B1: Engineering and computation Project

Calibrating a digital camera for computer vision

1 Introduction

This project aims to use optimization to build a real world model of a camera suitable for use in computer vision. The model is built considering a fixed focus, pin hole camera design. A grid of points was used as a calibration object. First the camera model and a calibration grid is positioned somewhere in space. A noisy image of the grid is generated and some outliers added. A transformation matrix (which transforms points from an object frame to the camera frame) called a homography is estimated with the help of RANSAC to reject outliers. A number of images are used to estimate homographies from which a matrix (k-matrix) entailing the intrinsic properties of the camera model is built. This estimated k-matrix is then optimized to produce the model for the camera.

2 Building a camera model

2.1 K-matrix

The camera model consists of an intrinsic and an extrinsic model. $\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

where K is the K-matrix and $\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$ is the transformation matrix comprising of a rotation matrix R and a translation vector t.

The 'intrinsic model' or the camera 'K-matrix' and describes how the image in the unit camera is transformed into the real camera.

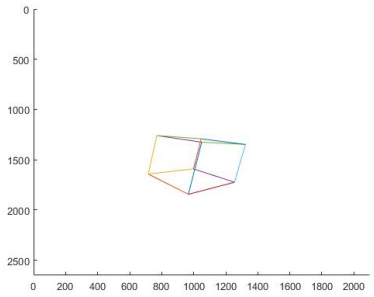
$$K = \begin{bmatrix} \alpha f & \gamma f & t_u \\ 0 & \beta f & t_v \\ 0 & 0 & 1 \end{bmatrix}$$

where αf is x focal length in pixels,βf is y focal length in pixels, γf is skewness, and t_u and t_v are principal points in sensor element steps

The K-matrix is built using the following 8 parameters: ChipWidth , ChipHeight , FocalLength , PixelWidth , PixelHeight , Skewness , P_u- The offset to the principal point as a fraction of the width ($t_u = P_u * ChipWidth$) P_v- The offset to the principal point as a fraction of the height ($t_v = P_v * ChipWidth$) The parameters are stored in a single vector which is then used by a function to test that the parameters are in range and satisfy design constraints. The parameters are then used to construct the k-matrix.

2.2 Transformation matrix

The rotation matrix was constructed using a matlab script that generates a random rotation matrix by starting with two random vectors and then normalizing, projecting out components and then generating the final column by using the cross - product. A random translation vector is also generated using the 'rand' function. The rotation and translation matrices are used to build the transformation matrix for placing the camera and the object in space. The ViewCamera function was used to view a cube using the previously built k-matrix to make sure the camera model was reasonable:



3 Building the calibration grid

The calibration grid consists of points describing a 1m by 1m grid with 10 mm tiles in the grid frame. The function BuildGrid was used to create the calibration grid.

```
%Calculate number of tiles in one row of the grid.
n = GridWidth/GridIncrement;
for j=1:n+1;

    %systematically fill in sections of length n+1 of Grid by
    %first defining all points with y coordinates y(1), then y(2), and so on
```

```
        Grid(1,(j-1)*(n+1)+1 : j*(n+1)) = x;
        Grid(2,(j-1)*(n+1)+1 : j*(n+1)) = y(j);
    end
```

4 Estimating a homography

Consider $x_2 = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix} & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

Since the z component is zero for all Points on the grid. Therefore, the transformation matrix which represents a homogeneous transformation between Points in the two planes (the calibration grid plane and the unit camera plane) in the extrinsic model can be written as

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$\begin{bmatrix} r_1 & r_2 & t \end{bmatrix}$ is known as a perspectivity and it represents a homogeneous transformation between Points in the two planes. The matrix that is the product of the k-matrix and this transformation matrix is called a homography. A homography is estimated by Constructing the camera model and the calibration grid. The grid is placed somewhere in space and a location for the camera is chosen using the FillImage function that allows us to fill the camera image by moving it closer to the grid. Then the camera is filled with a noisy image of the grid and point correspondences generated using BuildNoisyCorrespondences function. Correspondences are a set of pairs of vectors of the form $\begin{bmatrix} u \\ v \\ x \\ y \end{bmatrix}$ for each grid corner that lies inside the image. Some 'outliers' are then added by replacing $\begin{bmatrix} u \\ v \end{bmatrix}$ with a point somewhere in the image. Ransac estimation is then performed to reject the outliers and estimate the homography.

