Image Processing and Computer Vision (IPCV)

Prof. Dr. J. Weickert — Summer term 2019
Mathematical Image Analysis Group — Saarland University

Example Solutions for Homework Assignment 13 (H13)

Problem 1 (Homogeneous Coordinates)

We first want to remark that all projective coordinates that are a multiple of each other, i.e. where $\tilde{\boldsymbol{a}} = \lambda \tilde{\boldsymbol{b}}$ for $\lambda \neq 0$, denote the same point. This reflects the depth ambiguity.

(a) If $\boldsymbol{m}_1$ lies on $\boldsymbol{l}_1$ the following holds:

$$a_1 x_1 + b_1 y_1 + c_1 = 0 \tag{1}$$

Thus we have:

$$\widetilde{\boldsymbol{m}}_1^\top \boldsymbol{l}_1 = (x_1, x_2, 1)^\top \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} = x_1 \cdot a_1 + y_1 \cdot b_1 + 1 \cdot c_1 \overset{(1)}{=} 0 \ .$$

(b) Since the intersection point $\boldsymbol{m}_1$ lies on $\boldsymbol{l}_1$ as well as on $\boldsymbol{l}_2$, we know from (a) that $\widetilde{\boldsymbol{m}}_1^\top \boldsymbol{l}_1 = 0$ and $\widetilde{\boldsymbol{m}}_1^\top \boldsymbol{l}_2 = 0$ holds. Furthermore we know, that the inner product between two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is 0 iff $\boldsymbol{a} \perp \boldsymbol{b}$. Thus we have:

$$\widetilde{\boldsymbol{m}}_1 \perp \boldsymbol{l}_1 \quad \text{and} \quad \widetilde{\boldsymbol{m}}_1 \perp \boldsymbol{l}_2$$

That means $\widetilde{\boldsymbol{m}}_1$ is a vector which is orthogonal to the vectors $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$ at the same time. The vectors $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$ are linearly independent since $\boldsymbol{l}_1 \nparallel \boldsymbol{l}_2$. Hence, the vector $\widetilde{\boldsymbol{m}}_1$ is up to a scalar factor λ uniquely given by the crossproduct of $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$:

$$\begin{aligned} \widetilde{\boldsymbol{m}}_1 &= \lambda(\boldsymbol{l}_1 \times \boldsymbol{l}_2) \\ \Leftrightarrow \quad \frac{1}{\lambda}\widetilde{\boldsymbol{m}}_1 &= \boldsymbol{l}_1 \times \boldsymbol{l}_2 \end{aligned}$$

Due to the depth ambiguity this yields

$$\widetilde{\boldsymbol{m}}_1 = \boldsymbol{l}_1 \times \boldsymbol{l}_2$$

Alternatively:

If $\boldsymbol{m}_1$ is the intersection point of $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$ we know that following system of equations holds:

$$\begin{aligned} a_1 x_1 + b_1 y_1 + c_1 &= 0 \\ a_2 x_1 + b_2 y_1 + c_2 &= 0 \end{aligned}$$

Solving it for x_1 and y_1 gives:

$$x_1 = \frac{b_1c_2 - b_2c_1}{a_1b_2 - a_2b_1} \quad \text{and} \quad y_1 = \frac{a_2c_1 - a_1c_2}{a_1b_2 - a_2b_1}$$

Note that $a_1b_2 - a_2b_1 \neq 0$ due to $\boldsymbol{l}_1 \nparallel \boldsymbol{l}_2$. So

$$\widetilde{\boldsymbol{m}}_1 = \begin{pmatrix} \frac{b_1c_2 - b_2c_1}{a_1b_2 - a_2b_1} \\ \frac{a_2c_1 - a_1c_2}{a_1b_2 - a_2b_1} \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} b_1c_2 - b_2c_1 \\ a_2c_1 - a_1c_2 \\ a_1b_2 - a_2b_1 \end{pmatrix} = \lambda(\boldsymbol{l}_1 \times \boldsymbol{l}_2)$$

with $\lambda = \dfrac{1}{a_1b_2 - a_2b_1}$.

