Software Technical Report

Reece Arnott

1st September 2010

Contents

1 Introduction

This is not a user manual, it is a technical report written to help developers who want to know how the software works so they can change it and make it better. With very few exceptions all references referred to in this report were available at the time of writing as free downloads on the internet. The biggest single one exception is the book, Multiple View Geometry in Computer Vision[8], which should be available at most University libraries and is considered the "bible" in the Computer Vision/Projective Geometry field.

This report is split into sections explaining the different parts of the program, starting with the properties file that contains the parameters loaded at start-up. Then the design of the calibration sheet is described and the algorithms used are described with references to the original papers for more details.

A brief outline of the steps in this program are as follows:

1. Find the ellipses in the images and the circles in the calibration image.
2. Match these found ellipses in the images with the circles in the calibration image. This step is the focus of a paper that is currently being written.
3. Initially assume the centres of the circles and the centre of the ellipses are the same point and use these point pair matches to estimate the camera parameters.
4. Adjust the camera parameters to get a better estimate including adjusting one half of the point pairs to be a better match based on the estimate of the camera parameters.
5. For each image also segment the image to identify the known calibration sheet. This step also calculates 2d edge information which is not currently used.
6. Once all the camera parameters have been estimated for all images the the image segmentation information of each image is used to create a rough silhouette outline that is the maximum volume of interest.
7. This is then converted to triangles and output to an STL file.

Other steps that may produce a better estimation of the object within this rough silhouette outline have been coded but have been commented out as they currently do not give a good representation of the object.

1. The 2d edges found in each image are used to produce rays in 3d space, limited to those that intersect the volume of interest, and 3d edge points estimated by triangulation of intersecting rays.
2. This point cloud is then used as input to a space carving algorithm that carves the space into a series of interlocking tetrahedrons.
3. Using the information of which camera originated the rays that were used for estimating each point, tetrahedrons are eliminated that are occluding a point in one of the camera views for which is known to be visible.

4. Further eliminate all tetrahedrons that do not have a pathway through other tetrahedrons to the ones that have as one of their vertices the bottom point - i.e. the lowest z coordinate.

5. The tetrahedrons left are assumed to describe the shape of the object in question and so the hull is converted to triangles and written to file.

Before getting into the actual program some terminology is explained in the next section and the assumptions built into the software explained in Section 1.2.

1.1 Vector and Matrix terminology

A large portion of the algorithms used for this software package are manipulations of vectors and matrices. When talking about vectors and matrices we also use the term scalar to mean a single number, whereas a vector is a 1 dimensional set of numbers, normally written in a column, with a matrix being a 2 dimensional set of numbers, normally written as a rectangular sequence in rows and columns. Just as a scalar can be defined as a 1-vector, so a n-vector can be defined as an nx1 matrix. When the individual elements of a vector or matrix are referenced they are normally referenced with subscript numbers based on the row and then the column index.

There are a number rules for simple algebra about how vectors and matrices can be combined with other vectors and matrices. For a good introduction, see the Linear Algebra set of You Tube videos from the Khan Academy [3]. It is assumed that the reader has an understanding of these simple interactions throughout this report. Some of the common symbols used in this report are:

1. Identity matrix: Each square matrix has its own identity matrix,I, where every element is 0 except for the diagonal elements which are 1 when the row and column indices are the same.

2. The zero matrix: All elements of a matrix are 0 and the matrix itself has the symbol of 0.

3. The transpose of a matrix: The rows and columns of a matrix are switched, this is represented by a superscript T e.g. A^T

4. The inverse of a matrix: Only square matrices have an inverse whereby the matrix multiplied by its inverse gives the identity matrix, represented by a superscript -1 i.e. $AA^{-1} = I$

5. The pseudo-inverse of a matrix: Some square matrices don't have an exact inverse but there are some processes to estimate it and the matrix so estimated is called a pseudo inverse, represented by a superscript + i.e. $AA^{+} \backsimeq I$

6. Vector cross product and dot product: represented by $\times$ and . respectively.

7. Vector Euclidean length: expansion of simple Pythagorean formula to n dimensions: $\sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}$ and is represented by double vertical lines around the vector e.g.$||a||$

1.2 Assumptions written into the software

For simplicity of user tuning there are a number of assumptions written into the software that cut down on the number of parameters that need to be set for good object recognition. Most of the these are to do with the design of the calibration sheet and camera placement with respect to the calibration sheet. These assumptions are:

- The calibration sheet is simply a set of black circles on a white background that are all the same size which provide a uniquely distinguishable pattern from all allowable viewing angles. For more information on the design of the calibration sheet see Section 3.
- It is assumed that the calibration sheet in an image is lying on a flat surface.
- It is assumed that the pixels in the image used as the source for the printed calibration sheet are square and that the image is scaled to fill the entire printable area of the page. This printable area is defined by the selected size of the page minus the vertical and horizontal margins specified. The user must indicate whether or not the aspect ratio was preserved when this scaling occurred.
- If the images are assumed to be taken with the same camera with the same zoom and focus settings and there are 3 or more images at the same resolution, no assumptions need to be made about the internal camera parameters. If there are only two images from the same camera at static zoom, focus and resolution then it is necessary to assume that there is no skew in the camera image axes, i.e. they are perpendicular, although this is not normally a problem except for poorly made cameras or when taking a picture of a picture. If the images are to be considered independently, then one further assumption, in addition to the no skew assumption, must initially be made, that is either that the focal point of the camera is in the exact center of the image or that the camera has exactly square pixels. The focal point may not be in the exact center of the image but it is a good first guess and will probably only be off by a few pixels. It may also be able to be detected and corrected if there is considerable radial distortion in the image. This adjustment is not currently implemented, nor is the user selection of whether all images are taken with the same camera. The assumption that the image has exactly square pixels is fine if there is EXIF metadata attached as EXIF standards [2]specifically state that the pixels are assumed to be square but this metadata is not mandatory so the assumption of the position of the focal point is made instead. Although there is code to deal with this hierarchy of assumptions the interface to allow the user to choose which set of assumptions to use has not been created and so all images are currently considered independently of each other.
- It is currently assumed that the camera does not have much radial distortion, the functionality to deal with fish eye lenses etc. is currently commented out as there has not been the chance to test it out.

- It must be assumed that all images contain the full calibration sheet and while some of it may be obscured by the object to be detected, the image is not a close up of only a portion of the calibration sheet.
- It is assumed that the object to be detected is within a volume of space defined by the dimensions of the calibration sheet. This volume of interest is the volume of space directly above the calibration sheet to a height equal to the larger of the width or height of the calibration sheet.
- An image taken at a lower viewing angle is assumed to be taken with a camera closer to the calibration sheet than one that is at a higher viewing angle such that in the extreme at the user provided lowest viewing angle the camera is vertically above the edge of the calibration sheet. This assumption is not normally going to matter except at either extreme angles or extreme distances where this assumption is used for limits on the size of the ellipses that need to be identified in the image.
- Images are assumed to be taken with the camera above the calibration sheet with the object to be detected sitting on top of the calibration sheet and unmoved, with respect to the calibration sheet, in all images. This means that some objects with indentations near the bottom of the object may not be detected properly. In the future it is hoped to provide an interface for the user to invert the object and to combine the two or more partially detected objects to form a better representation of the object but this is not currently implemented.
- The object to be detected is assumed to not have a pattern on it that can be confused with the calibration circles viewed at an angle.
- The object is assumed to be in contact with the calibration sheet.
- The GUI is designed for use on a screen with a resolution of least 640x480.

2 Configuration file

The main configuration file used is specified by running the java application at the command line with the preferences file as an argument. If the file does not exist or the value needed does not exist default values are used. If it is not specified the file to be used defaults to the users home under the .reprap folder and is called reprapscanning.properties. Th fields in this configuration file can be split into three sets of values: those that may be of use to average user or application, those that manipulate the internal algorithms, and those that are specifically for debugging.

2.1 Core Settings

Most of these settings are not complicated and are changed by the user in the normal running of the algorithm. There is no specialised knowledge needed to use them and it is expected that they will be manipulated at will by those using the program, mostly through the GUI but occasionally by modification of the preferences file.

SaveOnProgramWindowClose - A boolean setting to set what happens to properties in this file on the user pushing close button on the window, defaults to true which will rewrite the properties file.

SaveOnProgramCancel - A boolean setting to set what happens to properties in this file on the user pushing the cancel button of the program. Defaults to false which will not rewrite the properties file.

SaveOnProgramFinish - A boolean setting to set what happens to properties in this file on normal exiting of the program when the user presses the finish button. Defaults totrue which will rewrite the properties file.

AutomaticStep[1-6] - A series of boolean settings to allow the program to take the default settings in this file and apply them without the user having the option to change them at each step via the GUI. They all default to false but can be used by outside programs to link into this program and run it in a customised way. Steps 1 and 2 are for user input to change the default settings, so if they are set to automatic the settings from this file are used without any changes. Currently setting steps 3-5 as automatic does nothing as no user interaction is needed anyway. Eventually this could be used to read previously calculated results from file rather than calculate them. If step 6 is set to automatic then the program will exit automatically without the user having to press the Finish button.

BlankOutputFilenameOnLoad - A boolean setting to set what happens when the properties file is parsed. If there exists a default output file and this is set to true it is blanked. Note that the GUI will not continue if the step in which the output filename is selected is not set to automatic and the filename is blank. This means that the user must choose an output filename. Defaults to true.

ImageFileList[0-n] - This is the list of the images to be used to create the 3D model. Defaults to an empty list. Note that when the application is run this list is sanity checked to make sure each file exists, is a readable image file, and is not already in the list.

CalibrationPatternFileList[0-n] - This is the list of the image files that are possible calibration patterns to choose from. Defaults to an empty list. Note that when the application is run this list is sanity checked to make sure each file exists, is a readable image file, and is not already in the list.

CurrentCalibrationPatternIndexNumber - This references the CalibrationPatternFileList and is the currently selected calibration sheet. Defaults to 0 i.e. the top of the list.

PaperSizeNameList[0-n] - Taken together these PaperSize lists determine the selection of paper sizes available in the GUI drop down list. This defaults to 2 names: “A4” and “US Letter”.

PaperSizeWidthmmList[0-n] - Taken together these PaperSize lists determine the selection of paper sizes available in the GUI drop down list. This defaults to 2 settings only: the widths of A4 and US Letter in millimetres i.e. 210 and 215.9 respectively

PaperSizeHeightmmList[0-n] - Taken together these PaperSize lists determine the selection of paper sizes available in the GUI drop down list. This defaults to 2 settings only: the heights of A4 and US Letter in millimetres i.e. 297 and 279.4 respectively.

CurrentPaperSizeIndexNumber - This references the PaperSize list and is the currently selected paper size. Defaults to 0 i.e. the top of the list.

PaperOrientationIsPortrait - Boolean setting indicating whether the calibration sheet was printed in Portrait or not. If it was not printed in Portrait it is assumed to be printed in Landscape. Defaults to true.

PaperSizeIsCustom - This boolean setting indicates whether or not the default paper size selection is a Custom size or not. If it is then the list of paper sizes mentioned above is not used and the paper is assumed to be the size as specified by the below settings. Defaults to false.