4.1 Noisy correspondences

The noisy correspondences are made using the matlab function BuildNoisyCorrespondences The function first transforms the calibration grid from the object frame to the camera frame. The grid points are transformed into 2-D homogeneous Points by removing the homogeneous scaler from the matrix, multiplying by the camera's KMatrix and store them into CameraGrid_2D_Hom. They are then normalized. The resulting points are called CameraGrid_uv. Then the points inside the camera's view are found using the following:

```
%Finding the size of the array required to store all the points
s = size(CameraGrid_2D_Hom, 2);
%Assign space for the matrix of points inside the camera.
InsidePoints_uv = zeros(2, s);
%Assign space for an array containing the index of all points within
%The camera's view (their column within CameraGrid_uv)
InsideIndex = zeros(1, s);
%Define an index to place coordinates in InsidePoints_uv
k = 1;
for i=1:s


    if CameraGrid_uv(1, i) > 0 && CameraGrid_uv(1, i) < (CameraWidth -1)
    ... && CameraGrid_uv(2, i) > 0 && CameraGrid_uv(2, i) < (CameraHeight -1)
    %If the [u,v] point is within the camera's view, place the
    %point in InsidePoints_uv and store its column index within InsideIndex
    InsidePoints_uv(:, k) = CameraGrid_uv(:,i);
    InsideIndex(k) = i;
    %Increment k
    k = k + 1;
    end

end
%subtract 1 form k to remove one extra increment at the end of the above loop
k=k-1;
```

then noise is added to the points. The index k is used to extract the [x,y] grid points from the calibration grid:

```
%Assign space for the corresponding points InsidePoints_xy = zeros(2,s);
for i = 1:k
    %Extract the index from InsideIndex
    j = InsideIndex(i);
    InsidePoints_xy(:, i) = CalibrationGrid(1:2, j);
end
```

finally the corresponding [u,v] and [x,y] grid points are stored in the correspondence matrix.

4.2 Ransac (Random sample consensus)

Ransac is the iterative method that is used to reject the outliers introduced in the correspondences which are then used to estimate the homography. The method rejects points using a user defined error and the probability of obtaining a reasonable result may be improved by increasing the number of runs.

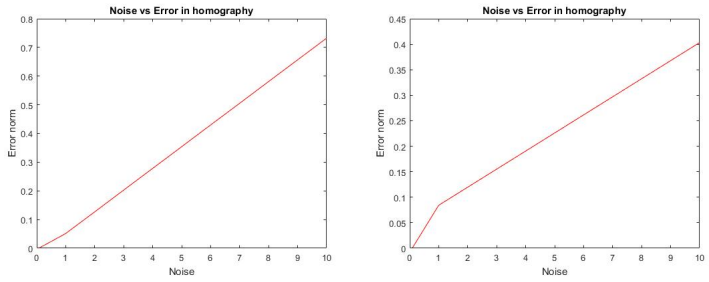
The matrix equation $\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ can be expressed as

$$\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -ux & -uy \\ 0 & 0 & 0 & x & y & 1 & -vx & -vy \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix}$$

The set of 4 points are taken from the correspondences and then used to estimate a homography. This homography is then used with object points (x,y) to estimate image points (u,v). These estimated image

points are then used with the actual image points from the correspondences to calculate an error if this error is below the user defined value, the index (the consensus set) of the said point is stored. This is done for all the points in the correspondence matrix and the outliers rejected. The remaining points are then used to estimate the 'best' homography. Single value decomposition (using matlab svd function) is used to invert the matrix on the left (shown above for one point only) to generate an accurate homography estimation.

