Example Solutions for Homework Assignment 13 (H13)

Problem 1: (Transformation Matrices)

In Lecture 25, the examples of a rotation around the z-axis and of a translation was given. We have to adapt this for rotations around the x-axis and y-axis.

Even if there was no rotation direction specified in the problem, we want to rotate in the mathematical positive sense, that means counter-clockwise. To understand this, we use a right-handed coordinate system as shown in Fig. 1. All later considerations will also work for a left-handed system, except that all rotation directions are exchanged. To explain a rotation around the z-axis, for instance, we take a look at the $x-y$-plane from the positive z-axis. This is shown in Fig. 2.

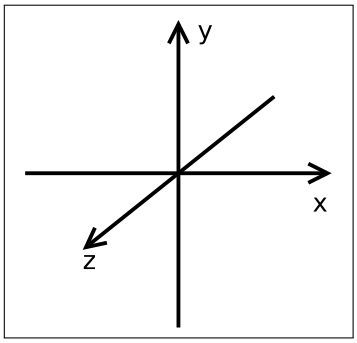


Figure 1: Right-handed coordinate system.

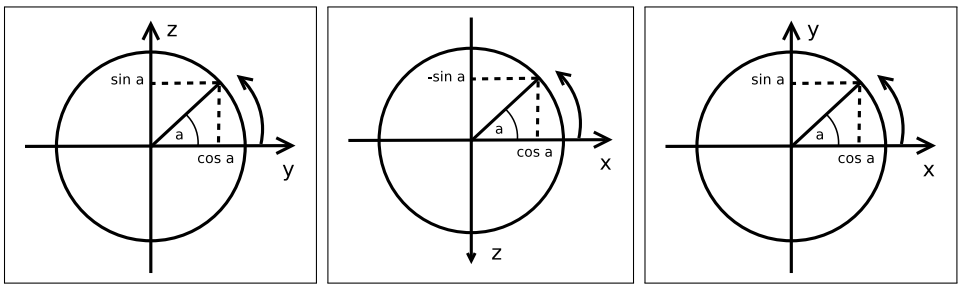


Figure 2: Rotation directions for the three axes. **Left:** Around x-axis. **Middle:** Around y-axis. **Right:** Around z-axis. We see that the y-axis differs from the other two.

We see that for counter-clockwise rotation, we have to distinguish rotations around the y-axis from those around x- and z-axes. We keep in mind that

the image of the standard unit vectors under a linear mapping can be found in the columns of the matrix. Together with Fig. 2 this explains why a rotation around the x- and z-axes looks like

$$R_x(\varphi) := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) & 0 \\ 0 & \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and

$$R_z(\varphi) := \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) & 0 & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

while a rotation around the y-axis is given by

$$R_y(\varphi) := \begin{pmatrix} \cos(\varphi) & 0 & \sin(\varphi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\varphi) & 0 & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Now we want to compute a transformation matrix in homogeneous coordinates which describes a rotation around the y-axis through an angle of 45 degrees followed by a translation with a vector $(-1, 2, -3)^\top$ and a rotation around the x-axis through an angle of -60 degrees.

The rotation around the y-axis through an angle of 45 is given by the matrix

$$R_1 := \begin{pmatrix} \cos(45°) & 0 & \sin(45°) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(45°) & 0 & \cos(45°) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad .$$

The translation by a vector $(-1, 2, -3)^\top$ is given by the matrix

$$T := \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad .$$

The rotation around the x-axis through an angle of -60 degrees is described

by the matrix

$$R_2 := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-60°) & -\sin(-60°) & 0 \\ 0 & \sin(-60°) & \cos(-60°) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(60°) & \sin(60°) & 0 \\ 0 & -\sin(60°) & \cos(60°) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

In a matrix-vector product, the vector is always multiplied from the right-hand side to the matrix. That means the rightmost transformation matrix is applied first to the vector, then its left neighbour etc. Transformations are applied from right to left. Thus the overall transformation matrix is then given by the matrix multiplication R_2TR_1:

$$R_2TR_1 = \begin{pmatrix} \cos(45°) & 0 & \sin(45°) & -1 \\ -\sin(60°)\sin(45°) & \cos(60°) & \sin(60°)\cos(45°) & 2\cos(60°) - 3\sin(60°) \\ -\cos(60°)\sin(45°) & -\sin(60°) & \cos(60°)\cos(45°) & 2\sin(60°) - 3\cos(60°) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & -1 \\ -\frac{\sqrt{6}}{4} & \frac{1}{2} & \frac{\sqrt{6}}{4} & 1 - \frac{3\sqrt{3}}{2} \\ -\frac{\sqrt{2}}{4} & -\frac{\sqrt{3}}{2} & \frac{\sqrt{2}}{4} & -\sqrt{3} - \frac{3}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

It is important to respect the order in which the matrices are multiplied as matrix multiplication does not commute.