PaperCustomSizeWidthmm - The size of the custom paper if selected, a minimum of 1mm, defaults to US Letter size i.e. 215.9mm. To be displayed and manipulated in the GUI to 1 decimal place only.

PaperCustomSizeHeightmm -The size of the custom paper if selected, a minimum of 1mm, defaults to US Letter size i.e. 279.4mm. To be displayed and manipulated in the GUI to 1 decimal place only.

PaperMarginHorizontalmm - The total of the left and right margins on the printed sheet, defaults to 0. A sanity check is run to make sure it is not larger than the currently selected paper size would allow. If it is it is reset to zero.

PaperMarginVerticalmm - The total of the top and bottom margins on the printed sheet, defaults to 0. A sanity check is run to make sure it is not larger than the currently selected paper size would allow. If it is it is reset to zero.

CalibrationSheetKeepAspectRatioWhenPrinted - It is assumed that when the calibration sheet was printed it was scaled to fit the size of printable area. If this boolean setting is set to true then this shrinking or enlarging was done whilst keeping the aspect ratio the same. Defaults to false.

OutputFileName - The filename of the file to save the output to. Defaults to output.stl in the users home folder unless the BlankOutputFilenameOnLoad field is set to true, in which case it is blanked. When the file is being saved this field is checked and if it is blank or invalid then no file is saved.

OutputObjectName - Metadata field to put in the stl file. Defaults to empty string.

2.2 More Advanced settings to adjust the functioning of the core algorithms

The settings for manipulating the algorithms used have names prefaced with AlgorithmSetting. It is not currently intended that they should have their values changed from within the GUI. They should not be changed by the average user and consist of the following:

AlgorithmSettingEdgeStrengthThreshold - The threshold value is a value which the edge strength (which is a value between 0 and 255) must be greater than for the edge to be used. 0 means all edges detected will be registered, 255 means none will. According to the original edge detection paper [7] 20 is a good threshold for a noise-free image and 30 for a noisy one. This same threshold value does "double-duty" as it is also used as the threshold value for determining whether a greyscale value for a pixel (in the range 0-255) is to be considered the same colour, either black or white, as another pixel which has previously been chosen as the blackest or whitest pixel in the local area. Defaults to 20.

AlgorithmSettingEllipseValidityThresholdPercentage - This validity threshold is what percentage of pixels in the rectangle formed around an ellipse that are outside the ellipse must be white, and what percentage of pixels in the ellipse must be black before it can be considered a valid ellipse. This test is done on the assumption that the ellipses we wish to find are black ellipses on white paper and cuts down on the number of inconsequential ellipses otherwise detected. This is a whole number between 1 and 100, defaults to 60.

AlgorithmSettingMaxBundleAdjustmentNumberOfIterations - The maximum number of times the bundle adjustment algorithm should try and adjust a first estimate of a set of values. Must be a natural number and defaults to 100.

AlgorithmSettingMaximumCameraAngleFromVerticalInDegrees - The maximum angle the calibration sheet will be viewed at. A whole number in the range 0-89. If it is less than zero it is reset to zero and if it is larger than 89 it is reset to 89. Defaults to 80.

AlgorithmSettingMinimumNumberofIntersectingRayPairsForPointEstimation - 3D points are estimated from triangulation based on multiple pairs of intersecting rays. This number must be a whole number. If it is greater than or equal to the number of images currently being processed then only those points that appear in all images as 2d edge points will be estimated. Defaults to 1. Currently not used.

AlgorithmSettingStepsAroundCircleCircumferenceForEllipseEstimationInBundleAdjustment - During the bundle adjustment process the center of the ellipse in the image is calculated using a deformation of the circle/ellipse on the calibration sheet using a number of points on the circumference of said ellipse. This number must be grater than or equal to 4 and defaults to 16. The more points, the more accurate the exact ellipse

shape and the more time it will take. Note that although the exact ellipse may change with more points, the ellipse center probably won't change much.

AlgorithmSettingVolumeSubDivision - The volume of interest, defined by the dimensions of the calibration sheet is sub-divided into a 3d grid of smaller volumes that define how small a feature is able to be detected. It is hoped that this could in the future be further refined to be a recursive sub-division. This whole number defines how fine that grid should be. Defaults to 128 meaning that the volume is divided into a grid of 128x128x128 sub-volumes. When used with a calibration sheet on an A4 or Letter sheet of paper this leads to voxels a little under 3mm on their largest side.

2.3 Specific Debugging settings

These debugging options are subject to change as developer needs change but currently these are the settings available:

DebugSaveOutputImagesFolder - Where to save images created by the setting of the other debug flags. If set to an empty string the images will not be saved. Defaults to an empty string.

DebugShowImageOverlay - Boolean setting that defaults to false. If set to true this will save a greyscale version of each input image with various found and deduced features overlaid on it such as the estimated calibration sheet circles and their centers, the centres of the ellipses found in the image, the edges of the calibration sheet and the calculated 3d edge points.

DebugShowImageSegmentation - Boolean setting that defaults to false. If set to true this will save a 4 colour image (black/white/light grey/dark grey) of each input image once it has been segmented into calibration sheet (white), edge (light grey), unknown(dark grey), or other (black).

DebugCalibrationSheetBarycentricEstimate - Boolean setting that defaults to false. If set to true this will save a greyscale image of the barycentric estimate of the calibration sheet for each input image along with white dots marking the centers of the calibration sheet the barycentric estimate is trying to match.

3 Calibration sheet design

The calibration sheet is assumed to be a simple white background with black circles on it. These circles are all the same size and should be placed in a semi-random manner around the sheet so that when viewed from any angle the configuration of the circles is unique. For best results this configuration should also be unique for configurations where some of these circles are obscured.

It is not known what the best calibration sheet is, but the worst would be a set of circles arranged around the center of the calibration sheet in a circle

as this will look the same from multiple directions. Another bad pattern is a regular grid pattern which will have at least 2 directions that could produce the same pattern. The example calibration sheet included with the software has worked for all test images so far but it is designed to be used in such a way that at least 6 of the 9 circles be visible in all images which limits the size of objects that can be processed.

Note that there when the estimation of the radial distortion matrix for large amounts of distortion, normally used when using a fish-eye lens, is implemented it will require at least 8 calibration points to be visible in the image for the estimation to work. If there are less points it will fail back to the single parameter estimation model which works best with low levels of distortion. This will therefore probably require a new calibration sheet with a larger number of circles so as to have at least visible in each image.

This seems to be a new way of thinking about calibration sheets and so a lot of research still needs to be done to figure out a way of defining a "best" calibration sheet.

4 Find Ellipses and Circles in the image

The ellipse detection mechanism used is specific for the task at hand, i.e. finding circles on a calibration sheet viewed at an angle, and relies on only two adjustable parameters, one threshold value for the initial edge detection, and one parameter describing the lowest viewing angle. The ellipse detection algorithm assumes the ellipses to be detected are near-circles viewed from an angle and at first glance needs 3 limiting parameters: maximum major semi-axis length, minimum major semi-axis length, and the lowest view angle. It is shown in Section 4.2 that the maximum and minimum major semi-axis lengths can be calculated from other information so only the lowest view angle needs to be defined. This is a value greater than or equal to 0 and less than 90 degrees. The view angle is measured from the normal to the calibration plane so 0 degrees is a view from directly above the calibration sheet. The algorithm broadly follows that laid down in [18] with the addition of some tests due to the use of the Absolute Difference Mask (ADM) algorithm[7] for edge detection which gives not only edge position and strength but also direction information. The algorithm is a combinatorial approach to the problem so the edge map used as input for the algorithm is filtered before being used for ellipse detection using suggestions given in [19]. A brief outline of the whole process is detailed below:

1. Optionally pre-process the image with a small Gaussian blur filter which is designed to smooth out random noise.

2. Use the ADM algorithm to create edge strength and direction maps. In the case of edges of ellipses the direction so produced will be the estimated direction of the tangent at that point. The edge direction will be one of 4 possible directions: vertical, horizontal, or one of the two diagonal directions.

3. Look in the 3x3 neighbourhood of the edges and if it isn't the maximum within this window, or under a threshold, suppress it. The threshold value used is the one loaded from the preferences file with the default value of

20 is taken from the suggested values from [7] of 20 for a nosie free image and 30 for a noisy image.

4. Remove any additional spurious edge points. This includes: points at the edge of the image that were detected as edges simply due to being at the edge of the image and excessively connected edge points which are defined as in i.e. edges that have 3 or more edge points within a circle of radius $\sqrt{2}$. There is another possible set of edge points that are suggested by[19] that can be deleted which are those points that are completely isolated i.e. there are no other edge points within the 3x3 mask centred on the edge point. However, while this is purported to work for the algorithm detailed in the original paper, the actual ellipse detection algorithm used needs this additional information.

5. Put the remaining edges and tangent directions into 1D arrays and, if the option is selected, to have a 2D sparse array with index pointers to the 1D array for the edges. This optional memory structure is used later for finding a subsets of the 1D arrays based on a bounding rectangle around a point.

6. The minimum major semi-axis length is compared to the resolution of the edge finding algorithm to make sure it is greater and if it isn't, it is reset to this value. The resolution of the edge finding algorithm refers to the idea that a point on the transition in an image between, for example, black and white, may be found as belonging to an edge on both sides of the actual dividing line, one where the edge is detected as a change from white to black and another where it is detected as a change from black to white. The resolution is the minimum distance away two edge pixels must be before they can be known to not be referring to the same point. In this case the ADM algorithm has a resolution of 5 pixels due to its use of the 5x5 neighbourhood in finding edge information.

7. The maximum major semi-axis is similarly compared to the length of the diagonal of the image and it is reset to that if the diagonal length is smaller.

8. Find ellipses using the edge arrays created. For each of the edge pixels (i) from 1 to N-1 do the following:

   (a) Go through each of the edge pixels (j) from i+1 to N, optionally limited to those within a bounding rectangle based on the maximum long axis length, and test the (i,j) pair to see if it is possible they are on opposite sides of an ellipse. These tests are:

      i. The tangent directions of the two edges are the same.
      ii. The distance between the two edge pixels is between the limiting minimum and maximum long axis length.
      iii. The centre point on the line between the two edge points is not within any of the ellipses detected so far.

   (b) If these tests are passed the pair of edges are assumed to be on opposite sides of the same ellipse long axis and so from this assumption the major semi-axis length, now labelled a, is calculated as well as

the centre point used in the last comparison is used as the center point for the ellipse and the angle of the long axis compared to the x-axis, the orientation angle τ using equations 1-4 in[18].

(c) The accumulator is reset and the minimum length of the minor semi-axis, b, is calculated using the calculated a and the lowest view angle. From [10] we have a formula that can be used to relate the eccentricity of an ellipse to the view angle on the assumption that the ellipse is actually a circle: $e = \sin(\theta)$. Given the standard definition of the eccentricity: $e^2 = \frac{a^2-b^2}{a^2} = 1 - \frac{b^2}{a^2}$ and the Pythagorean trigonometric identity $\cos^2(\theta) + \sin^2(\theta) = 1$ we can restate this as $\cos(\theta) = \frac{b}{a}$ or, rearranging to solve for b, $b = a\cos(\theta)$. If this is smaller than the resolution of the edge finding algorithm, then b is reset to the resolution.