The estimator was tested with no noise and outliers to confirm correct operation. It was observed that the homography returned from Ransac was same as the original homography and the consensus set was the entire set of points (no zero elements)



The difference between the norm of the original homograpy and the estimated homography was plotted against varying levels of noise. Plots for error rejection of 3 and Ransac runs of 50 and error rejection of 1 and Ransac runs of 500 respectively are shown above. Outlier probabilities of 0.05 and 0.5 with no noise were also tested and almost no difference in the original homography and estimated homography was observed.

5 Estimating k-matrix

A number of images of the calibration grid are used to estimate the k-matrix. This is done to satisfy a rank requirement and ensure good conditioning of the estimation process.

A constraint equation for each image from $H = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix}$ can be obtained $\psi \begin{bmatrix} \phi_{11} \\ \phi_{12} \\ \phi_{13} \\ \phi_{22} \\ \phi_{23} \\ \phi_{33} \end{bmatrix} = 0$

Whereψ is the matrix composed of homography elements and $\phi = (K^{-1})^T K^{-1}$ is is the product of a lower triangular matrix,$(K^{-1})^T$ and an upper triangular matrix K^{-1}.Due to noise and error, there would

be no vector that make the right hand side of the above equation zero, therefore the norm that would minimize the matrix is found and the ϕ vector estimated using single value decomposition. This ϕ vector is used to construct the k-matrix using Cholesky Factorization. Scaling is used to scale the calibration grid and the camera dimensions to improve the conditioning of the calculations. The estimator was tested with no noise and outliers and the estimated K-matrix compared to the original K-matrix to confirm correct operation.

6 Optimizing K-matrix

The k-matrix is optimized by minimizing the variance of the back-projection errors of the consensus sets of calibration grid-Point image grid-point correspondences. The optimization method used is called the Levenberg-Marquardt algorithm. The algorithm works by converting the current set of 'best' positional parameters from the number of images being used to a set of 3x3 Perspectivites. The positions of the points in the image are predicted using the transformation matrices and the best k-matrix estimate at that moment. The actual image positions are then subtracted from the predicted image positions to build an error vector. The error vector from each image is squared and then summed. The parameters are then perturbed slightly and the error recalculated. The process stops when the error reaches a minimum.
The error function is defined as $E(p+\delta p) = E_0 + (\frac{\partial E}{\partial p})^T \delta p + \frac{1}{2}\delta p^T (\frac{\partial^2 E}{\partial p^2})\delta p$
WhereE_0 is the value of the cost function with the current set of parameters. δp is the set of small changes to the parameters that are added $(\frac{\partial E}{\partial p})^T$ is called the gradient or steepest descent. $(\frac{\partial^2 E}{\partial p^2})$ is a matrix of all the second derivatives of the error function with respect to the parameters and is called the Hessian. We have a minimum when this matrix is positive definite and $(\frac{\partial E}{\partial p})^T = 0$. The gradient function can then be expressed as$(\frac{\partial E}{\partial p})^T = e^T J$ and the Hessian H can be expressed as $H \approx J^T J$

6.1 Generating 3x3 perspectivities

The translation vector and rotation matrix are extracted from the data of the estimated homography and k-matrix. The rotation matrix is then used to find the axis of rotation and angle of rotation which is then shifted by 4pi to Generate a shifted angle - axis representation of the rotation (where the norm of Angle-Axis representation is the rotation angle) to make the transformation matrix ready for optimization. The rotation matrix is rebuilt using the Rodrigues Rotation formula and with the translation vector, it is used to generate the 3x3 perspectivity which is used to compute the predicted image points. Shown below is

a part of the ComputePridicted_uv function:

```
R = RodriguesRotation ( RotationAxis , RotationAngle );
%Build Transformation matrix using first 2 columns of rotation matrix
%and the translation vector
T = zeros(3);
T(:,1) = R(:,1);
T(:,2) = R(:,2);
T(:,3) = NTRANSLATION;
```