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:

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

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

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

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

$$x_1 = \left(\frac{21}{2},\ \frac{1}{2}\right)^\top$$
$$x_2 = \left(\frac{19}{2},\ \frac{\sqrt{2}}{2}\right)^\top$$

that we first transform into homogenuous coordinates

$$\widehat{x}_1 = \left(\frac{21}{2},\ \frac{1}{2},\ 1\right)^\top$$
$$\widehat{x}_2 = \left(\frac{19}{2},\ \frac{\sqrt{2}}{2},\ 1\right)^\top$$

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

$$\left(A_1^{\text{int}}\right)^{-1} = \left(A_2^{\text{int}}\right)^{-1} = \begin{pmatrix} 1 & 0 & -10 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

This yields

$$\widehat{x}_{i1} := \left(A_1^{\text{int}}\right)^{-1} \widehat{x}_1 = \left(\frac{1}{2}, \frac{1}{2}, 1\right)^\top$$
$$\widehat{x}_{i2} := \left(A_2^{\text{int}}\right)^{-1} \widehat{x}_2 = \left(-\frac{1}{2}, \frac{1}{\sqrt{2}}, 1\right)^\top$$

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

$$X_1 := \lambda_1 \widehat{x}_{i1}$$
$$X_2 := \lambda_2 \widehat{x}_{i2}$$

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 homogenuous coordinates.

$$\widehat{X}_1 := \left(\lambda_1 \frac{1}{2}, \lambda_1 \frac{1}{2}, \lambda_1, 1\right)^\top$$
$$\widehat{X}_2 := \left(-\lambda_2 \frac{1}{2}, \lambda_2 \frac{1}{\sqrt{2}}, \lambda_2, 1\right)^\top$$

and apply the inverse of the extrinsic camera parameters matrix.

$$\left(A_1^{\text{ext}}\right)^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \left(A_2^{\text{ext}}\right)^{-1} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & \frac{5}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & -\frac{5}{\sqrt{2}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

which gives us

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

These are the projection lines in homogenuous 3D World coordinates. The intersection point of the corresponding line equations yields the original scene point. A quick computation yields that $\lambda_1 = \lambda_2 = 10$ and therefore, by plugging in, we find that the original point was $(5, 5, 10)^\top$. Thus the sought depth is 10.

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 */
/* ------------------------------------------------------------------ */
```

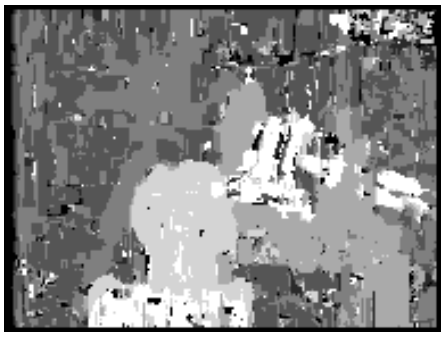
Now we turn our attention to the experimental results. The left and right image of the Tsukuba stereo pair are shown below.

`tsu_l.pgm`

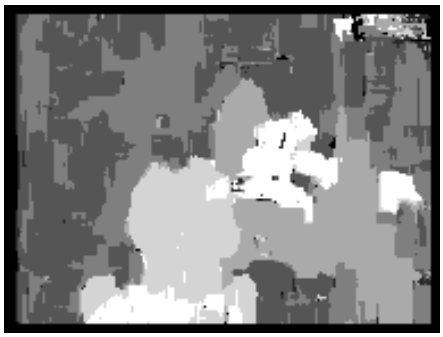
`tsu_r.pgm`

(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.

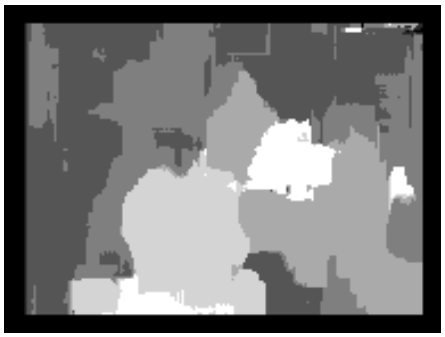
ground truth



window size n=2



window size n=4



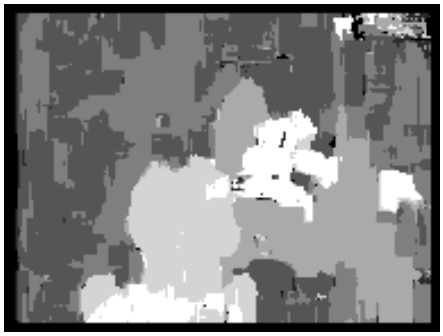
window size n=8

(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

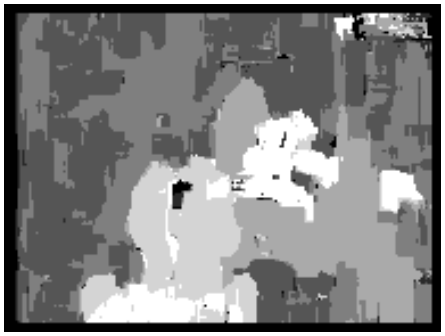
results are basically the same. Only at locations where the original image was already very bright (e.g. the statue in the foreground) small differences occur. This however, is due to the limitation of the brightest grey value to 255 in the PGM format (saving the modified image after rescaling sets all 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