(d) For each edge pixel (k) from 1 to N, optionally limited to those within a bounding rectangle based on the calculated center of the ellipse and a, if the edge pixel is not i or j, and the distance between the calculated center of the ellipse and edge k is less than or equal to a, calculate the minor semi-axis length b using equations 5 and 6 in [18]and increment the accumulator.

(e) Find the element in the accumulator with the maximal count. If the edges used to create this b value come from all 8 octants of the ellipse this value is confirmed as the new semi-minor axis length b. Add the ellipse to the output list of detected ellipses and take out any edges from the array that are within this ellipse. Exit the j loop prematurely on the assumption that an edge is part of a maximum of one ellipse.

4.1 Circle detection in the calibration sheet

A perfect circle is simply an ellipse with both a and b of the same length. Given the pixelation of the calibration image a and b will not be exactly the same length but this is taken into account by setting the lowest angle input into the ellipse detection algorithm to be 10 degrees meaning that allowable minor semi-axis value is 98.5%-100% of the detected major semi-axis value.

As no information about the size of the circles in the calibration sheet is known, although it is assumed that all the circles are the same size, the minimum major semi-axis length is initially set to 0 and the maximum the diagonal length of the calibration sheet. The optional bounding rectangle technique is not used as the calculated bounding rectangle would be a significant portion of the image and the sparse nature of the edge points means that significant amounts of time would be taken up in simply finding the next edge point to process.

Once one circle is found with these parameters the minimum and maximum major semi-axis lengths are reset based on the parameters of this circle.

When all the near-circular objects have been found the radius is estimated as the average of the minor and major semi-axes of the detected ellipses in the calibration sheet.

The size and margins of the printed calibration sheet are then taken into account to give the size and shape of the printed circles. In the case where the

printed calibration sheet does not have the aspect ratio of the calibration sheet image preserved, this produces an axis aligned ellipse rather than a circle.

4.2 Calculation of the maximum and minimum semi-axis lengths for ellipse detection

It is assumed that as the viewing angle gets further from the vertical then the camera gets closer to the calibration sheet so that in the limit where we are at the lowest angle the camera is directly above the edge of the calibration sheet. Therefore if we construct a right angle triangle with one of the sides being the length of the diagonal of the image and the angle opposite it being the lowest viewing angle θ then we have constructed with the lengths of the two unknown sides being the closest and furthest distances to any possible ellipses at the lowest view angle. Using the standard trigonometric definition $\sin(\theta) = \frac{opposite}{hypotenuse}$ we can find the length of the hypotenuse, and from that and the standard Pythagoras equation we can find the length of the other side. Using these two lengths we now have a ratio of the maximum to minimum semi-axis lengths for a circle viewed at the the lowest viewing angle, given the above assumptions. Multiplying this by the ratio of the printed calibration sheet calculated semi-axis lengths if needed, due to the lack of preservation of the aspect ratio, gives a final ratio for the printed ellipse. Now if we can find either a maximum or minimum semi-axis length we can calculate the other.

It is further assumed that the image contains all of the calibration sheet so that we can calculate the maximal size of the ellipses which is if the image contains nothing but the calibration sheet. For this to occur the camera must be directly above the calibration sheet center and the circles will be seen as ellipses with the same axis ratio as the printed calibration sheet representation calculated above. To get the maximum major semi-axis length therefore we just need the semi axis lengths of the printed ellipses in the calibration sheet, which were calculated above, and the ratio of width to height in the calibration sheet and image, fundamental properties of the image.

5 Matching points

The calibration sheet has a series of n identical circles on it which, when imaged from an angle, become ellipses, and the ellipse finding algorithm finds m ellipses in the image. It is assumed that the ellipse finding algorithm correctly matches a subset of the circles with no extraneous ellipses. To a first approximation the centre of the imaged ellipse is also the center of the original calibration circle so we then have two sets of points that we wish to pair up but no way of knowing how to do so.

If the camera position, and therefore perspective transformation warping of the calibration sheet, were known we could take a subset of point pairs and calculate where the image points corresponding to the other calibration points were based on their relationship to each other from the known calibration sheet. With no information on how these points pair up we can use a combinatorial approach and try every combination and find the one with least error.

However, the reason we are doing this is to find the camera position in the first place. To solve this “chicken and egg” situation we use a barycentric

coordinate transform as a first approximation to the perspective transform.

5.1 Barycentric Coordinate Transform

Standard barycentric coordinates are based on the relative distance from one point to each of a set of two or three points. The number of anchor points define the number of dimensions the point in question is restricted to. With two anchor points the point we are interested in can only lie on a line passing through the anchor points, with three it lies on a plane constructed to include the three anchor points. As these barycentric coordinates give a relative position of a point with respect to the anchor points, they can be used to transform a point to another coordinate system so long as the mapping between the anchor points is known. While this transformation is not exactly the same as the correct projective transformation would give, it is a good estimate in the local neighbourhood.

The easiest form of barycentric coordinates to visualise are the 2 coordinates needed to find a point on a line segment between two endpoints. Any point on the line can be referred to by how close it is to the two points, with the coordinates adding up to one. If the point is exactly at the end of the line, the coordinate relating to that end-point will be 1 and the other 0. If it is at the half way point between the two, both coordinates will be 0.5. If it is closer to one end than the other, the coordinate for the end it is closer to will be more than the other. This can be extended by the use of negative numbers to refer to a point on the line past the endpoints of this line segment and it can also be further extended to form the standard triangular barycentric coordinates with 3 coordinates that relate a point in a plane to its relative placement to the 3 corner points of a triangle.

5.2 Combination versus Permutation

When counting the number of different subsets of r unique elements in a set of n unique elements the total number of subsets is different depending on whether the order of elements in the subsets matter or not. If the order of elements is of no importance then it is called a combination of elements, and a permutation if order is important. These are symbolised by $C_r^n$and $P_r^n$and are defined as follows:

$P_r^n = \frac{n!}{(n-r)!}$

$C_r^n = \frac{n!}{r!(n-r)!}$

where $n!$ is the standard factorial expression i.e. for all positive integers $n! = n * (n-1) * (n-2) * ... * 2 * 1$ with 0! being defined as 1.

In BigO order notation both these are close to $O(n^r)$ for $r \ll n$.

Programmatically, both combinations and permutations can be done using nested for loops, with the variable affected being an index into the array of n elements, although for permutations there also needs to be a check for the uniqueness of each element. The nested for loops for combinations are simply described by the following pseudo-code:

```
for item1 = 1 to (n-r+1) do
    for item2 = item1+1 to (n-r+2) do
    ...
```

```
for item_r = item_{r-1}+1 to n do
    ...
```

The code for permutations are more complex but can be expressed as below:

```
for item_1 = 1 to n do
    for item_2 = 1 to n do
        skip = item_1==item_2
        if !skip for item_3= 1 to n do
            ...
            skip = item_{r-1}== item_1 or item_{r-1}== item_2 or ... item_{r-1}== item_{r-2}
            if !skip for item_m = 1 to n do

              skip = item_r== item_1 or item_r== item_2 or ... item_r== item_{r-1}
              if !skip
                ...
```

5.3 Outline of method

Given that if the anchor points for the barycentric transformation are spread further apart they are more representative of the global perspective transformation we can cut down the work the algorithm needs to do if we choose the anchor points intelligently. Rather than using every combination of three image points, we find the single best combination of three image points defined by those that form a triangle with the largest area and then match those three image points to every permutation of the calibration points to find the best match. One formulation of the area of a triangle is $\frac{1}{2}||AB \times AC||$ where AB and AC are the vectors between the two pairs of points forming the corners of the triangle. As we don't actually need to know the exact area, we just want to rank them, we simply use the magnitude of the cross product of these two vectors to avoid the unnecessary division by two and find the three points that have the largest magnitude of their cross product.

Once the anchor image points have been found for each of the permutations of the point pairs formed by pairing these points up with 3 of the calibration sheet points we do the following:

1. For each calibration point that has not yet been paired up, label this point the source point and find the 3 closest calibration points that have been paired up.

2. Use these 3 points to give the source point barycentric coordinates. If the source point is collinear with two of these points, we can simplify the problem and just use these two points to give 2 barycentric coordinates rather than 3 in the more general case.

3. Use these barycentric coordinates and the image points that are the pairs to the 3 calibration points to convert the source barycentric coordinates to an image point.

4. Measure the Pythagorean distance from this point to the nearest unpaired image point and call this the distance error for this point pair match

5. From the point pair matches generated by steps 1-4, find the point pair match with the smallest distance error and add this to the array of point pairs.

6. Repeat steps 1-5 m-3 times to find all point pairs and cumulative error. Note that if the cumulative error is greater than that of the current contender, the loop can be exited prematurely to go onto the next permutation.

7. Once all the points have been matched, find the 3 closest calibration points to each of the anchor points and use these to find the respective image point and add the distance error for each of these three matches to the cumulative error. Note that this doesn't need to be done if the cumulative error is already greater than the current contender.

8. If the cumulative error for these point pair matches is less than that for the current contender, this set becomes the new "correct" point pair match.

The final correct combination is then the combination that has the lowest cumulative distance error. As currently proposed $m \leq n$ so the worst case is $O(n^5)$. Obviously then, if n is not relatively small this algorithm becomes infeasible but at a minimum we only need 4 point pairs for our calculations, see section 6.2, so we need only enough points that four will always be visible. The current calibration sheet has nine points and completes these combinatorial calculations in a few tens of microseconds on a modern (2009) dual core machine. A paper is currently being written that is mainly an expanded form of this explanation.

6 Camera Calibration from point pair matches on planar calibration sheet

Note that the methodology stated here is in large part a rewording and more expansive explanation of the method outlined by Zhang in [20]. Also note that there were mistakes in some of the formulae in the original published paper that have been fixed in the on-line version, available as a Microsoft Technical Report from the Microsoft website.

6.1 Camera Calibration Overview

The goal of camera calibration is to be able to produce a 3x4 matrix (P) that transforms a real world 3d homogeneous point into a 2d homogeneous image point and conversely can be used to turn a 2d image point into a ray in the real world. i.e.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = P \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Once this matrix P has been calculated for each image we can then elicit information about the shape of the object we are looking at by taking a real world point and seeing what happens when we transform it into the image coordinates of each image.

