Image Processing and Computer Vision 2005/06

Example Solutions for Practical Assignments 2 (P2)

Problem 1.

In order to apply the affine transformation $f(x) = mx + n$, we have to determine the slope m and the offset n first. Since we know that $F(x_{min}) = a$ and $F(x_{max}) = b$ we have two linear equations for the two unknowns. Solving this 2×2 linear system of equations yields $m = \frac{b-a}{x_{max}-x_{min}}$ and $n = a - x_{\mathrm{min}} \frac{b-a}{x_{max}-x_{min}}$.

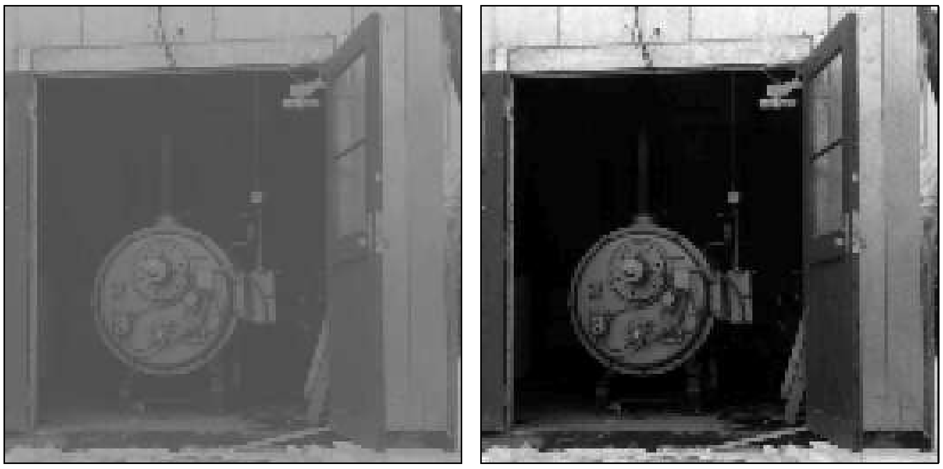

Figure 1: *Left:* Original image. *Right:* Transformed image ($a = 0$, $b = 255$).

```
/* ------------------------------------------------------------ */
void rescale

     (double  **u,        /* input image, range [0,255] */
      long    nx,         /* size in x direction */
      long    ny,         /* size in y direction */
      double  a,          /* smallest transformed grey level */
      double  b,          /* largest transformed grey level */
      double  *g)         /* transformed grey levels */

/*
 affine rescaling of the grey values of u such that
 min(u) -> a, and max(u) -> b.
*/
```

```
{
long    i, j;       /* loop variables */
double  min, max;   /* extrema of u */
double  m,n;        /* time saver */

/* determine extrema of u */
min = max = u[1][1];
for (i = 1; i <= nx; i++)
{
  for (j = 1; j <= ny; j++)
  {
    if (u[i][j] < min) min = u[i][j];
    if (u[i][j] > max) max = u[i][j];
  }
}

/* compute parameters */
m = (b-a) / (max-min);
n = a-min*m;

/* determine mapping for all grey values */
for (i = 0; i <= 255; i++)
    g[i] = m*i+n;

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

Problem 2.

Here, we apply the gamma correction function from the lecture.

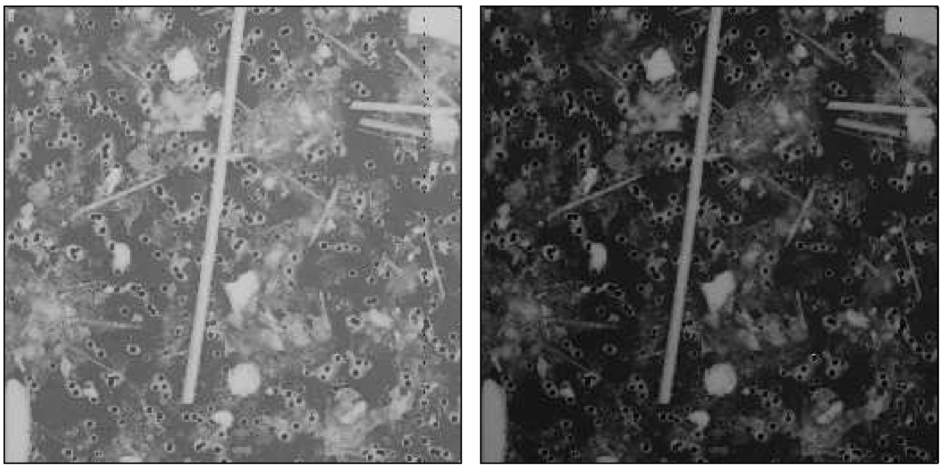

Figure 2: *Left:* Original image. *Right:* Transformed image ($\gamma = 0.4$).

```
/* ------------------------------------------------------------ */
void gamma_correct

     (double  gamma,      /* gamma correction factor */
      double  *g)         /* transformed grey levels */

/*
 applies gamma correction to the 256 grey levels that may appear
 in byte wise coded images
*/

{
long  k;   /* loop variable */

/* determine mapping for all grey values */
for (k = 0; k <= 255; k++)
    g[k] = 255.0 * pow(k/255.0, 1.0/gamma);

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

Problem 3.

