Comp. 4900C: Assignment #3
Due: March 20, 2007

The goal of this assignment is to implement some code that performs calibration using the method described in the lectures; by first computing a projection matrix from a set of correspondences, and then decomposing that matrix to find the extrinsic and intrinsic parameters.

You can do this assignment in any language you like, but I believe that using Ch is the easiest solution for two reasons; first it is easy to manipulate arrays, and second I give you the code for the main routine in Ch, and your task is to write the code for a number of other subroutines and run the main program with three different sets of noise parameters.

The main code in Ch is below:

```
int main( int argc, char** argv )
{
   array double threedpoints[NUM_FEATURES][3];
   array double twodprojections[NUM_FEATURES][2];
   array double projectionmatrix[3][4];
   array double r[3][3];
   array double t[3], K[3][3];
   int i, j;

   /* compute the parameters for the projection matrix */
   eulerangletorotmatrix(.2, .4, .6, r);
   setkmatrix(K);
   settrans(t);
   composeprojectionmatrix(r, t, K, projectionmatrix);

   /* first compute some random 3d points */
   computerandomthreedpoints(threedpoints);
   /* apply the projection matrix to the 3d points */
   applyprojectionmatrix(threedpoints, twodprojections, projectionmatrix);
   /* add some noise to the 2d points */
   addnoiseto2dpoints(twodprojections, 4.0);
   /* test the application of the projection matrix */
   testprojectionmatrix(threedpoints, twodprojections, projectionmatrix);
   /* now compute a new projection matrix only from the 3d and 2d points */
   computeprojectionmatrix(threedpoints, twodprojections, projectionmatrix);
   /* check that the application of this new projection matrix works */
   testprojectionmatrix(threedpoints, twodprojections, projectionmatrix);
   /* decompose projection matrix to get extrinsic and intrsinic calibration */
   decomposeprojectionmatrix(projectionmatrix, r, t, K);
```

```
    /* compute best R matrix */
    findbestRmatrix(r, r);
    /* the R, T and K matrix should be same as was set originally */


}
```

The basic idea is to compute a projection matrix given value R, K and T. Then create some random 3d points, and project these 3d points into 2d using this projection matrix. These 3d to 2d point correspondences are then used to compute a new projection matrix, and finally this newly computed matrix is decomposed to find the final R, K and T. The two should be approximately equal.

I will give you the code for:

eulerangletorotmatrix, setkmatrix, settrans, computerandom3dpoints, and addnoiseto2dpoints.

Your job is to write the code for the following routines:

```
/* take the R, T, and K and create the projection matrix P as output */
void composeprojectionmatrix(array double r[3][3], array double t[3], array double
K[3][3], array double P[3][4])

/* apply the projection matrix to the 3d points to get a set of 2d feature points as output */
void applyprojectionmatrix(array double threedpoints[NUM_FEATURES][3],
                           array double twodprojections[NUM_FEATURES][2],
                           array double projectionmatrix[3][4])

/* take the projection matrix, apply it to the given 3d points, and print out the distance
between the newly projected points and the given twod point projections. Only a printout,
is necessary, since nothing is returned  */
void testprojectionmatrix(array double threedpoints[NUM_FEATURES][3],
                          array double twodprojections[NUM_FEATURES][2],
                          array double projectionmatrix[3][4])

/* given a set of correspondences from 3d to 2d points compute the projection matrix
   from these correspondence values. Output is projectionmatrix.
   Use the method described in 6.3 of Trucco and Verri */
void computeprojectionmatrix(array double threedpoints[NUM_FEATURES][3],
                             array double twodprojections[NUM_FEATURES][2],
                             array double projectionmatrix[3][4])


/* take the projection matrix and decompose it to find the rotation, translation and
calibration. Algorithm is taken from 6.3.2 of Trucco and Verri, Output is r, t, and K */
```

```
void decomposeprojectionmatrix(array double P[3][4], array double r[3][3], array double
t[3], array double K[3][3])

/* take the given rotation matrix (rin) which is not a true rotation matrix and make find
the closest true rotation matrix (rout) using the SVD approach. Again from Trucco and
Verri */
void findbestRmatrix(double array rin[3][3], double array rout[3][3])
```

Your goal is to write the routines, and to run the main program three times with different amounts of 2d noise:

```
   addnoiseto2dpoints(twodprojections, 0.0);
   addnoiseto2dpoints(twodprojections, 2.0);
   addnoiseto2dpoints(twodprojections, 4.0);
```

This is with increasing values of 2d noise, and the results should be slightly different.

In all three cases print out the original K, R, and T and the final computed K, R and T. Answer in writing the following question: what is the impact of increasing the 2d noise values on the computed K, R and T. Why is this the case?