So $\frac{1}{\lambda}\widetilde{\boldsymbol{m}}_1 = \boldsymbol{l}_1 \times \boldsymbol{l}_2$ and the depth ambiguity yield

$$\widetilde{\boldsymbol{m}}_1 = \boldsymbol{l}_1 \times \boldsymbol{l}_2$$

(c) Since both points, $\boldsymbol{m}_1$ as well as $\boldsymbol{m}_2$ lie on $\boldsymbol{l}_1$ we know from (a) that $\widetilde{\boldsymbol{m}}_1^\top \boldsymbol{l}_1 = 0$ and $\widetilde{\boldsymbol{m}}_2^\top \boldsymbol{l}_1 = 0$ holds. We get again:

$$\widetilde{\boldsymbol{m}}_1 \perp \boldsymbol{l}_1 \quad \text{and} \quad \widetilde{\boldsymbol{m}}_2 \perp \boldsymbol{l}_1$$

The vectors $\widetilde{\boldsymbol{m}}_1$ and $\widetilde{\boldsymbol{m}}_2$ are linearly independent since $\boldsymbol{m}_1 \neq \boldsymbol{m}_2$. So the vector $\boldsymbol{l}_1$ is up to a scalar factor λ uniquely given by the crossproduct of $\widetilde{\boldsymbol{m}}_1$ and $\widetilde{\boldsymbol{m}}_2$:

$$\begin{aligned} \boldsymbol{l}_1 &= \lambda(\widetilde{\boldsymbol{m}}_1 \times \widetilde{\boldsymbol{m}}_2) \\ &= (\lambda\widetilde{\boldsymbol{m}}_1) \times \widetilde{\boldsymbol{m}}_2 \\ &= \widetilde{\boldsymbol{m}}_1 \times (\lambda\widetilde{\boldsymbol{m}}_2) \end{aligned}$$

The depth ambiguity yields

$$\boldsymbol{l}_1 = \widetilde{\boldsymbol{m}}_1 \times \widetilde{\boldsymbol{m}}_2$$

Alternatively:

We want to show, that $\boldsymbol{l}_1$ is given by:

$$\boldsymbol{l}_1 = \widetilde{\boldsymbol{m}}_1 \times \widetilde{\boldsymbol{m}}_2 = \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \times \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} y_1 - y_2 \\ x_2 - x_1 \\ x_1y_2 - x_2y_1 \end{pmatrix}$$

i.e.

$$(y_1 - y_2)x + (x_2 - x_1)y + (x_1y_2 - x_2y_1) = 0$$

Since a line is uniquely defined by two points it suffices to show, that the points $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ lie on $\boldsymbol{l}_1$.

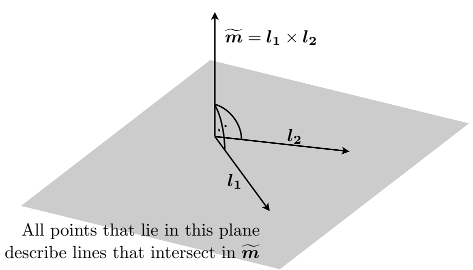
Plugging in $\boldsymbol{m}_1$:

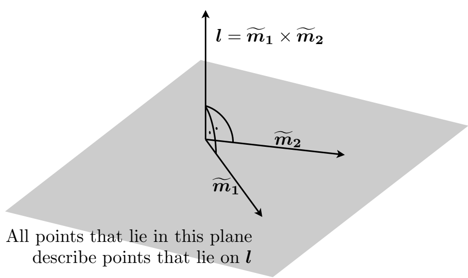
$$\begin{aligned} & (y_1 - y_2)x_1 + (x_2 - x_1)y_1 + (x_1y_2 - x_2y_1) \\ = \; & y_1x_1 - y_2x_1 + x_2y_1 - x_1y_1 + x_1y_2 - x_2y_1 \\ = \; & (y_1x_1 - x_1y_1) + (x_1y_2 - y_2x_1) + (x_2y_1 - x_2y_1) \\ = \; & 0 \end{aligned}$$

Plugging in $\boldsymbol{m}_2$:

$$\begin{aligned} & (y_1 - y_2)x_2 + (x_2 - x_1)y_2 + (x_1y_2 - x_2y_1) \\ = \; & y_1x_2 - y_2x_2 + x_2y_2 - x_1y_2 + x_1y_2 - x_2y_1 \\ = \; & (y_1x_2 - x_2y_1) + (x_2y_2 - y_2x_2) + (x_1y_2 - x_1y_2) \\ = \; & 0 \end{aligned}$$

Last but not least consider the following two pictures which summarise what we have learned in this assignment:





