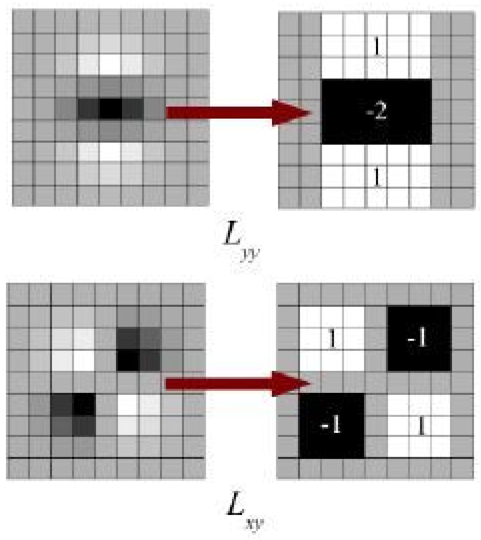
MOBILE ROBOTICS
Two View Sparse Reconstruction

Unni Krishnan R Nair
20171405

Detect and match SURF points(Speeded Up Robust Features)

In SIFT, Lowe approximated Laplacian of Gaussian with Difference of Gaussian for finding scale-space. SURF goes a little further and approximates LoG with Box Filter. Below image shows a demonstration of such an approximation. One big advantage of this approximation is that, convolution with box filter can be easily calculated with the help of integral images. And it can be done in parallel for different scales. Also the SURF rely on determinant of Hessian matrix for both scale and location.



For orientation assignment, SURF uses wavelet responses in horizontal and vertical direction for a neighbourhood of size 6s. Adequate guassian weights are also applied to it.

Then they are plotted in a space as given in below image. The dominant orientation is estimated by calculating the sum of all responses within a sliding orientation window of angle 60 degrees. Interesting thing is that, wavelet response can be found out using integral images very easily at any scale.

indexPairs = matchFeatures(features1,features2) returns indices of the matching features in the two input feature sets. The input feature must be either binary Features objects or matrices.

RANSAC to find good points

```
Given:
    data – a set of observations
    model – a model to explain observed data points
    n – minimum number of data points required to estimate model parameters
    k – maximum number of iterations allowed in the algorithm
    t – threshold value to determine data points that are fit well by model
    d – number of close data points required to assert that a model fits well to data

Return:
    bestfit – model parameters which best fit the data (or nul if no good model is found)

iterations = 0
bestfit = nul
besterr = something really large
while iterations < k {
    maybeinliers = n randomly selected values from data
    maybemodel = model parameters fitted to maybeinliers
    alsoinliers = empty set
    for every point in data not in maybeinliers {
        if point fits maybemodel with an error smaller than t
             add point to alsoinliers
    }
    if the number of elements in alsoinliers is > d {
        % this implies that we may have found a good model
        % now test how good it is
        bettermodel = model parameters fitted to all points in maybeinliers and alsoinliers
        thiserr = a measure of how well bettermodel fits these points
        if thiserr < besterr {
            bestfit = bettermodel
            besterr = thiserr
        }
    }
    increment iterations
}
return bestfit
```

Find Fundamental Matrix using 8 pt algorithm

$$\begin{bmatrix} x'_i & y'_i & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0$$

$$x_i x'_i f_{11} + x_i y'_i f_{21} + x_i f_{31} +$$
$$y_i x'_i f_{12} + y_i y'_i f_{22} + y_i f_{32} +$$
$$x'_i f_{13} + y'_i f_{23} + f_{33} = 0$$

$$\begin{bmatrix} x'_i & y'_i & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0$$

Given m point correspondences…

$$\begin{bmatrix} x_1 x'_1 & x_1 y'_1 & x_1 & y_1 x'_1 & y_1 y'_1 & y_1 & x'_1 & y'_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_m x'_m & x_m y'_m & x_m & y_m x'_m & y_m y'_m & y_m & x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

Think: how many points do we need?