For our purposes the camera calibration matrix P can be broken down into a number of sub-matrices in the following way:

$P = K[R|t]Z$
where:

K — A 3x3 matrix of the form $K = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$ which embodies the internal parameters of the camera. The Zhang method estimates these parameters from constraints that can be placed on a geometric entity called the Image of the Absolute Conic (ω) which is then decomposed to find the the camera matrix as $\omega = (KK^T)^{-1}$. If the camera matrix can be assumed to be the same for multiple images i.e. the images were taken with the same camera with the same zoom, lens settings, and resolution then the camera matrix can be reliably estimated from 3 such images with 4 known point matches on the calibration sheet in each image. This is because each image needs four point matches on the calibration plane to be able to estimate a planar homography that takes the calibration plane in the real world and maps it to the image plane. Each one of these homographies places two constraints on the Image of the Absolute Conic (IAC) and so if we have 3 such homographies the camera matrix can be estimated. If there are only two homographies or the images are assumed to be independent of each other we need to make some assumptions about the nature of the camera. These are discussed in Section 6.4.

[R|t] — A 3x4 matrix which can be further broken down into a 3x3 Rotation matrix and a 3x1 translation vector. These matrices can be estimated from a planar homography once the camera matrix K has been found. See Section 6.5.

Z — A 4x4 matrix which needs to be added as the Zhang method for estimating the above matrices sets an arbitrary z scale. The real world x and y axes are defined as being based on the x and y directions of the calibration sheet with the scale being in real world millimetres based in turn on scaling of the pixels of the calibration sheet using the provided paper size and margins on the assumption that pixels of the provided calibration sheet image are square. To provide for the case where this is not true or there are rounding errors in the calculations, the real world z scale is defined in its most general sense to be the average of the x and y scales. A procedure is described to calculate the z scale factor in order to offset the arbitrary scaling based on the lengths of transformed unit vectors. This method leaves only the ambiguity of the sign of the z scale which is solved by assuming that the camera is always "above" the calibration sheet. For the purposes of the calculations of K and [R|t], Z is assumed to be the identity matrix. Once the z scale factor is found the matrix Z is then modified to be of the form $Z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ where s is the internal z scaling factor. See Section 6.6.

Note that we also have the complication that the image points so estimated will not in general match up with the actual image due to the effects of lens distortion. To get around this we also need a model to estimate the effect of lens distortion on the image and negate it. There are two kinds of lens distortion, radial and tangential, but as mentioned in [17], from experience it has been shown that only radial distortion needs to be considered. There are two models that are used in this software, one of which only works with high radial distortion and one that only works with low radial distortion. The first is to model distortion using a 3x3 matrix (D) calculated after the other matrices based on the methodology used in [9] to re-envision radial distortion as a Fundamental Matrix of distortion. In the case where there are less than the minimum number of point pair matches to estimate this matrix or the resultant matrix fails a sanity check on the position of the centre of the distortion, the second model is used. As the radial distortion is modelled by an infinite series, a distortion function is estimated based on a truncated form of this series with one coefficient, k_1,that needs to be estimated, similar to the original one suggested in [17]. Once the distortion is estimated the image is pre-processed to take out the radial distortion before transformations between real world and image coordinates are done for detection of the object. The distortion is estimated for each image separately as the distortion can be significantly different between images, even on the same camera, due to differences in focal length and especially zoom [14]. This means the user can just "point and shoot" and not have to worry about keeping the focus and zoom the same between shots. See Section 6.7 for more details.

Once all these parameters have been estimated individually they are further refined collectively by a bundle adjustment algorithm, in this case the Levenberg-Marquardt algorithm. See Section 6.8.

6.2 Estimating Planar Homography (H)

In this context we are wanting to take the calibration plane and transform it for each image plane. This planar homography matrix, H, will then successfully map all points lying on the calibration sheet onto the image. Note that as we have defined the real world coordinate system by the placement of the calibration sheet this will map all real world points with z=0 to the image but it does not say anything about any other real world points. There are standard algorithms for doing this and the one chosen was the normalised Direct Linear Transform (DLT) algorithm. A short outline of this algorithm is given here but this is a standard algorithm and is explained in detail in other sources, eg. Chapter 4 of [8]. This algorithm requires 4 or more point pair matches to give a unique solution.

1. Create and apply normalising matrices to the two sets of points such that the centroid of the set of points is at the origin and the average distance from the origin is $\sqrt{2}$.

2. Each point pair gives 2 rows, each with 9 entries, in a matrix A where Ah=0 with h being the 9 vector corresponding to the 9 entries in the 3x3 normalised planar homography matrix H'.

3. Use SVD to solve and find h. See Section B

4. From the 9 x 1 vector h, construct the 3 x 3 matrix H'.

5. Use the normalising matrices from step 1 to de-normalise H' to get H.

This may end up with an over-constrained homography, in which case the adaptive RANSAC method is used to find an unknown proportion of outliers and then and create a better homography with fewer point pairs. Again this is a standard algorithm and is explained in detail in Chapter 4 of [8] with only an outline given here. The threshold for the determination of whether a point is an outlier or inlier is found as part of the normalised DLT algorithm as the maximum transfer error i.e. how far the image point of the point pair is from the image point calculated from the calibration point and the homography. A diminishing percentage of that value, starting at 100%, down to 0%, in steps of 1%, is used as the distance threshold for RANSAC calculations so that if the homography is still found to be over-constrained the RANSAC algorithm is run again with a smaller distance threshold until the homography is found to not be over-constrained or less than 4 point pairs are defined as inliers. The steps in the algorithm are:

- Initialise the sample count to be 0, $N = \infty$, current assumed inliers are 0, and p, the probability we have chosen at least one sample with one outlier in it, to be 0.99.
- While $N >$ *samplecount* repeat
  - Choose a random sample of 4 point pairs where no 3 of them are collinear.
  - Calculate the homography using the normalised DLT algorithm for these four points.
  - Using all the point pairs, compare the image point to the calculated image point using the calibration point of the pair and the calculated homography. Classify the point pair as inlier or outlier based on the distance between these points using the distance threshold passed into the algorithm as the dividing point.
  - Calculate the variance of the inliers.
  - We may need update the value of N using the value of p and the current estimated number of outliers. This is only done if the number of inliers is greater than the current assumed inliers. Note that this will only tune N downwards as more and more inliers are found. N is the number of samples we must pick to have a probability of p (currently set to 0.99) of picking one in which all of the samples are inliers.
  - If the currently calculated homography produces either: more inliers, or the same number of inliers but with less variance, or this is the first time through the loop, set the current best set of inliers to be the current set of inliers.
  - Increment the sample count by 1.
- Use the best set of inlier point pairs to create a homography using the normalised DLT algorithm.

As a final step in both these algorithms, use the bundle adjustment Levenberg-Marquardt algorithm to get the maximum likelihood estimation using all the point pairs. See Section 6.8.

6.3 Explanation of the Image of the Absolute Conic (ω)

The concept of the Image of the Absolute Conic (IAC) seems rather abstract but is quite useful so it would be good to give a short explanation of where it comes from. This involves a number of ideas that may seem unrelated at first but they do come together.

1. We can define parallel lines in 2D as lines that meet at "the line at infinity" and define this line at infinity in 2D homogeneous coordinates to be $(x, y, 0)^T$

2. In 2D Euclidean geometry a circle and an ellipse have different properties, the most relevant being that two circles will in general intersect in two points, whereas two ellipses intersect at four points. In both cases we are intersecting two second degree curves or solving two quadratic equations and should therefore get four solutions. The difference is that in the case of circles two of these points are complex points.

3. The equation of a circle in homogeneous coordinates $(x, y, w)^T$ is: $(x - aw)^2 + (y - bw)^2 = r^2w^2$ and we can see that the points $(1, -i, 0)^T$ and $(1, +i, 0)^T$ lie on every circle. These are the complex points that are two of the intersection points of two circles and they lie on the line at infinity. These points are called circular points.

4. Projective geometry is a more generalised form of geometry than the standard Euclidean geometry we are used to from high school and a projective transformation does not necessarily preserve angles, line length, ratio of lengths, or anything other than the straightness of lines but in the 2D case it can be limited to be Euclidean if we single out a line at infinity and subsequently two circular points on this line.

5. We can generalise this result to 3D in the following way: two spheres intersect in a circle but two general ellipsoids intersect in a general fourth degree curve. We can see that in homogeneous coordinates (x,y,z,t) all spheres intersect the plane at infinity in a curve with the equation $x^2 + y^2 + z^2 = 0; t = 0$. This is a second degree curve, called a conic, lying on the plane at infinity and consisting only of complex points. This is called the absolute conic. A 3D Euclidean space is defined by singling out a plane at infinity and specifying a particular conic lying on this plane as the absolute conic.

6. For a camera not on this plane at infinity, the plane at infinity in the world maps one-to-one onto the image plane as each point on the image plane maps to a line in the world that intersects the plane at infinity at one and only one point. The absolute conic, being on the plane at infinity must also map point-for-point to the image plane and the conic so mapped is the Image of the Absolute Conic (IAC).

7. If the IAC is known then the Euclidean structure of a scene can in principal be found. In the case of a single camera this is somewhat limited as a point in the image being reconstructed can only be reconstructed as a ray in the real world but, in the case of multiple camera angles, real world points can be estimated by points of intersections of rays from different cameras etc.

The IAC is a symmetrical 3x3 matrix with 6 unique entries so it requires 5 constraints to estimate it up to scale. The main ones used in this implementation and emphasised by Zhang are 2 constraints from each planar homography. Therefore if we have 3 or more images known to be taken from the same camera with the same settings then the IAC can be found from these alone. Due to the fact that the IAC is also defined as $\omega = (KK^T)^{-1}$there are a number of constraints that can be used based on assumptions about the camera matrix parameters as well:

1. 1 constraint comes from the assumption that there is zero skew i.e. image axes are perpendicular and so pixels are not a general parallelogram but are restricted to be rectangular. This is normally a valid assumption but is only made if there are less than 3 images to be used to estimate the IAC.

2. 2 constraints come from the assumption that the principal point is at the image coordinate origin. This assumption is only used when each image is taken individually and the image coordinate origin is pre-processed to be at the centre of the image as this is a good first guess but will normally be out by a few pixels. If the radial distortion is large enough the center of distortion can be estimated and this used as the coordinate origin in subsequent calculations. Note that due to the specific implementation of the derivation of the IAC, namely expressing the constraints as linear combinations of the IAC entries that sum to zero, the principal point cannot be assumed to be anywhere but the origin so if it is known to be at a specific point the coordinate frame needs to be changed to make this point the image origin before the IAC is calculated.

3. 2 constraints come from the assumption that the pixels of the image are square i.e. the pixel aspect ratio is 1:1 and there is zero skew. Note that due to the way the IAC and camera matrix are put together you cannot easily assume the pixel aspect ratio is 1:1 and *not* assume zero skew. There is optional metadata for images stored in the EXIF format that can confirm that this is a valid assumption but, as it is optional, its absence cannot be taken to infer that this assumption is invalid. Currently these constraints are only used when multiple planar homography constraints are not all independent leading to 3 independent constraints although it is possible in future the EXIF information may be read and this constraint added if appropriate.

As the current implementation treats each image independently the first two assumptions have to be made, although there is also code to implement this hierarchy of constraints if need be.