Problem 2: (Fundamental Matrix for the Orthoparallel Camera Setup)

We want to exploit the formula $\boldsymbol{l}_2 = \boldsymbol{F}\tilde{\boldsymbol{m}}_1$ in order to get the fundamental matrix for the orthoparallel case. To this end we first collect some facts with respect to the epipolar lines: The vector $\boldsymbol{l}_2 = (a, b, c)^\top$ describes the epipolar line $ax + by + c = 0$. In the orthoparallel case all epipolar lines are parallel to the image x-axis. Thus b cannot be 0. Furthermore we can reformulate $ax + by + c = 0$:

$$y = -\frac{a}{b}x - \frac{c}{b}$$

As the slope $-\frac{a}{b}$ of the lines has to be 0 we know that $a = 0$. In addition we see that the y-intercept of a line is given by $-\frac{c}{b}$. For a point $\boldsymbol{m}_1 = (x, y)^\top$ and its corresponding epipolar line $\boldsymbol{l}_2$ it must hold that $y = -\frac{c}{b} \Leftrightarrow c = -by$. So in the orthoparallel case the epipolar line $\boldsymbol{l}_2$ of a point $\boldsymbol{m}_1 = (x, y)^\top$ is given as

$$\boldsymbol{l}_2 = (0, b, -by)^\top$$

Provided this knowledge we get

Point $\boldsymbol{m}_1 = (x, y)^\top$	Corresponding epipolar line $\boldsymbol{l}_2$
$(0, 0, 1)^\top$	$(0, b, -b \cdot 0)^\top = (0, b, 0)^\top$
$(1, 0, 1)^\top$	$(0, b, -b \cdot 0)^\top = (0, b, 0)^\top$
$(0, 1, 1)^\top$	$(0, b, -b \cdot 1)^\top = (0, b, -b)^\top$

Now we compute $\boldsymbol{F}\tilde{\boldsymbol{m}}_1$ and compare the result with the expected line $\boldsymbol{l}_2$ according to the table:

$$\boldsymbol{F}\begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix}f_{1,3}\\f_{2,3}\\f_{3,3}\end{pmatrix} \overset{!}{=} \begin{pmatrix}0\\b\\0\end{pmatrix} \quad\Rightarrow\quad \begin{array}{c}f_{1,3} = 0\\f_{2,3} = b\\f_{3,3} = 0\end{array} \tag{2}$$

$$\boldsymbol{F}\begin{pmatrix}1\\0\\1\end{pmatrix} = \begin{pmatrix}f_{1,1} + f_{1,3}\\f_{2,1} + f_{2,3}\\f_{3,1} + f_{3,3}\end{pmatrix} \overset{(1)}{=} \begin{pmatrix}f_{1,1}\\f_{2,1} + b\\f_{3,1}\end{pmatrix} \overset{!}{=} \begin{pmatrix}0\\b\\0\end{pmatrix} \quad\Rightarrow\quad \begin{array}{c}f_{1,1} = 0\\f_{2,1} = 0\\f_{3,1} = 0\end{array} \tag{3}$$

$$\boldsymbol{F}\begin{pmatrix}0\\1\\1\end{pmatrix} = \begin{pmatrix}f_{1,2} + f_{1,3}\\f_{2,2} + f_{2,3}\\f_{3,2} + f_{3,3}\end{pmatrix} \overset{(1)}{=} \begin{pmatrix}f_{1,2}\\f_{2,2} + b\\f_{3,2}\end{pmatrix} \overset{!}{=} \begin{pmatrix}0\\b\\-b\end{pmatrix} \quad\Rightarrow\quad \begin{array}{c}f_{1,2} = 0\\f_{2,2} = 0\\f_{3,2} = -b\end{array} \tag{4}$$

Hence by (1),(2),(3) we get

$$\boldsymbol{F} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & b \\ 0 & -b & 0 \end{pmatrix}$$

Note that $\boldsymbol{F}$ is defined up to a scaling factor. Hence b can be chosen arbitrarily. In the literature b is often set to 1:

$$\boldsymbol{F} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}$$

Problem 3: (Stereo Reconstruction)

