Comp. 4900C: Assignment #3
Due: March ~~18~~ 7, 2008

The goal of this assignment is to implement some code that performs calibration using the method described in the text; by first computing a projection matrix, 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 subroutines.

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;

   eulerangletorotmatrix(.2, .4, .6, r);
   setkmatrix(K);
   settrans(t);
   composeprojectionmatrix(r, t, K, projectionmatrix);

   printf("Final r %f\n", r);
   //readinput("run.tr", camera0, allthreedpoints, twodpoints0);

   /* 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 applicaton 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 orignally since data has no noise */


}
```

The basic idea is to create some random 3d points, and compute a projection matrix for a given value R, K and T. Then project these 3d points into 2d using the computed projection matrix. These 3d to 2d point correspondences are used to recompute a new projection matrix, and this 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 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 3d points, and print out the distance between
the newly projected points and the given twod point projections. Only a printout, and no
output otherwise  */
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
   that produces 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 to fine the rotation, translation and
calibration. 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 rin matrix which is not a true rotation matrix and make it
   into the closest true rotation rout matrix. 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 make three runs with
    addnoiseto2dpoints(twodprojections, 0.0);
    addnoiseto2dpoints(twodprojections, 2.0);
    addnoiseto2dpoints(twodprojections, 4.0);

This is with increasing values of 2d noise.

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.