The error vector is then calculated in the ComputeImageErrors function (s is the number of non zero elements in the consensus set) :

```
%Compute error vector
for k = 1:s

    %Subtract the actual image points (extracted from the correspondence
    %matrix) from the predicted image points
    Error(((2*k)-1):(2*k),1) = ((Pridicted_uv(1:2,k)) - Actual_uv(1:2,k));

end
```

6.2 The Jacobian

The jacobian is constructed by first computing the predicted image points using ComputePridicted_uv function before any pertubation of the parameters. The parameters are then perturbed one by one and adjusted image points calculated. The initial and adjusted image points are then used to calculate the derivative vector:

```
%Find the number of non-zero points in the consensus set
k = nnz(NCONSENSUS);
for z = 1:k

    %Find the u related derivatives
    Derivative(((2*z)-1),1) = (AdjustedUVPoints(1,z) - InitialUVPoints(1,z))/Increment;
    %Find the v related derivatives
    Derivative((2*z),1) = (AdjustedUVPoints(2,z) - InitialUVPoints(2,z))/Increment;
```

```
end
```

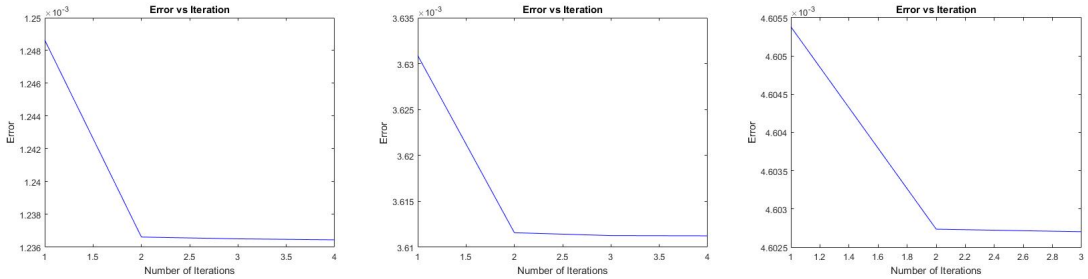
The derivative vectors are then ‘stacked’ to form the jacobian.

6.3 Optimization

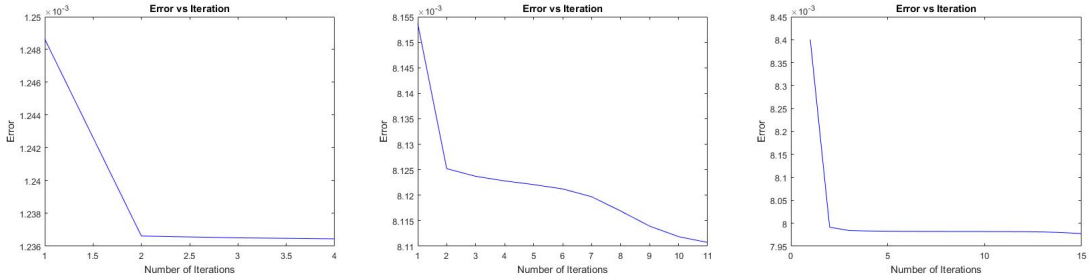
After constructing the jacobian, the gradient vector is computed. A weighting μ is used guide the change in parameters and a factor ν is used to ramp up μ very quickly if the cost goes up. This is used to smoothly switch between steepest desent and Newton steps to zero in on the minimum. After μ and ν are initialized the change in parameters is calculated. If the gradient $e^T J$ is small for the stopping criteria, convergence is achieved and optimization can end. Otherwise predicted change in the error is computed as $e^T J \delta p$. New test parameters are defined, the image location is perturbed and the error vector for this image computed. The change in error divided by predicted change in error (gain) is found, this is used to dampen μ and determine if the jacobian is accurate. If the gain is low, ($<$1/3) the jacobian is recalculated. If the error went up, μ is increased, the previous parameters are reset and the process repeated. If the error went down, the new parameters accepted and μ dampened. The error vector is then updated and the new gradient computed.