The right way to solve this exercise is to go backwards through the formula given in lecture 25 on slide 17 for each of the cameras. We are given the image coordinates

$$x_1 = \left(2,\ \frac{6}{5}\right)^\top$$
$$x_2 = \left(\frac{3}{4},\ \sqrt{2}\right)^\top$$

that we first transform into homogenuous coordinates

$$\widehat{x}_1 = \left(2,\ \frac{6}{5}, 1\right)^\top$$
$$\widehat{x}_2 = \left(\frac{3}{4},\ \sqrt{2}, 1\right)^\top$$

Now we apply the inverse of the matrix containing the intrinsic parameters.

$$\left(A_1^{\text{int}}\right)^{-1} = \left(A_2^{\text{int}}\right)^{-1} = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

This yields

$$\widehat{x}_{i1} := \left(1,\ \frac{6}{5}, 1\right)^\top$$
$$\widehat{x}_{i2} := \left(-\frac{1}{4},\ \sqrt{2}, 1\right)^\top$$

This means that the projection lines we are looking for are given in camera coordinates by

$$
\begin{aligned}
X_1 &:= \lambda_1 \widehat{x}_{i1} \\
X_2 &:= \lambda_2 \widehat{x}_{i2}
\end{aligned}
$$

Note that we don't have to take care of the focal length as it is 1 for each camera. Now we transform these vectors in 3-D homogeneous coordinates.

$$
\begin{aligned}
\widehat{X}_1 &:= \left(\lambda_1,\ \frac{6}{5}\lambda_1, \lambda_1, 1\right)^\top \\
\widehat{X}_2 &:= \left(-\lambda_2 \frac{1}{4},\ \lambda_2\sqrt{2}, \lambda_2, 1\right)^\top
\end{aligned}
$$

and apply the inverse of the extrinsic camera parameters matrix.

$$
\left(A_1^{\text{ext}}\right)^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \left(A_2^{\text{ext}}\right)^{-1} = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 2 \\ 0 & 1 & 0 & -2 \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

It is actually easy to calculate the inverse extrinsic matrix of the second camera. We see that the extrinsic matrix is given by a rotation around the y-axis around -30 degrees, followed by a translation by the vector $(-\sqrt{2}, 2, -\sqrt{2})^\top$. That means the inverse is given by the concatenation of the inverse operations in inverted order, so a translation by $(2\sqrt{2}, -4, \sqrt{2})^\top$, followed by a rotation around the y-axis by 30 degrees. This yields the given inverse extrinsic matrix. A multiplication with the projections gives us

$$
\begin{aligned}
\left(A_1^{\text{ext}}\right)^{-1} \widehat{X}_1 &= \left(\lambda_1,\ \frac{6}{5}\lambda_1, \lambda_1, 1\right)^\top \\
\left(A_2^{\text{ext}}\right)^{-1} \widehat{X}_2 &= \left(\frac{3\lambda_2}{4\sqrt{2}} + 2, \sqrt{2}\lambda_2 - 2, \frac{5\lambda_2}{4\sqrt{2}}, 1\right)^\top
\end{aligned}
$$

These are the projection lines in homogeneous 3D World coordinates. The intersection point of the corresponding line equations yields the original scene point. The lines intersect if we can find λ_1 and λ_2 such that all the coordinates are equal. This yields 3 equations with 2 unknowns:

$$\begin{aligned}\lambda_1 &= \frac{3\lambda_2}{4\sqrt{2}} + 2\\ \frac{6}{5}\lambda_1 &= \sqrt{2}\lambda_2 - 2\\ \lambda_1 &= \frac{5\lambda_2}{4\sqrt{2}}.\end{aligned}$$

Plugging the third equation into the first yields $\lambda_2 = 4\sqrt{2}$, and the third equation yields $\lambda_1 = 5$. We never used the second equation but λ_1 and λ_2 satisfy it. Plugging $\lambda_1 = 5$ into the first line gives the original point $(5, 6, 5)^\top$. Thus the sought depth is 6.

