LECTURE MACHINE VISION 2019/20

Institut für Mess- und Regelungstechnik, Karlsruher Institut für Technologie
Dr. Martin Lauer, Christian Kinzig, M.Sc.

Example Solutions for Practical Exercises: Line Estimation

- **Least-sum-of-squares Estimator**
  The basic idea is explained on slide 04/32, the recipe can be found on slide 04/30:
  1. calculate $\sum_i x_{i,1}, \sum_i x_{i,2}, \sum_i x_{i,1}^2, \sum_i x_{i,2}^2, \sum_i x_{i,1}x_{i,2}$
  2. calculate $\alpha = \sum_i x_{i,1}^2 - \frac{1}{N}(\sum_i x_{i,1})^2$,
     $\beta = \sum_i x_{i,1}x_{i,2} - \frac{1}{N}\sum_i x_{i,1} \cdot \sum_i x_{i,2}$,
     $\gamma = \sum_i x_{i,2}^2 - \frac{1}{N}(\sum_i x_{i,2})^2$ (slide 04/35)
  3. calculate Eigenvalues and Eigenvectors of matrix $\begin{pmatrix}\alpha & \beta \\ \beta & \gamma\end{pmatrix}$
  4. choose smaller Eigenvalue and related Eigenvectors as λ and $\vec{n}$
  5. calculate $c = -\frac{1}{N}\sum_i \langle \vec{n}, \vec{x}_i \rangle$ (slide 04/29)

- **Start- and Endpoint**
  The basic idea is to project the edge points onto the line which was determined before. Using the algorithm from slide 04/4 we get a value τ_i for each point $\vec{x}_i$. The extremal points can be determined searching for the smallest and largest τ_i. To be able to apply this approach we need to determine two points on the line before. As derived on slide 04/6 one point $\vec{p}$ is given by $\vec{p} = -c\vec{n}$. A second point can be determined starting from $\vec{p}$ and moving into the direction of the line. The direction of the line can be calculated turning $\vec{n}$ by 90°.

  The complete algorithm combining the least-sum-of-squares estimator with the endpoint calculation is given on the next page (function lslinefit).

  Applying the LS-fit and plotting the line segments in the image can be done as follows:

```
imshow (postit2g); hold on;
for i=1:length(pixellist)
    S = lslinefit (pixellist(i).list);
    plot ([S.point1(1) S.point2(1)],[S.point1(2) S.point2(2)],'r-o');
end
```

- **Robust Fitting**
  The subsequent pages show implementations of the weighted least squares approach, the M-estimators, LTS, and RANSAC as described on the lecture slides of chapter 4.

```
function [ S ] = lslinefit( P )
% LSLINEFIT calculate least-squares line estimation from a
% set of 2D-positions
% [ S ] = lslinefit( P )
%   P: list of positions (one point per line)
%   S: struct, that contains a description of the line segment
%   S.norm: the normal vector
%   S.theta: direction of the normal vector (angle in rad)
%   S.offset: offset of the line, i.e.
%             x*cos(theta)+y*sin(theta)+offset=0
%   S.point1, S.point2: start- and endpoint
N = length(P);
% 1st step (see description):
sx = sum(P(:,1));
sy = sum(P(:,2));
sxx = sum(P(:,1).^2);
syy = sum(P(:,2).^2);
sxy = sum(P(:,1).*P(:,2));
% 2nd step:
M=[sxx-sx*sx/N sxy-sx*sy/N; sxy-sx*sy/N syy-sy*sy/N];
% 3rd step:
[V D]=eig(M);
% 4th step:
if ( D(1,1)<D(2,2) )
    S.norm = V(:,1);
else
    S.norm = V(:,2);
end
S.theta=atan2(S.norm(2),S.norm(1));
S.offset=-sum(P*S.norm)/N;
% end of least-sum-of-squares estimator
% from here on: calculate end points
% determine a point p on the line:
p=-S.offset*S.norm;
% determine a vector orthogonal to normal vector
r=[ S.norm(2); -S.norm(1) ];
% determine tau-values for all points:
tau=P*r;
% determine minimal and maximal tau:
tmn=min(tau);
tmx=max(tau);
S.point1=p+tmn*r;
S.point2=p+tmx*r;
```