6.4 Estimating Camera Matrix (K) given H via ω

The camera matrix K is of the form $K = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$. $u_0$and $v_0$are the coordinates in the image coordinate frame of the principal axis. α,β, and γ embody the shape of pixels and the focal distance, in pixel coordinates, and can be further decomposed as $\alpha = f$, $\beta = fa$,$\gamma = fs$ with $f = s_u$, the focal length in u-pixels, $a = \frac{s_v}{s_u \cos(\theta)}$ the aspect ratio v:u pixels $s = -tan(\theta)$ the skew factor as suggested in [15].

Normally $\gamma = 0$ as most cameras either have perpendicular axes or are processed so that the digital image obtained has perpendicular axes, but a non-zero value can also be interpreted as a result of taking an image of an image. The Zhang method estimates these parameters from constraints that can be placed on the Image of the Absolute Conic (ω) which is then decomposed directly into the five camera matrix components and a scaling factor via formulae devised by Zhang.

The estimation of the camera matrix is then replaced with the problem of estimating the IAC. This can be done, up to scale, by creating a matrix A where each row of A is a constraint on the values of the six components of ω and solving to find these components. i.e.

$B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix}$ and $A \begin{bmatrix} B_{11} \\ B_{12} \\ B_{22} \\ B_{13} \\ B_{23} \\ B_{33} \end{bmatrix} = 0$ where $B = \lambda\omega$, with λ being a scaling factor.

Because there is a further constraint on ω that it be a symmetric positive definite matrix, as are all matrices formed by one matrix multiplied by its own transpose, we test the matrices B and -B to see if either of them is symmetric positive definite. If neither of them is symmetric positive definite it is normally an indication that the homography may be over constrained and so it can be calculated again using less points with the adaptive RANSAC method. If this has already been done there is catch all code that will do a polar decomposition: $B = UP$ where P is the symmetric positive definite part of the matrix and U is the unitary part i.e. $U^TU = I$. It is assumed at this point the only reason this will happen is because of rounding errors so we can throw away the U portion and set $B = P$. This can be calculated from a standard SVD decomposition of B. See Appendix B.

The matrix B so estimated can then be directly decomposed into the components of K and the scaling factor λ using the formulae given in the Zhang paper and reproduced below:

$v_0 = (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2)$
$\lambda = B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11}$
$\alpha = \sqrt{\lambda/B_{11}}$
$\beta = \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)}$
$\gamma = -B_{12}\alpha^2\beta/\lambda$
$u_0 = \gamma v_0/\beta - B_{13}\alpha^2/\lambda$

An overview of how these can be derived is in Appendix A.

Each planar homography leads to two independent rows of A, equation 8 of [20]. Therefore if the only constraints are to be from the planar homographies we need 3 non-degenerate images to find the IAC and therefore the camera parameters. If we only have two images or we wish to treat each image independently then we need to define other constraints so they can easily be added into the matrix A. As the components of the IAC are related to the components of the camera matrix we can use assumptions about some of them redfined as linear combinations of the components of the IAC that sum to zero:

1. The constraint that there is no skew i.e. $\gamma = 0$ leads to the constraint that $\omega_2 = 0$ which can be done by adding a row to A: (0,1,0,0,0,0).

2. The constraint that the principal point is at the image coordinate origin i.e. $u_0 = 0$,$v_0 = 0$ leads to $\omega_4 = 0$and $w_5 = 0$ which are accounted for by adding two rows to A: (0,0,0,1,0,0) and (0,0,0,0,1,0).

3. The constraint that we have square pixels i.e. $\gamma = 0, \alpha = \beta$ leads to the constraints that $\omega_2 = 0$ and $\omega_1 - \omega_3 = 0$ which are two rows to A: (0,1,0,0,0,0) as with the no skew constraint and (1,0,-1,0,0,0).

With combinations of these constraints we will then have a matrix with a rank of at least 5 and it is possible to estimate ω by using SVD, see Appendix B, and therefore the elements of the camera matrix. Currently the images are being considered independently so the no skew and principal point constraints are used. In the future case of two independent images we use only the first assumption of no skew and, for three or more, we don't need to make any such assumptions. The square pixel constraint is not currently to be used unless the matrix A is of rank 3 which would only happen in the case where there are multiple images but the constraints they produce are not all independent.

6.5 Estimating Rotation and Translation Matrix (R|t) given K and H

Once the camera matrix K has been estimated it can be used in conjunction with the planar homography to find a good first estimate of the rotation and translation matrix. Again Zhang provides a simple set of formulae to find this:
$r_1 = \lambda K^{-1} h_1$
$r_2 = \lambda K^{-1} h_2$
$r_3 = r_1 \times r_2$
$t = \lambda K^{-1} h_3$

where r_1, r_2, r_3 are the 3 columns of the R matrix, $h_1, h_2, h_3$are the 3 columns of the planar homography matrix H, and λ is a scaling factor. In the case of perfect estimation of K, $\lambda = 1/||K^{-1}h_1|| = 1/||K^{-1}h_2||$ but in the general case the two will give slightly different answers. If you pick the first, you over-fit to the image x axis and tend to be out in the y direction. If you pick the second, you over-fit in the image y axis and tend to be out in the x direction. In this software implementation this has been solved by setting it to the average of the two i.e. $\lambda = \frac{2}{||K^{-1}h_1|| + ||K^{-1}h_2||}$ although in testing it was found that the two values are only different by fractions of a percent.

These formulae can be derived from the definition of P as $P = K[R|t]Z$ where $Z = I$ and can therefore be ignored, and the knowledge that the first,

second, and fourth columns of P are the first, second, and third columns of H, leaving only the third column of P undefined, hence the arbitrary scaling factor λ.

Note that the rotation matrix so estimated will not in general be an actual rotation matrix but the closest fit can be found using SVD, see Appendix B.

6.6 Estimating Z scale factor (Z)

The Zhang method will find a camera matrix plus rotation and translation matrix but the z scaling factor is rather arbitrary as we have defined z=0 to be the calibration plane and therefore have no data on z. We do however have knowledge of the scaling done in x and y so we can define our first guess at the z scale in terms of x and y. The x and y scales of the calibration sheet and therefore the internal coordinates have been changed to be in millimetres to reflect real world measurements and should therefore be the same and it makes sense to define the real world z scale similarly. This will serve in most cases but for additional generality and robustness we will define the z scale to be the average of the x and y scales.

By so doing we have therefore defined $||u_{zr}|| = \frac{||u_{xr}||+||u_{yr}||}{2}$ with u_{xr}, u_{yr}, u_{zr} being x,y, and z unit vectors in the real world frame. We also want $||u_{zi}|| = \frac{||u_{xi}||+||u_{yi}||}{2}$ where u_{xi}, u_{yi}, u_{zi} are x,y, and z unit vectors in the image homogeneous coordinate frame . To get the correct scale conversion factor therefore we use the formula $s = \frac{||u_{xi}||+||u_{yi}||}{2||u_{zi}||}$. To get the lengths of these vectors we transform the real world homogeneous origin $(0,0,0,1)^T$to image homogeneous coordinates and find the unit vectors distance between it and the real world points $(1,0,0,1)^T, (0,1,0,1)^T$, and $(0,0,1,1)^T$ also transformed to image homogeneous coordinates. From this we find s but with an ambiguity of sign. To take out this last ambiguity we must assume that all images are taken with a camera in the positive real world z direction i.e. the camera is above the calibration sheet. If we then compare the length of the vector connecting the inhomogeneous projected image points of the origin and $(0,0,1,1)^T$ with the origin and $(0,0,-1,1)^T$ we can easily see whether the sign is correct. For a camera in the positive z direction, the vector from the positive z point to the origin should be larger than the vector from the negative z point to the origin. If this is not the case, simply multiply s by -1.

The final matrix Z is a simple 4x4 matrix of the form:

$$Z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6.7 Estimating Radial Distortion Matrix (D) or Radial Distortion formula for pre-processing

The lens in a camera introduces distortions in the image with radial distortion being the one most commonly corrected for. As you get further from the center of distortion, straight lines are imaged as more and more curved. The point pairs used for the estimation of radial distortion are the image points and their paired calibration sheet point transformed, using the above estimated matrices,

to image points with the assumption being that any transformation error can be attributed to radial lens distortion.

This software is to implement two models for radial lens distortion; the first is based on a simple formula that needs one parameter estimated and is given a known center of distortion, but only works at low levels of distortion. The second estimates distortion using a 3x3 matrix but only works at high levels of distortion, such as with fish-eye lenses and cheap cellphone or web-cams. Note that with this second method the 3x3 matrix can be manipulated to also give a center for the distortion.

It is intended that the distortion matrix be initially estimated with a sanity check run on it to see if we should instead fail over to the radial distortion formula but this is not yet implemented. Instead only the first one parameter estimation is currently working with the code for the distortion matrix currently commented out as it has not been tested. For completeness, both methods are discussed here.

6.7.1 Radial Distortion Formula

The radial distortion formula is a variation of the standard infinite sum Taylor series introduced in [16, 17] and used with minor variations ever since: $r_u = r_d(1+k_1 r_d^2+k_2 r_d^4...)$, with $r_u$being the undistorted radius, $r_d$being the distorted radius, and $k_{1..\infty}$being the parameters to be estimated. The main variations are where to truncate the series with most finding that only estimating one or two terms is enough. In this case the formulation is simply truncated at the first term i.e.$r_u = r_d(1 + kr_d^2)$ or, a form that we will use later, $r_d^2 k = \frac{r_u}{r_d} - 1$. A point must be defined as the center of distortion and with no other relevant information the only point that can be used is the centre of the image.

We can use our set of point pairs to find a best fit k by constructing two matrices A and B and getting the least squares solution to $Ak = B$ where A and B are both nx1 matrices, and therefore k is scalar, or a 1x1 matrix. Each row of matrix A will contain a single column with the value r_d^2 and each row of B will contain the value $\frac{r_u}{r_d} - 1$. As can be seen each row will then be the equivalent of $r_d^2 k = \frac{r_u}{r_d} - 1$ for different $r_d$and r_u values but each row will have same value for k.

6.7.2 Undistorting the Image

Note that the above function is not easily reversible so that although we have a formula that tells us where the undistorted point is if we have a distorted point, we can not easily find where a distorted point maps to given an undistorted point.

For this reason if this is the distortion method used we must take the whole image, which is currently distorted, undistort each point, and resample it to get an approximately undistorted image for use in the latter sections. In comparison the Radial distortion matrix has a defined inverse so it is hoped that this time consuming step is not necessary and the distortion can be applied only to those points that need it.

As the undistorting of the image is a time consuming task a few enhancements are made. The undistorted points are stored in a Uniform 2D grid with approximately the same number of cells as there are pixels in the image. For

each pixel point to be re-sampled the grid is queried starting with the cell in which the pixel point resides, then moving out to include its eight neighbour pixels in a 3x3 grid centred on the starting cell, and then 5x5, or even 7x7 mask until at least one cell contains a point. The grey-scale values of the points in the cells queried are recorded, as well as the point placements and the grey-scale value of the new undistorted pixel is interpolated from.

