate S_R^i for the second line segment in an analogous way

Two Individual Weights

Obtain the weight

$$\omega_{dist}^{i} = \frac{1}{2\sigma_l \sigma_r \pi} \exp\left(-\frac{(S_L^i - \mu_l)^2}{2\sigma_l} - \frac{(S_R^i - \mu_r)^2}{2\sigma_r} \right)$$

with respect to distance values on L and R, where μ_l, μ_r, σ_l, and σ_r are estimated constants (say, zero-mean $\mu_l = \mu_r = 0$ for the ideal case) based on experiments for the given application

For the generated center point (c_n^i, y_n), the weight equals

$$\omega_{center}^{i} = \frac{1}{\sigma_c \sqrt{2\pi}} \exp\left(-\frac{\left(\left| \frac{1}{d_r(c_n^i, y_n)} \right| - \mu_c \right)^2}{2\sigma_c} \right)$$

where μ_c and σ_c are again estimated constants

Total Weight

Total weight for the ith particle $\hat{\mathbf{a}}_n^i$, at the beginning of the iterative condensation process, is given by

$$\omega_i = \omega_{dist}^i \cdot \omega_{center}^i$$

These weights decide about the "influence" of the particles during the condensation process

A normalization of all the N_{part} weights is normally required before applying one of the common condensation programs

There is a condensation procedure available in `OpenCV`: run for the selected numbers of iterations and use the specified winning particle; this defines here the lane borders in the next image row; see `CvConDensation`

Condensation

Iterative condensation decides which of the randomly generated particles is taken as result for the next image row

One iteration of condensation, also called *resampling*, may merge particles, delete particles, or create new particles for improving "quality" of particles

A particle with a high weight is very likely to "survive" the resampling process; resampling takes all the current weighted particles as input and outputs a set of weighted particles

Often, particles "shift" towards such particles which had a higher weight in the input data; a small number of iterations or resamplings is often appropriate (e.g. 2 to 5)

At the end of the iterations, the particle with the highest weight is taken as the result, or the weighted mean of all particles at the end of the resampling process

Agenda

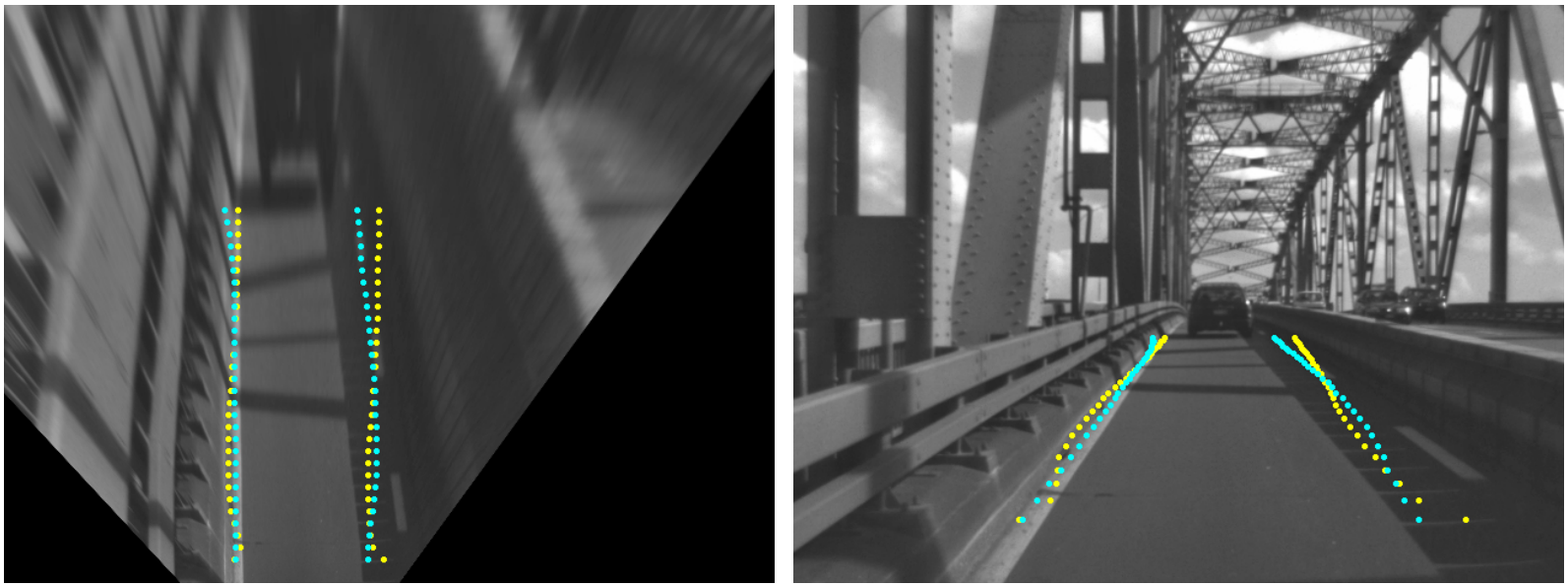
More Than One Row at a Time

By using more than just one row for defining a particle, or by using separate particle filters (i.e. α_1 and α_2) for the left or right lane border, results can be improved in general

Example: Using just 3 rows forward appears in general to be more reasonable than 8 rows; number of used rows should be changed dynamically according to state of lane borders

Cyan points are here for 2 rows backward and only 3 rows forward

Illustration of Results



Left: Generated points (l_n, y_n) and (r_n, y_n) using the described method (in yellow) and an extended method where each particle is defined by 2 rows backward and 8 rows forward (in cyan), defining an $11 \times 4 = 44$-dimensional particle space

Right: Generated points backprojected into the corresponding source frame

Copyright Information

This slide show was prepared by Reinhard Klette
with kind permission from Springer Science+Business Media B.V.

The slide show can be used freely for presentations.
However, *all the material* is copyrighted.

R. Klette. Concise Computer Vision.

In case of citation: just cite the book, that's fine.