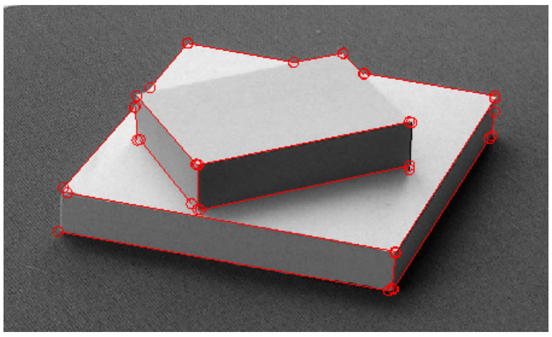
```
function [ S ] = wlslinefit( P, w )
% WLSLINEFIT calculate weighted least-squares line estimation from a
% set of 2D-positions
% [ S ] = lslinefit( P, w )
%   P: list of positions (one point per line)
%   w: list of weights (one per point)
%   S: struct, that contains a description of the line segment
%   S.norm: the normal vector
%   S.theta: direction of the normal vector (angle in rad)
%   S.offset: offset of the line, i.e.
%             x*cos(theta)+y*sin(theta)+offset=0
%   S.point1, S.point2: start- and endpoint
% 1st step (see description):
W = sum(w);
sx = sum(w'.*P(:,1));
sy = sum(w'.*P(:,2));
sxx = sum(w'.*(P(:,1).^2));
syy = sum(w'.*(P(:,2).^2));
sxy = sum(w'.*(P(:,1).*P(:,2)));
% 2nd step:
M=[sxx-sx*sx/W sxy-sx*sy/W; sxy-sx*sy/W syy-sy*sy/W];
% 3rd step:
[V D]=eig(M);
% 4th step:
if ( D(1,1)<D(2,2) )
  S.norm = V(:,1);
else
  S.norm = V(:,2);
end
S.theta=atan2(S.norm(2),S.norm(1));
S.offset=-sum(w'.*(P*S.norm))/W;
% end of least-sum-of-squares estimator
% from here on: calculate end points
% determine a point p on the line:
p=-S.offset*S.norm;
% determine a vector orthogonal to normal vector
r=[ S.norm(2); -S.norm(1) ];
% determine tau-values for all points:
tau=P*r;
% determine minimal and maximal tau:
tmn=min(tau);
tmx=max(tau);
S.point1=p+tmn*r;
S.point2=p+tmx*r;
```