Here, we implement the histogram equalisation from the lecture. Two algorithms are presented. The first one maps the grey values directly if the inequality holds (early map). The second one is closer to the lecture and maps the grey values *en block* after the inequality is violated (late map).

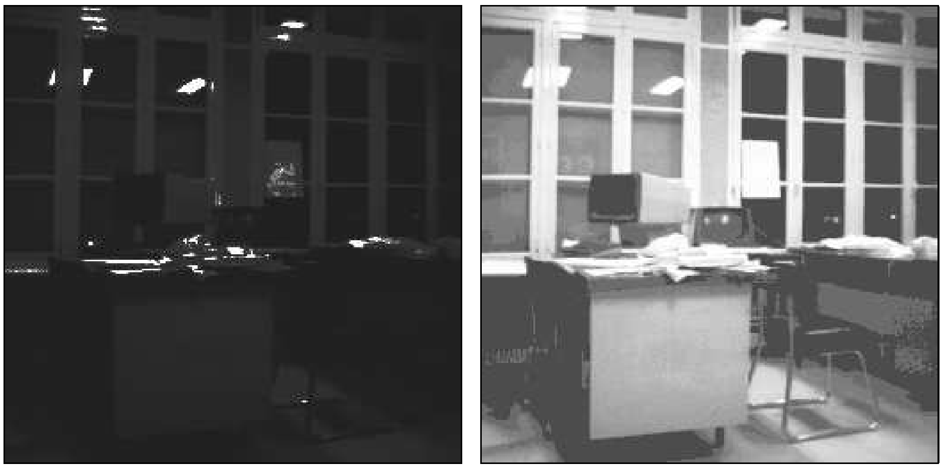

Figure 3: *Left:* Original image. *Right:* Transformed image.

```
/* ----------------------------------------------------------- */
void hist_equal_early_map

     (double  **u,        /* input image, range [0,255] */
      long    nx,         /* size in x direction */
      long    ny,         /* size in y direction */
      double  *g)         /* transformed grey levels */

/* performs histogram equalization on u. */

{
long    i, j, k;          /* loop variables */
long    r;                /* current summation index r */
long    k_r;              /* current summation index k_r */
double  hist[256];        /* histogram */
long    n;                /* pixel number */
double  psum, qsum;       /* sums in equalisation algorithm */
/* initialise histogram with zero */
for (k = 0; k <= 255; k++)
    hist[k]=0;

/* create histogram of u with bin width 1 */
for (i = 1; i <= nx; i++)
  for (j = 1; j <= ny; j++)
    hist[(long) u[i][j]]++;

/* initialise psum and k_r */
k_r = 1;
psum = hist[0];

/* total number of pixels */
n = nx*ny;


for (r = 1; r <= 256; r++)
{
  /* determine right hand side */
  qsum = r * n / 256.0;

  /* increase left hand side as long as it is smaller or equal   */
  /* the right hand side; note that each k_r is only used once.  */
  for (; (k_r <= 256) && (psum <= qsum); k_r++)
  {
    /* perform mapping : we have to substract 1 from the grey */
    /* value index r and k_r to get the actual grey values */
    g[k_r-1] = r - 1;

    /* increase left hand side */
    psum += hist[k_r];

  }
}
}

/* ----------------------------------------------------------- */


/*------------------------------------------------------------*/

void hist_equal_late_map
```

```
     (double  **u,        /* input image, range [0,255] */
      long    nx,         /* size in x direction */
      long    ny,         /* size in y direction */
      double  *g)         /* transformed grey levels */

/* performs histogram equalization on u. */

{
long    i, j, k;          /* loop variables */
long    r;                /* current summation index r */
long    k_r;              /* current summation index k_r */
long    n;                /* pixel number */
double  hist[256];        /* histogram */
double  psum, qsum;       /* sums in equalisation algorithm */

/* initialise histogram with zero */
for (k = 0; k <= 255; k++)
    hist[k]=0;

/* create histogram of u with bin width 1 */
for (i = 1; i <= nx; i++)
  for (j = 1; j <= ny; j++)
    hist[(long) u[i][j]]++;

/* initialise psum and k_r */
k_r  = 1;
psum = hist[0];

/* total number of pixels */
n = nx*ny;

/* j is the index of the last grey value that has been mapped */
j   = 0;


for(r=1; r<=256; r++)
{
  /* determine right hand side */
  qsum = r * n / 256.0;

  /* search the largest index k_r such that the inequality is satisfied */
  while(psum <= qsum)
  {
    psum += hist[k_r];
    k_r  = k_r+1;
  }

  /* perform mapping from j which corresponds to k_{r-1} to k_r-1. */
  /* we have to substract 1 from the grey *value index r and k_r */
  /* to get the actual grey values */
  while(j < k_r-1)
  {
    /* perform mapping */
    g[j] = r-1;

    /* store last index that has been mapped (=k_{r-1})
    j    = j+1;
  }
}
}
/*--------------------------------------------------------------*/
```