Problem 4: (Correlation-Based Stereo Method)

(a) In the template file `corTemplate.c` three code pieces had to be added. The correct solution reads

```
/* -------------------------------------------------------- */
void mean_window

     (long     n,         /* window size 2*n+1              */
      long     nx,        /* image dimension in x direction */
      long     ny,        /* image dimension in y direction */
      float    **f,       /* input image                    */
      float    **f_mean)  /* output: mean image             */

/*
 computes mean with window of size (2*n+1)*(2*n+1)
*/

{
long    i, j, k, l;           /* loop variables */
long    nn;                   /* number of pixels in window */

/* number of pixels in window */
nn=(2*n+1)*(2*n+1);

/* sum up and compute mean (borders are left out) */
for (i=1+n; i<=nx-n; i++)
   for (j=1+n; j<=ny-n; j++)
     {
      f_mean[i][j]=0;

      /* sum up */
        for (k=-n; k<=n; k++)
           for (l=-n; l<=n; l++)
              f_mean[i][j]+=f[i+k][j+l];

      /* compute mean */
      f_mean[i][j]=f_mean[i][j]/nn;
     }
```

```
return;

} /* mean_window */
/* ------------------------------------------------------- */


/* ------------------------------------------------------- */
void sum_window

     (long     n,         /* window size 2*n+1               */
      long     nx,        /* image dimension in x direction */
      long     ny,        /* image dimension in y direction */
      float    **f1,      /* input image 1                   */
      float    **f2,      /* input image 2                   */
      float    **f1_mean, /* window mean for image 1         */
      float    **f2_mean, /* window mean for image 2         */
      long     s,         /* shift of second image           */
      float    **f_sum)   /* output: mean compensated sum    */

/*
 sums up mean compensated product of a first and a shifted
 second image
*/

{
long    i, j, k, l;           /* loop variables */

/* for each pixel */
for (i=1+n+s; i<=nx-n; i++)
   for (j=1+n; j<=ny-n; j++)
     {
      f_sum[i][j]=0;

      /* sum up expression */
      for (k=-n; k<=n; k++)
         for (l=-n; l<=n; l++)
            f_sum[i][j]+=( (f1[i+k  ][j+l]-f1_mean[i  ][j])
                          *(f2[i+k-s][j+l]-f2_mean[i-s][j]) );
    }
return;
```

```
} /* sum_window */
/* ---------------------------------------------------------- */


/* ---------------------------------------------------------- */
void correlation

     (long     nx,        /* image dimension in x direction */
      long     ny,        /* image dimension in y direction */
      float    **f1,      /* input image 1                  */
      float    **f2,      /* input image 2                  */
      long     max_disp,  /* max. disparity                 */
      long     n,         /* window size (2*n+1)*(2*n+1)    */
      float    ***cor)    /* out: correlation value for all */
                          /*     pixels and all disparities */

/*
 Computes the correlation between square neigbourhoods of all
 pixels in the first frame and all shifted neighbourhoods
 along the x-axis  in the second frame.
*/

{
long    i, j, k, l;   /* loop variables */
float   help;         /* temporary variable */
float   **f1_mean;    /* weighted mean values of
                         neighbourhoods in first image */
float   **f2_mean;    /* weighted mean values of
                         neighbourhoods in second image */
float   **f1f1_sum;   /* summed up squared mean compensated
                         first image*/
float   **f2f2_sum;   /* summed up squared mean  compensated
                         second image */
float   ***f1f2s_sum; /* summed up product between mean
                         compensated first image and shifted
                         mean compensated second image
                         (for all disparities) */

/* ---- allocate storage ---- */
alloc_matrix (&f1_mean, nx+2, ny+2);
alloc_matrix (&f2_mean, nx+2, ny+2);
```

```
alloc_matrix (&f1f1_sum, nx+2, ny+2);
alloc_matrix (&f2f2_sum, nx+2, ny+2);
alloc_cubix  (&f1f2s_sum, max_disp+1, nx+2, ny+2);



/* ---- compute neighbourhood means ---- */
mean_window(n, nx, ny, f1, f1_mean);
mean_window(n, nx, ny, f2, f2_mean);

/* ---- compute all entries for the correlation ---- */
sum_window(n, nx, ny, f1, f1, f1_mean, f1_mean, 0, f1f1_sum);
sum_window(n, nx, ny, f2, f2, f2_mean, f2_mean, 0, f2f2_sum);

for (k=0; k<=max_disp; k++)
  sum_window(n, nx, ny, f1, f2, f1_mean, f2_mean,
             k, f1f2s_sum[k]);

/* ---- compute correlation ---- */

/* initialise correlation with -1.0 */
for (k=0; k<=max_disp; k++)
   for (i=1; i<=nx; i++)
      for (j=1; j<=ny; j++)
         cor[k][i][j]=-1.0;

/* compute correlation */
for (k=0; k<=max_disp; k++)
   for (i=1+k+n; i<=nx-n; i++)
      for (j=1+n; j<=ny-n; j++)
        {
          /* if denominator is non-zero */
          if  (f1f1_sum[i][j]*f2f2_sum[i-k][j]!=0.0)
           {
            cor[k][i][j] =
              f1f2s_sum[k][i][j] / (  sqrt(f1f1_sum[i  ][j])
                                    * sqrt(f2f2_sum[i-k][j]) );
           }
          /* else is handled by initialisation */
      }
```