6.7.3 Radial Distortion Matrix

The radial distortion matrix is based on [9] and the logic behind it is quite an involved process. A short summary only will be provided here. The Fundamental Matrix is a matrix that contains all the information about the relationship of two images of the same scene to each other and is normally calculated from the underlying camera parameters and rotation and translation matrices for each image. In this case the Fundamental matrix is being used to compare the set of distorted points to the set of undistorted points. The standard way to construct the Fundamental matrix is by use of the normalised 8 point algorithm which requires at least 8 point pairs. Once the Fundamental matrix is constructed then it can be decomposed to reveal the center of the distortion and a way of mapping between distorted and undistorted image points. There are however three degrees of ambiguity for this mapping which need to be solved before this becomes a unique mapping. This can be done by simply enforcing the constraints that are part of the definition of radial distortion, namely that the amount of distortion is based purely on the distance from the center of distortion, and that the order of a ranking of points based on their distance from the center of distortion does not change between the distorted and undistorted images.

After the Fundamental matrix is calculated we have six terms of the 3x3 matrix D and we need to find the other 3. The first row of D is the second column of F, and the second row of D is the negative of the first column of F, and the third row is the vector v that we still need to find. i.e.

$$D = \begin{bmatrix} F_{12} & F_{22} & F_{32} \\ -F_{11} & -F_{21} & -F_{31} \\ v_1 & v_2 & v_3 \end{bmatrix}$$

To get the final row we order the image points in order of increasing distance from the center of distortion and construct a matrix (n-1)x3 where each row is $(\lambda x, \lambda y, \lambda)$ where the x and y values are from the individual point coordinates of the calibration sheet coordinates and λ is a value calculated by comparing the radial distance from the center of distortion of this point to the radial distance of the next point in the list.

Once this is done we use SVD to find the minimum of $||Av||$ subject to the constraint that $||v|| = 1$. See Appendix B

6.8 Bundle Adjustment for Maximum Likelihood Estimation using Levenberg-Marquardt Algorithm

For a good introduction to the different algorithms used for minimisation problems, including the Levenberg-Marquardt (LM) algorithm, see [12]. Without getting into the specifics, we are trying to find the global minimum of a multi-dimensional function so we find a good first approximation and then we find the slope in the current neighbourhood and take a step in the "downward" direction

until we reach the minimum. If the first approximation was a good one, this will be the global minimum. The intricacies of the particular algorithm chosen are in how big a step we should take, in which direction, and what the stopping condition should be and as the original source code for the LM algorithm implemented came from [4] and the core of it has not been changed most of this can be glossed over.

What we do need to know is that the LM algorithm uses the partial derivative terms to determine the next step. In the original code, each problem needed to have its own partial derivative function pre-defined but this has been replaced with a generic partial derivative estimation method. Recall that a derivative is a measure of how much a function changes as the input changes with the partial derivative being simply the measure of how much a function with multiple inputs changes as a single one of those inputs changes with all others being equal. This measure is defined as the instantaneous value of the slope for a point in the limit where the change in the input becomes zero and in high school we learnt all the rules of thumb for taking a function and creating from it a derivative function which could then give the instantaneous slope for any input point.

However, in this case we define a general partial derivative and so go back to the basics and not worry about a derivative function but simply calculate the partial derivative of a input value as what happens if this input value is changed slightly. In the current code this is done by taking two values of 99.9999% and 100.0001% of the target input value, unless the target input value is zero, in which case we use +/-0.0001, leave all the rest of the inputs the same, and calculate the output value of the function at these two points. The partial derivative is then returned as the difference in the input values divided by the difference in output values.

We can now concentrate of defining the functions we wish to find the maximum likelihood and least error for.

6.8.1 2D Planar Homography

Normally the 2D planar homography estimated using the normalised DLT algorithm, see Section 6.2, is already near enough to the best estimate that this will not give a better answer but, just in case it will, the homography matrix is used as the nine dimensional function to minimise with the points in the calibration sheet as the input and the image points as the target output.

6.8.2 Camera Calibration

To minimise the adjustment that needs to be done the nine element rotation matrix is converted to a 3-vector using the Rodrigues formula which has the same information in a more compact form, specifically the axis of rotation is that of the vector and the rotation angle is given by the magnitude. This can be done with the following formulae; for a derivation of these see Appendix 4 of [8].

$$(R - I)r' = 0$$
$$2\cos(\theta) = trace(R) - 1$$
$$2\sin(\theta) = r'^T \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix}$$

$r = \theta r'$

where I is the 3x3 identity matrix, R is the original rotation matrix, r' is unit length direction vector which can be found from the first formula using SVD, θ is the rotation angle, and r is the final Rodrigues 3-vector. The trace function is simply the sum of the diagonal elements of the matrix.

It should be noted that in some formulations, the magnitude for the 3-vector is calculated directly using the arc-cos or arc-sin functions but as mentioned in Appendix 4 of [8] this is not numerically accurate and fails when the angle is 180 degrees. A better way is to feed the sine and cosine values into the two parameter arc-tan function i.e. $\theta = \arctan 2(\frac{1}{2} r'^T \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix}, \frac{trace(R)-1}{2})$

We will also need to go from the Rodrigues formulation to the full rotation matrix. This is accomplished with the following formula:

$R = I + \sin(\theta) [r']_\times + (1 - \cos(\theta)) [r']_\times^2$ where $[r']_\times$ is a 3x3 matrix representation of the cross-product function for the 3x1 vector r'. See Appendix C.

Once this has been done we can then make more efficient use of the LM algorithm to do a bundle adjustment on the various parameters: 3 for rotation, 3 for translation, 5 for internal camera calibration, 2 for the image origin, and either 1 for the lens distortion function, or 9 for the lens distortion matrix. The input is the set of centres of the calibration sheet ellipses transformed to the image, found by transforming multiple points on the circumference and finding the middle. The target is the set of image points paired with these ellipse centres, which, in the case of the 1 parameter distortion function, are changed to take account of this distortion.

7 Image Segmentation Technique

An important part of the process is to determine which parts of the image are part of the currently unknown object of interest and which parts of the image are part of the calibration sheet. The process for doing this relies on the assumption that lighting of the object is nearly uniform and that parts of the object and the calibration sheet near the edges are significantly different enough from their neighbouring background pixels that the edge detection algorithm can find these edges.

The image segmentation algorithm processes the edge map, detected ellipses, and grey-scale map to produce a 2D map where each cell has been processed to be one of four states: unknown, calibration sheet, edge, or other. The segmentation technique used is conservative in detecting the calibration sheet in that if a cell is set to calibration sheet it will be part of the calibration sheet but there may be other cells that have been miscategorised that may in fact be part of the calibration sheet. This is used later so that if a 3D point back projects to a calibration sheet cell in any image it is known to be part of the calibration sheet and thought to be part of the object if it does not.

The segmentation takes place in a series of steps:

1. The segmentation map is initially set with all cells inside the 4 sided polygon, or tetragon, formed by the calibration sheet corners are set to be

unknown and those outside are set to other. This is based on two barycentric coordinate transforms using two sets of three of the four corners of the calibration sheet transformed from real world to image coordinates. If one or more of the barycentric coordinates are negative then the cell is outside the triangle formed by these three corners. The two sets of corners are chosen so that they have two corners in common and these corners are diagonally opposite corners. This means that if the cell is outside both triangles it is, by definition, outside the calibration sheet in the image.

2. The edge map is processed to set some of the unknown cells to be edge cells with reference to the edge detection algorithm resolution so that for each edge point, cells within the radius of the resolution of this point are also set to the edge state.

3. The ellipses that have been mapped to calibration sheet circles are used to set cells to be in the calibration sheet state. Note that this should include some of the cells previously set to be edge state.

4. A flood fill algorithm is used on contiguous edge state cells to change them to calibration sheet state if they have any such cells on their border.

5. A weighted average grey-scale value is calculated for each ellipse center using up to eight points weighted by the inverse of their distance squared from the center point. The points used are selected by traversing in the eight cardinal directions from the center point checking each cell to see if it is unknown. When the furtherest such cell is found the grey-scale value of the points this distance away in each of the eight directions are compared to the grey-scale value of the center point and if the difference is over the threshold used for the edge map it is used in the weighted average.

6. The above grey-scale values for each ellipse center are used to calculate a weighted average grey-scale value for each unknown cell and this value compared to the actual grey-scale value. If the difference is above the threshold used for the edge map the cell is set to the other state.

7. A final flood fill is done to change contiguous unknown cells to calibration sheet state if they have any calibration sheet cells on their border.

For use in further processing this four state 2D array is post-processed down to a boolean 2D array where the cell is set to true if it is in the calibration sheet state or false otherwise.

This is currently a very basic segmentation technique that has a tendency to not work on shadows so is only really effective in an environment of highly diffuse lighting. This will need to be changed in the near future to a more robust algorithm.

8 Finding the Object

The initial rough estimate of the object uses the image segmentation to create a silhouette of the object using a voxelisation of the volume of interest. This will give a maximal shape of the object and may be able to define holes through the object but will not be able to show concavities. For example, if the object is

a coffee mug, this technique will be able to distinguish the shape of the handle but will not be able to show the inside of the cup, it will simply be a cylinder.

This "holes but not hollows" problem means a second technique should be used to further estimate the more fine-grained shape of the object. One such approach is using 3d edge estimation from the 2d edges detected in the images. This is currently not used and is likely to dropped in favour of a colour/texture matching algorithm in the future.

8.1 Voxelisation

The object is assumed to lie within the volume of interest: that volume of space immediately above the calibration sheet out to a height that is the larger of the dimensions of the calibration sheet. This volume of space is subdivided into equal sized 3d boxes called voxels. In the current implementation the volume is subdivided into a 128x128x128 grid by default. For each voxel, each of the eight vertices is categorised as being inside or outside the object by testing to see if it point back-projects to a point in all images and if so whether this point has been categorised as the calibration sheet in any image. If a voxel has all eight vertices categorised as outside, the voxel is defined as being outside the object. If all eight vertices are categorised as inside, the voxel is similarly categorised as being inside the object. If some vertices are inside and others outside, the voxel is said to be a surface voxel. Note that as the voxels share vertices, the actual implementation tests the 129x129x129 vertices and categorises the voxels accordingly.

However, as the image segmentation only definitely defines the calibration sheet, it may be that some voxels are categorised as inside or surface voxels when they are actually voxels that have vertices that back-project to the background in an image, not the object we are interested in. To overcome this we post-process the inside and surface voxels and set to outside those that do not have a pathway through other inside and surface voxels to the calibration sheet. This means voxels "floating in space" are identified and ignored.

This is a three step process where in the first step we start at the calibration sheet and sweep upwards a layer at a time and identify those inside and surface voxels that are attached to inside or surface voxels in the layer below (that are not marked for deletion) and mark for deletion any that are not. In the second step the inside and surface voxels left are used as a seed for a 3d flood fill where any voxels marked for deletion that are immediate neighbours of those that are not, are themselves reset to not be deleted. Finally, any voxels still marked for deletion are reset to be outside voxels.

This gives us the final output: a "chunky" representation of the maximal volume of the object.