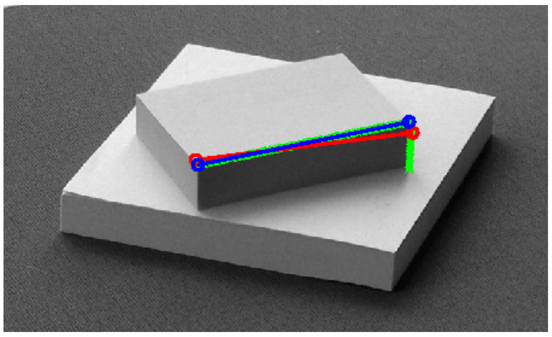
```
function [ S ] = mlinefit( P, type, parameter )
% MLINEFIT calculate M-estimator line estimation from a set of 2D-positions
% [ S ] = mlinefit( P, type, parameter )
%   P: list of positions (one point per line)
%   type: either 'Huber' or 'Cauchy' to specify the error term
%   parameter: parameter of the error term (k or c)
%   S: struct, that contains a description of the line segment
%   S.norm: the normal vector
%   S.theta: direction of the normal vector (angle in rad)
%   S.offset: offset of the line, i.e.
%             x*cos(theta)+y*sin(theta)+offset=0
%   S.point1, S.point2: start- and endpoint
[N d]=size(P);
weights = (1/N)*ones (1, N); % initialize weights equally
for i=1:5 % repeat 5 times
  S = wlslinefit (P, weights); % calculate weighted least squares fit
  d = (P*S.norm+S.offset)'; % caluclate distances
  % recalculate weights:
  if (strcmp(type,'Huber'))
    absd = abs(d);
    weights = [absd<=parameter]+[absd>parameter]*parameter./absd;
  elseif (strcmp(type,'Cauchy'))
    weights = (d.^2/(parameter*parameter)+1).^-1;
  else
    warning (['unknown error type ' type])
  end
end
```

```
function [ S ] = ltslinefit( P, q )
% LTSLINEFIT calculate LTS line estimation from a set of 2D-positions
% [ S ] = lslinefit( P, q )
%   P: list of positions (one point per line)
%   q: acceptance ratio (between 0 and 1)
%   S: struct, that contains a description of the line segment
%   S.norm: the normal vector
%   S.theta: direction of the normal vector (angle in rad)
%   S.offset: offset of the line, i.e.
%             x*cos(theta)+y*sin(theta)+offset=0
%   S.point1, S.point2: start- and endpoint
[N d]=size(P);
qn = floor(N*q);  % the number of points considered
S.theta=0; % an arbitrary line as baseline
S.offset=0;
S.norm=[1 0]';
error = sum(abs(P*S.norm));
for i=1:10 % repeat 10 trials
  perm = randperm (N);
  subset_index = perm(1:2); % randomly select two points
  perm_old = zeros (N,1);

  while (true) % while not this trial converged...
    S1 = lslinefit (P(subset_index,:)); % fit line to selected points
    d = abs(P*S1.norm+S1.offset); % error for all points
    [ds, perm] = sort (d); % sort in order of ascending error
    if (perm(1:qn)==perm_old(1:qn)) % stop, if subset does not change
      break;
    end
    subset_index = perm (1:qn); % qn best points for next iteration
    perm_old = perm;
  end
  error1 = sum(d(subset_index));
  if (error1 < error) % check if present trial is best
    error = error1;
    S = S1;
  end
end
```

```
function [ S ] = ransaclinefit( P, q )
% RANSACLINEFIT calculate RANSAC line estimation from a
% set of 2D-positions
% [ S ] = ransaclinefit( P, q )
%   P: list of positions (one point per line)
%   q: threshold (RANSAC sensitivity threshold)
%   S: struct, that contains a description of the line segment
%   S.norm: the normal vector
%   S.theta: direction of the normal vector (angle in rad)
%   S.offset: offset of the line, i.e.
%             x*cos(theta)+y*sin(theta)+offset=0
%   S.point1, S.point2: start- and endpoint
[N d]=size(P);
S.theta=0; % an arbitrary line as baseline
S.offset=0;
S.norm=[1 0]';
inlier_index = (abs(P*S.norm)<q); % those points are inliers
error = N-sum(inlier_index);
for i=1:200 % repeat 200 trials
  perm = randperm (N);
  subset = P(perm(1:2),:); % randomly select two points
  S1 = lslinefit (subset); % fit a line to the selected two points
  inlier_index1 = (abs(P*S1.norm+S1.offset)<q);
  error1 = N-sum(inlier_index1); % error for all points
  if (error1 < error)
    error = error1;
    S = S1;
    inlier_index = inlier_index1;
  end
end
% end of least-sum-of-squares estimator
% from here on: calculate end points
% determine a point p on the line:
p=-S.offset*S.norm;
% determine a vector orthogonal to normal vector
r=[ S.norm(2); -S.norm(1) ];
% determine tau-values for all points:
tau=P(inlier_index,:)*r;
% determine minimal and maximal tau:
tmn=min(tau);
tmx=max(tau);
S.point1=p+tmn*r;
S.point2=p+tmx*r;
```


Lines fitted to the edges using the least-sum-of-squares approach


Results of the LTS approach. Green: points used. Red: least-sum-of-squares approach with all points. Blue: LTS-fit with 80% acceptance rate.