Image Processing and Computer Vision 2005/06
Example Solutions for Practical Assignments 7 (P7)

Problem 1

In the template file `corTemplate.c` three code pieces had to be added. The correct solution is give by

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

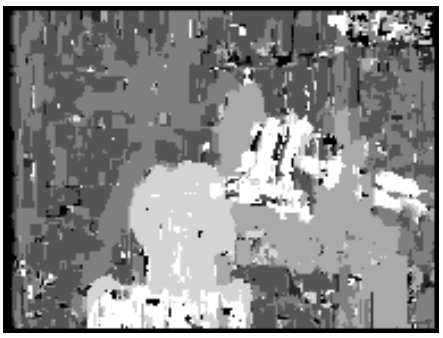
Now we turn our attention to the experimental results. The left and right image of the Tsukuba stereo pair are shown below.

tsu_l.pgm

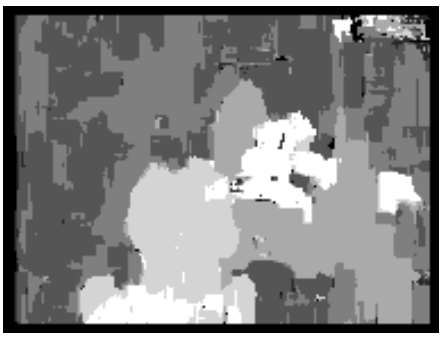
tsu_r.pgm

(b) Let us now investigate what happens if we increase the windows 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.

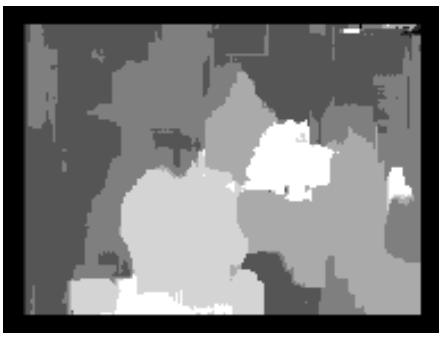
ground truth



window size n=2



window size n=4

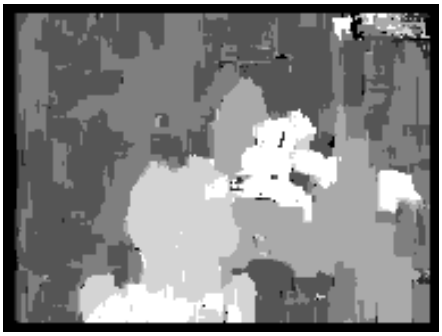


window size n=8

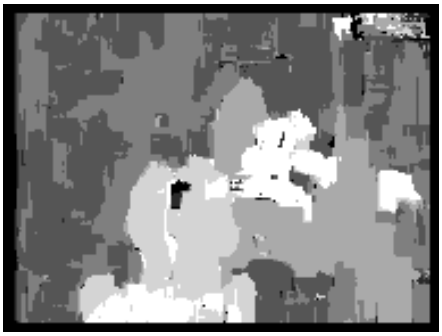
(c) Finally, we study what happens if we shift the grey values of one image by 50. To this end, we replace the right image by the corresponding shifted variant. As one can see from the obtained disparity maps, the results are basically the same. Only at locations where the original image was already very bright (e.g. at 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 hast set all values larger than 255 to 255). Since the correlation windows are compensated by their mean value, a shift of the grey values 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 tranformation without changing the result.

`tsu_r.pgm`

`tsu_r_mod.pgm`



original, winow size n=4



modified, window size n=4