8.2 3D Edge Estimation (currently not used)

For ease of calculation a single bounding box is calculated to enclose the "chunky" representation so that obviously irrelevant points or lines can be more easily excluded by testing them against this single bounding box. Each 2d edge point in each image is converted to a ray in space and tested to see whether or not it interests with, firstly, this bounding volume, then the more defined volume

inside this, our current volume of interest. If it does not interesect our volume of interest it can be ignored from this point on.

We now have a series of rays in space that intersect our volume of interest and we wish to match up the rays from each image that relate to the same point and triangulate them to give a precise 3d point in space.

To do accurate matching of these rays we need to rectify the pairs of images so that if a pair of edges relate to the same 3d point, they have the same x coordinate in the new rectified reference frame. The technique used to calculate the rectification homographies was taken from[11]. Once the matched points have been identified they are adjusted to minimise a cost function as in Algorithm 12.1 of [8].

Once all combinations of image pairs have been rectified and rays matched the 3d points are estimated using the linear triangulation method suggested in Algorithm 12.1 [8].

Each point is tested as the vertices of the voxels were above and ignored if it back-projects to the calibration sheet in any image.

In this way we end up with a point cloud of 3d points within the "chunky" voxelised maximum volume object representation found earlier.

There is however still the issue that whilst this technique will find the real 3d edge points it is also likely to come up with spurious ones as well. This can be seen by thinking of two images taken of a table-top corner where each image contains the corner but one image contains the edge clockwise from the corner and the other image contains the edge counter-clockwise from the corner. The two edges form the bases of two triangles in space the camera centres being the third points of the triangles. Assuming the camera positions are estimated accurately, the corner point will also be estimated accurately. However, the points along the respective edges will be used to estimate a series of points forming a diagonal line above the tabletop where the two triangles of the edge line projection planes intersect.

This issue is negated in the main by testing that all edge points are within the voxelised representation and do not back-project to the calibration sheet in any image but there are still spurious points identified so this is currently commented out.

9 Converting surface voxels to triangles

Each surface voxel has six faces. For each face of such voxels two simple tests are performed: Is the face at the edge of the volume of interest or is the neighbouring voxel sharing that face specified as an outside voxel? If either of these tests is positive the face is split into two triangles and these are added to the list of surface triangles. The triangles are arranged so that the two shared vertices are diagonally opposite corners of the voxel face. In each case the triangle normal is calculated so that it is oriented away from center point of the voxel.

10 Converting point cloud to triangles (currently unused)

Once a 3D cloud of points has been found as a representation of the object it must then be converted to non-intersecting triangles so that it can be output into a STL file. If the set of points were known to all lie of the convex hull of the object then we could use one of a number of methods of Delaunay triangulation to convert this hull into a number of triangles. However, this may not be the case and so we need to do something a little more complicated. First we carve the space around the point cloud into an interlocking grid of irregular tetrahedrons with triangles for their faces. The tetrahedrons so created will fill the space and by definition have faces in common with their neighbours. An internal tetrahedron will have all of its faces shared with a neighbour whereas a surface tetrahedron will have at least one face that is associated with that tetrahedron only. By eliminating the irrelevant tetrahedrons from this space a set of triangles describing the surface can be found as the faces of any tetrahedrons that are not connected to any other tetrahedrons.

The most convenient form of triangulation for these triangular face is the Delaunay triangulation which is a technique for taking a set of points, normally done in 2D, and creating non-intersecting set of triangles from them. The more formal definition is that no point is inside the circumcircle of any of the triangles. A circumcircle is the circle made by constructing a circle such that the three vertices are all on the circumference of the circle. By definition the center of this circle will be at the point of intersection of the three lines that bisect the three edges of this triangle and the radius is the distance from this centre to any of the three vertices. A Delaunay triangulation tends to avoid long skinny triangles and maximises the minimum of the three angles in the triangle.

The idea of Delaunay triangulation can be extended to 3 dimensions and create Delaunay tetrahedrons where each non-intersecting tetrahedron is made with four points which describe a circumsphere which does not contain any of the other points in the point cloud. As each of these tetrahedrons will have faces in common with its neighbours, having just the normal pointing a different way, a tetrahedron can be tested and if it is found that one face is not part of the object, the whole tetrahedron can be deleted which leaves the neighbouring tetrahedrons that had faces in common as surface tetrahedrons until such time as they are tested and potentially eliminated as well.

The DeWall algorithm, described in [**?**], was used as the basis of the space carving. This paper describes an extension to Delauney triangulation, to an arbitrary number of dimensions with a clever modification to the common "Divide and Conquer" recursion approach used to simplify the problem. In most "Divide and Conquer" methods there is significant amounts of time needed to stitch together the results from the various sub-problems but this is not the case in the DeWall algorithm as the problem is subdivided in a way that means there is no overlap in the results returned by the recursive subproblems. For simplicity the following explanation will be confined to the case of converting a 3D point cloud into tetrahedrons.

10.1 DeWall Algorithm

The DeWall algorithm is based on a recursive "Divide and Conquer" strategy whereby the function is given a list of triangular faces and initially just asked to put them into two lists based on whether they are on one side or the other of a dividing plane or a third if they span the dividing plane. The point list is also divided based on this dividing plane. The dividing plane is cyclically chosen to be a plane orthogonal to the 3 axes and is halfway between the midpoints of the point array when they are ordered by this axis's values. The list of triangular faces that span the dividing plane are processed one at a time to find a fourth point that fulfils the criteria to create a Delaunay tetrahedron with the minimal Delaunay distance, when possible. This Delaunay distance is simply the radius of the circumsphere created by these four points or its negative if the circumsphere centre and the fourth point are on opposite sides of the plane created by the first three points.

This tetrahedron is then added to the final list of tetrahedrons and the faces of this tetrahedron are then added to one of the three lists of triangular faces. Processing continues until no more triangular faces remain to be processed in the list that spans the plane. At this point, if one or both of the other lists contains any triangles to test the procedure is called recursively with this list and the appropriate half set of points.

To seed this an initial tetrahedron needs to be created. This is done by building the tetrahedron up one point at a time: the first point is the point closest to the initial dividing plane, the second as the point on the other side of the dividing plane that is closest to the first. The third is created using the first two points as two vertices of a Delaunay triangle with the third point that one which allows the creation of a circumcircle with the minimum radius. The fourth is then created in much the same way as a normal Delaunay tetrahedron given a triangular face.

This is the core algorithm but there were also some additional enhancements for speed. The main one is that instead of testing all points to find the minimal Delaunay distance, the points are partially ordered by using a 3D Uniform Grid arrangement and points in the cells are tested in stages of increasing distance from the triangular face. If a known minimum solution is found in an earlier stage the testing terminates early.

To gather information on how complete the algorithm is i.e. information used to update the progress bar, as each point is used in new face a counter is incremented and each time a face is used to try and create a tetrahedron the counter for each of the three points in the triangle are decremented. Whenever the count decrements to zero the point has been finished with.

10.2 Problems with the DeWall Algorithm and implemented solutions

It is not mentioned in the paper but in the documentation bundled with the code referenced in the paper it mentions that sometimes the algorithm gets into trouble and so an additional check is needed for Cyclic Tetrahedron Creation (CTC). This simply means that before a tetrahedron is added to the output list it is checked that it does not already exist in the list. If this situation were to arise and nothing was done this could lead to an infinite loop. The main reason

for this is mis-categorisation of the triangular faces due to rounding errors on the calculated split-plane.

Note that if the additional faces to be processed are always added to the list being currently processed then this algorithm ceases being recursive and becomes simply sequential and so the leading cause of the Cyclic Tetrahedron Creation is eliminated. However, this makes the resulting algorithm magnitudes slower and is not a very elegant solution. A hybrid approach leads to the creation of a threshold value, in this case a minimum number of points to be processed, for changing this algorithm from recursive to sequential. It was found that setting this threshold to 100 points did not delay the algorithm much and eliminated most of the test cases where a CTC fault was detected. In all other cases it was found that changing the orientation of the initial slice plane eliminated the problem. For completeness there is also a final fallback to a fully sequential algorithm if CTC faults are detected for are 3 orientations of the initial dividing plane. It may be that a better solution would be to dynamically double the minimum number of points threshold and try each orientation again until success occurs but with no test cases it is uncertain how long this would take as opposed to going directly to the strictly sequential implementation.

There is also another problem with the algorithm in that the creation of the first tetrahedron will fail in two pathological cases: if the four points selected are all in the same plane or there exists a fifth point co-spherical with the first four i.e. the five points lie on the surface of the same sphere. Unless the points are all in a grid arrangement or all points are actually points on the surface of a sphere, it is likely that changing the initial slice plane is all that is required. Given that the points themselves are currently being calculated it is unlikely that either case will actually arise.

10.3 Elimination of non-object tetrahedrons

The core of the concept of spotting and eliminating spurious tetrahedrons is the idea that in estimating a 3d point it is inherently assumed that nothing is between the camera centres and the the 3d point to be estimated. As we have a list of 3d points and know the images they were estimated from and also where the the camera centre was for these images, we can eliminate any tetrahedrons that have a face that is intersected by a line segment constructed by one of the points in the point cloud and one of the camera centre points associated with the estimation of the point. In the absence of noise this will work to reveal the surface of the object, including concavities but with noise it may also split the object into disjoint pieces. This seems to be the same concept described briefly in Section 4 of [13] as “simple carving”. It may be that in the future this be enhanced with the probabilistic carving described in the same section. Note that if the object is disjoint the piece that contains the point with the lowest z value is considered the valid section.

Once the non-object tetrahedrons have been eliminated a simple list of the surface triangles can be compiled creating an ordered list of triangular faces, similar to the AFL in the DeWall algorithm, that has each of the four faces of each tetrahedron added in turn and checked to see if the face is already inserted and set the d1 and d2 values appropriately. This will then give a list that contains each triangular face and an indication of how many tetrahedrons it is associated with. The surface triangles will be those triangular faces that only

belong to one tetrahedron.

11 Writing the STL file

Having compiled a list of triangles, it is simple enough to create an STL file. There are two versions of the STL file format, the ASCII representation and the more compact binary version. Currently only the ASCII version is implemented but it is planned that the binary format will be supported in the 1.0 version. The STL file format contains a list of triangles and for each triangle there is a normal vector, of length 1, and the three 3d points representing the vertices of the triangle. It should be noted that to get rid of any issues with rounding errors in calculations producing gaps in the surface mesh where there should not be any, the triangles are stored internally with a list of 3 integers which are pointers to an index of 3d points. This also means that in previous stages triangles can be compared for equality based on the indices for their vertices without having to worry about trying to compare floating point numbers. It is only at this step that the triangle vertices are converted to triples of floating point numbers. The format of the ASCII version of the STL file is described in [6] along with the note that some applications read the normal vector from the file whereas others calculate it from the three points of the triangle. The standard solution is that the normal vector is calculated simply by the cross-product of the vectors AB and AC but this could give a vector pointing in the opposite direction to the desired one. If this method gives a different result from the stored normal vector then the solution is to swap points B and C when writing the file so that either method gives a vector in the same direction.

Note that a common problem in STL file creation is the breaking of vertex-vertex rule which says that all adjacent triangles share two vertices. This is not a problem in the current implementation but needs to be kept in mind when modifications are made in the future.