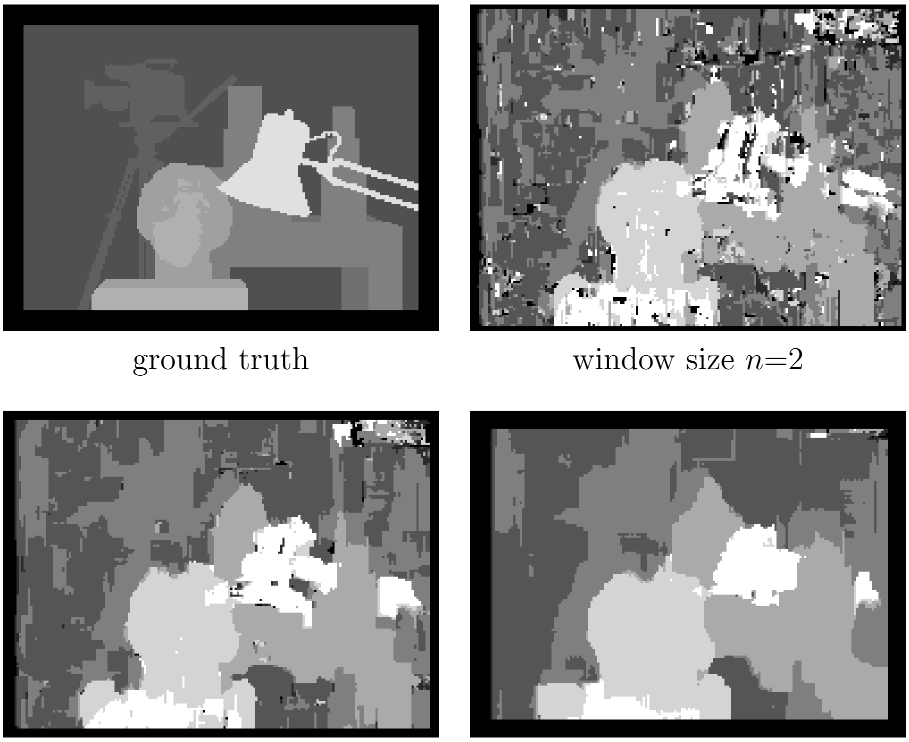
```
/* ---- disallocate storage ---- */
disalloc_matrix (f1_mean, nx+2, ny+2);
disalloc_matrix (f2_mean, nx+2, ny+2);
disalloc_matrix (f1f1_sum, nx+2, ny+2);
disalloc_matrix (f2f2_sum, nx+2, ny+2);
disalloc_cubix  (f1f2s_sum, max_disp+1, nx+2, ny+2);

return;
} /* correlation */
/* ---------------------------------------------------------------- */
```

Now we turn our attention to the experimental results. The left and right image of the Tsukuba stereo pair are shown below.

tsu_l.pgm

tsu_r.pgm

(b) Let us investigate what happens if we increase the window size. As one can see, outliers disappear and the estimation becomes more homogeneous. However, at the same time edges become dislocated and the result loses sharpness.



ground truth | window size n=2

window size n=4 | window size n=8

(c) Finally, we study what happens if we shift the grey values of one image by 50. To this end, we replace the right image by the corresponding shifted variant. As one can see from the obtained disparity maps, the results are basically the same. Only at locations where the original image was already very bright (e.g. the statue in the foreground) small differences occur. This however, is due to the limitation of the brightest grey value to 255 in the PGM format (saving the modified image after rescaling sets all values larger than 255 to 255). Since the correlation windows are compensated by their mean value, a shift of the grey values in one or both images does not change the result. In fact, the normalisation in the correlation does even allow the grey values of both images to undergo an affine transformation without changing the outcome.

`tsu_r.pgm`

`tsu_r_mod.pgm`

original, window size n=4

modified, window size n=4