6.3.1 K-matrix optimization performance

Various levels of noise variance and outlier probabilities were used to test how quickly the optimization reached the stopping criteria of norm(Gradient)/ ProblemSize $<$ 0.0001.



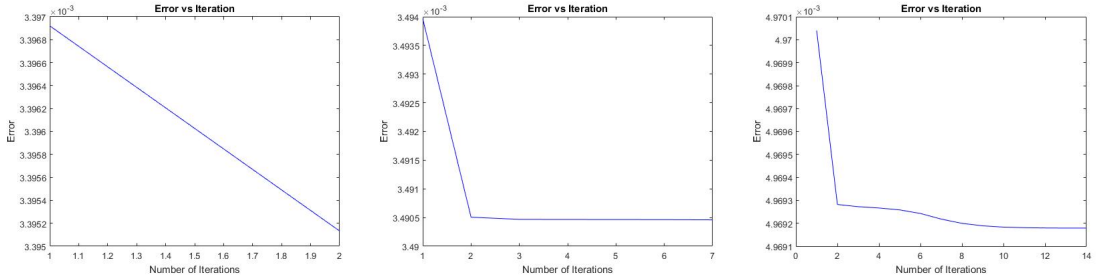
Constant noise variance of 0.5, changing outlier probability from 0.5 , 0.1 , to 0.05 respectively.



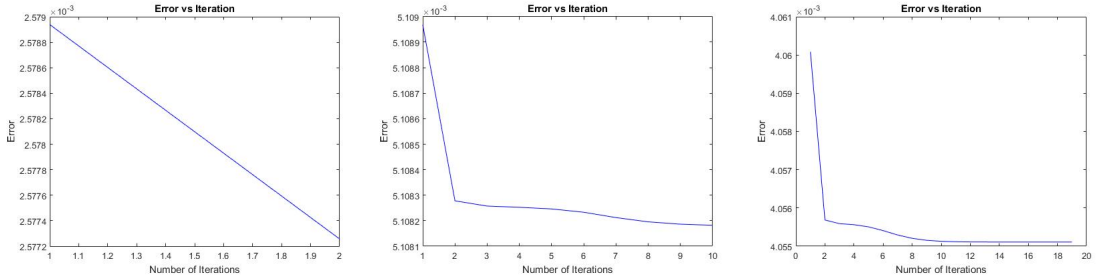
Constant outlier probability of 0.5, changing noise variance from 0.5 , 1 to 5 respectively.

6.3.2 Stopping criterion

Different stopping criterion were also used to observe behavior.



Constant noise and outlier probability. Changing stopping criteria norm(Gradient)/ ProblemSize $<$ s where s = 0.0001 ,0.00001, and 0.000001 respectively.



Constant noise and outlier probability. Changing stopping criteria norm(Gradient,inf)/ ProblemSize $<$ s where s = 0.0001 ,0.00001, and 0.000001 respectively. Infinity norm of the gradient vector would output the highest absolute value of the gradient vector, thus only the highest gradient element would be used for the stopping criteria instead of using a mean of all the gradient elements. It is observed that the stopping criteria using the infinity norm performs worse than the 2 norm.

7 Conclusion

Over the course of this project various algorithms were employed to reject outliers and then mitigate the effects of other errors to find the k-matrix. It is observed that the model estimation works well for small amounts of errors and gives reasonable results but perhaps better techniques could be explored such as instead of using a simple iterative method like RANSAC to using maximum likelihoods like KALMANSAC to make the model estimation more accurate and robust.