12 Conclusion

Once the STL file has been created this can be imported into a 3d CAD program and used as the starting point for the digital design file of the real world object. It is hoped that in the future this software becomes good enough that in most cases the output file will not have to be manipulated and can be used by a 3d printer, such as one built by the DIY 3d printer project Reprap[5], to give the user an accurate copy of the object. The quality of the output at the present time leaves a lot to be desired however.

The ultimate aim is for this software to be integrated into the software for the Reprap project and the Reprap printer redesigned with a computer controlled camera so that the Reprap can become a push-button 3d “photocopier”. There is still a long way to go but it is hoped that developers will build on what this software does to make it possible.

A Derivation of Zhang formulae for Camera Matrix Parameters

A knowledge of the how the formulae for the camera matrix parameters was derived is not needed to make use of them but is included for completeness as only the outline is given in the original paper and they were re-derived during bug-fixing to make sure the formula were correct.

Given that we have created a matrix B that is the IAC (ω) up to scale and that the IAC can be expressed in terms of the Camera Matrix K, we can express the components of B in terms of the components of K plus a scaling factor λ. Our starting point is the following definitions:

$B = \lambda\omega$

$\omega = (KK^T)^{-1} = (K^{-1})^T K^{-1}$

$KK^{-1} = K^{-1}K = I$ (the standard definition of an inverse of a matrix)

So, we solve the last equation first to find K^{-1}in terms of the camera parameters used in K then substitute the answer into the other two equations.

$K^{-1}\begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ leads to 9 equations which can be trivially solved for the components of $K^{-1}$to give:

$$K^{-1} = \begin{bmatrix} \frac{1}{\alpha} & \frac{-\gamma}{\alpha\beta} & \frac{v_0\gamma - u_0\beta}{\alpha\beta} \\ 0 & \frac{1}{\beta} & \frac{-v_0}{\beta} \\ 0 & 0 & 1 \end{bmatrix}$$

substituting this into the equation $\omega = (K^{-1})^T K^{-1}$gives the expansion labelled (5) in the Zhang paper

$$\omega = \begin{bmatrix} \frac{1}{\alpha^2} & \frac{-\gamma}{\alpha^2\beta} & \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} \\ \frac{-\gamma}{\alpha^2\beta} & \frac{\gamma^2}{\alpha^2\beta^2} + \frac{1}{\beta^2} & \frac{-\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} \\ \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} & \frac{-\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} & \frac{(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1 \end{bmatrix}$$

Substituting this into the equation $B = \lambda\omega$ gives:

$$\begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix} = \lambda \begin{bmatrix} \frac{1}{\alpha^2} & \frac{-\gamma}{\alpha^2\beta} & \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} \\ \frac{-\gamma}{\alpha^2\beta} & \frac{\gamma^2}{\alpha^2\beta^2} + \frac{1}{\beta^2} & \frac{-\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} \\ \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} & \frac{-\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} & \frac{(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1 \end{bmatrix}$$

Taking these equations out of the matrix form gives 6 equations:

$B_{11} = \frac{\lambda}{\alpha^2}$

$B_{12} = \frac{-\lambda\gamma}{\alpha^2\beta}$

$B_{13} = \lambda\frac{v_0\gamma - u_0\beta}{\alpha^2\beta}$

$B_{22} = \frac{\lambda\gamma^2}{\alpha^2\beta^2} + \frac{\lambda}{\beta^2}$

$B_{23} = \lambda(\frac{-\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2})$

$B_{33} = \lambda(\frac{(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1)$

We also have the knowledge that α and β are the aspect ratio of the pixels and are therefore positive and that the scaling factor,λ is not zero.

We can now relate these to the Zhang equations:

$$v_0 = (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2) \tag{1}$$

As both the numerator and denominator of this equation are used in further calculations they are expanded out here and labelled :

$Numerator = (B_{12}B_{13} - B_{11}B_{23}) = \frac{\lambda^2 v_0}{\alpha^2\beta^2}$

$Denominator = (B_{11}B_{22} - B_{12}^2) = \frac{\lambda^2}{\alpha^2\beta^2}$

We can now see that dividing the numerator by the denominator is simply $\frac{\lambda^2 v_0}{\alpha^2\beta^2} / \frac{\lambda^2}{\alpha^2\beta^2}$ in which all the terms cancel out except for $v_0$given that none of α,β or λ are zero.

$$\lambda = B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11} \tag{2}$$

Again this is set up so that all but a single λ term on the right hand side cancels out. This can be thought of as taking the terms of $B_{33} = \lambda(\frac{(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2}) + \frac{\lambda v_0^2}{\beta^2} + \lambda$ and constructing two equations to be taken away from it to take out the terms we don't want i.e. $\lambda(\frac{(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2}) = B_{13}^2/B_{11}$ and $\frac{\lambda v_0^2}{\beta^2} = Numerator * v_0/B_{11}$ which when both taken away from B_{33} leave simply λ and so give the above equation.

$$\alpha = \sqrt{\lambda/B_{11}} \tag{3}$$

This is simply a rearrangement of the equation for B_{11}. Given our knowledge that α is positive we obviously use the positive square root.

$$\beta = \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)} \tag{4}$$

This manipulates the above calculated Denominator using λB_{11} to cancel out the $\alpha$and λ terms, leaving just β^2 term from which we recover the positive square root i.e. $\lambda B_{11} = \frac{\lambda^2}{\alpha^2}$ so $\frac{\lambda^2}{\alpha^2} / \frac{\lambda^2}{\alpha^2\beta^2}$ simplifies to β^2given our knowledge that neither α nor λ is equal to zero.

$$\gamma = -B_{12}\alpha^2\beta/\lambda \tag{5}$$

This is simply a rearrangement of the equation for B_{12}.

$$u_0 = \gamma v_0/\beta - B_{13}\alpha^2/\lambda \tag{6}$$

This is simply a rearrangement of the equation for B_{13}.

B A Note on Singular Value Decomposition

Singular Value Decomposition (SVD) is the Swiss-army knife of matrix manipulation and is used in a variety of circumstances. It is a decomposition of a matrix, A, with at least as many rows (m) as columns (n), into 3 distinct matrices of the form $A = USV^T$ where U is an m x n matrix with orthogonal columns, S is an n x n diagonal matrix with positive entries arranged in descending order down the diagonal, and V is an orthogonal n x n matrix.

In the context of this piece of software SVD is normally used to find the solution to an equation of the form $Ab = 0$ where A is a constructed matrix and b is a n x 1 vector to be found. In these cases, provided there are enough rows of A, then the solution is simply the last column of V. If the number of rows is less than the number of columns then it is common practice to bulk up the matrix with rows of zeroes.

If the rank of the matrix so constructed is one less than the number of columns then the last column of V gives the exact vector b such that $Ab = 0$. If the matrix is full rank then the problem is over-constrained and the last column of V gives the least squares minimisation to the solution. Note that if the number of columns is greater than the rank by two or more then the problem is under-constrained and there are multiple solutions.

A related use of SVD is finding the minimum of $||Ax||$ subject to the constraint that $||x|| = 1$. Once again the solution is simply the last column of V.

Another use of SVD is in the taking a general 3x3 matrix and finding the closest rotation matrix to it. In this case if $R'$is the general matrix, and R is the rotation matrix you want, $R' = USV^T$ and $R = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(UV^T) \end{bmatrix} V^T$. Most references recommend the simpler $R = UV^T$ but as mentioned in [1] this may in fact be the reflection of the real rotation matrix if the determinant is -1, hence the adjusted formula.

A matrix can also be decomposed $A = U'P$ such that U' is a unitary portion ($U'^T U' = I$) and P is the symmetric positive semi-definite portion meaning $x^T Px >= 0$ for all vectors x. If the original matrix A is invertible then this decomposition is unique and the matrix P is symmetric positive definite, meaning $x^T Px > 0$ for all vectors x. This decomposition can be constructed from the SVD in the following way: if $A = USV^T$then $U' = UV^T$ and $P = VSV^T$.

C Cross Product Matrices

The standard 3-vector cross-product $a \times b = (a_2b_3 - a_3b_2, a_3b_1 - a_1b_3, a_1b_2 - a_2b_1)^T$ can be described by a specialised form of a matrix multiplication, represented by $[a]_\times b$ where $[a]_\times$is a special 3x3 matrix made from the 3-vector a in the following way:

$$[a]_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

This matrix is skew-symmetric, meaning $[a]_\times^T = -[a]_\times$

Other properties of this matrix include the relationship $[a]_\times b = (a^T [b]_\times)^T$ and that the vector a is both the left and right null vector of $[a]_\times$ i.e. $a^T[a]_\times = 0^T$ and $[a]_\times a = 0$

References

[1] Computation of the rotation matrix. http://www.kwon3d.com/theory/jkinem/rotmat.html.

[2] Exif file format. http://www.exif.org/specifications.html.

[3] Khan academy. http://www.khanacademy.org/.

[4] Levenberg-Marquardt in java. http://scribblethink.org/Computer/Javanumeric/index.html.

[5] Reprap homepage. http://reprap.org/bin/view/Main/WebHome.

[6] STL (file format) - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/STL_(file_format).

[7] Fahad Alzahrani and Tom Chen. Real-time high performance edge detector for computer vision applications. In *Asia and South Pacific Design Automation Conference*, pages 671–672, Chiba, Japan, January 1997.

[8] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[9] S. B. Hartley, R. Kang. Parameter-Free radial distortion correction with center of distortion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8):1309–1321, 2007.

[10] L. A. Kenna. Eccentricity in ellipses. *Mathematics Magazine*, 32(3):133–135, February 1959.

[11] C. Loop and Z. Zhang. Computing rectifying homographies for stereo vision. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 125–131, 1999.

[12] K. Madsen, H. B. Nielsen, and O. Tingleff. Methods for Non-Linear least squares problems (2nd ed.), 2004.

[13] Q. Pan, G. Reitmayr, and T. Drummond. ProFORMA: probabilistic feature-based on-line rapid model acquisition. In *Proc. 20th British Machine Vision Conference (BMVC)*, London, September 2009.

[14] Al-Ajlouni, S. and CS Fraser. Zoom-Dependent calibration for Consumer-Grade cameras. In *ISPRS Commission V Symposium 'Image Engineering and Vision Metrology'*, pages 20–25, Dresden, Germany, September 2006.

[15] B. Triggs. Autocalibration and the absolute quadric. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 609–614, June 1997.

[16] Roger Y. Tsai. An efficient and accurate camera calibration technique for 3D machine vision. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Miami Beach, FL*, pages 364–374, 1986.

[17] Roger Y. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal of Robotics and Automation*, 3(4):323–344, August 1987.

[18] Y. Xie and Q. Ji. A new efficient ellipse detection method. In *Proceedings of the 16th International Conference on Pattern Recognition*, volume 2, page 20957, Quebec City, QC, Canada, 2002.

[19] S.C. Zhang and Z.Q. Liu. A robust, real-time ellipse detector. *Pattern Recognition*, 38(2):273–287, 2005.

[20] Z. Zhang et al. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, November 2000.