Solve using SVD

Then reduce dimensionality

```
[u d v] = svd(F);

new_d = diag([d(1,1) d(2,2) , 0]);

F = u * new_d * v' ;
```

Find Essential Matrix

E = K'*F*K

Reduce the dimensionality

[u d v] = svd(E);

new_d = diag([(d(1,1)+d(2,2))/2, (d(1,1)+d(2,2))/2 , 0]);

E = u * new_d * v' ;

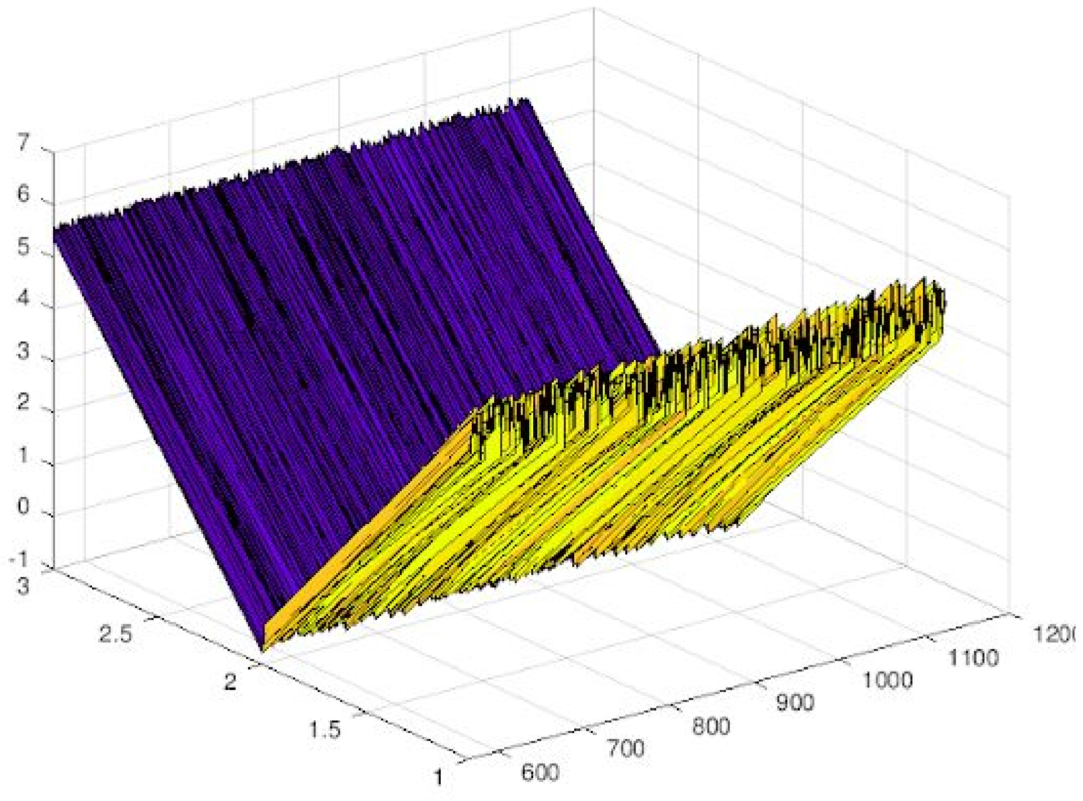
Decompose Essential Matrix into R and T of the camera

Get Projection matrix of the respective cameras and use it to project 2d to 3d

Of the 4 possible R and Ts select the best fitting R and T and use it to project all points

```
function [pts3D] = algebraicTriangulation(pts2D_1, pts2D_2, ProjMat_1,
ProjMat_2)

ProjMat_1=[ProjMat_1(1,1), ProjMat_1(1,2), ProjMat_1(1,4); ProjMat_1(2,1),
ProjMat_1(2,2), ProjMat_1(2,4); ProjMat_1(3,1), ProjMat_1(3,2),
ProjMat_1(3,4)];

ProjMat_2=[ProjMat_2(1,1), ProjMat_2(1,2), ProjMat_2(1,4); ProjMat_2(2,1),
ProjMat_2(2,2), ProjMat_2(2,4); ProjMat_2(3,1), ProjMat_2(3,2),
ProjMat_2(3,4)];

pts3D=inv(ProjMat_1)*pts2D_1;

temp=inv(ProjMat_2)*pts2D_2;

pts3D=[pts3D temp];

%pts3D=transpose(pts3D);

end
```

FINAL PLOTTED 3D surface



7
6
5
4
3
2
1
0
-1
3
2.5
2
1.5
1
600
700
800
900
1000
1100