Queensland University of Technology

Honours thesis

Planar Calibration of Camera Arrays for Light Field Acquisition

Ashley Stewart

supervised by

Dr. Donald Dansereau

November 9, 2016

Abstract

Light field cameras are an emerging technology with unique post-processing capabilities. Their applications in computer vision are numerous, especially in robotics, navigation systems and surveillance. Within the medical field, there is a serious potential for occlusion removal in endoscopic imaging and surgical theatre videos using light field cameras. In order for results to be achieved in any such application, calibration is essential. Existing calibration procedures are either conceptually complex metric procedures, or non-metric procedures inflexible to camera orientation and planarity. We present a novel and conceptually simple non-metric calibration for light field acquisition, suitable for camera arrays with constant yet arbitrary camera poses. We also provide a quantitative measure of calibration accuracy, and use it to demonstrate the procedure's efficacy with our Raspberry Pi camera array. Additionally, we present qualitative results by rendering light fields at varying levels of focus and occlusion, and demonstrate success in capturing and rendering light field video - an area of particular interest for our applications. Finally, we reflect on implementation challenges and lessons learned.

Contents

1 Introduction

1.1 Motivation

Light field cameras are an emerging technology with unique post-processing capabilities. Along with providing useful spatial information for computer vision, light field cameras can identify objects, construct artificial views, and render with increased focus in poor visibility. Their applications in computer vision are numerous, especially in robotics, navigation systems and surveillance.

Light field technology in the medical field may facilitate the removal of occluders from surgical images, such as those obtained by an endoscope during arthroscopic procedures. Vision through an endoscope in such procedures is often hindered by floating debris, bubbles and equipment. The US corporations Stryker and Intuitive Surgical have both expressed interest in applying computer vision technology in these areas. In addition, light field technology could also be applied in the surgical theatre, providing useful views for training and education. This could be achieved using occlusion removal techniques with light field video.

To enable any light field camera to perform well in such applications, accurate calibration is essential. A calibrated light field camera is able to capture light fields because it is aware of the relationships between each of its views. Our implementation uses a camera array, and although several calibration procedures exist for camera arrays, current procedures are either conceptually complex 'full metric calibrations', or they are inflexible to camera orientation and non-planar setups.

1.2 Outcomes

We present a novel and conceptually simple calibration for light field acquisition, suitable for camera arrays with arbitrary camera poses. We also provide a quantitative measure of calibration accuracy, and use it to demonstrate the procedure's efficacy with our Raspberry Pi camera array. Additionally, we present qualitative results by rendering light fields at varying levels of focus and occlusion, as well as success in capturing and rendering light field video - an area of particular interest for our applications. Finally, we reflect on

implementation challenges and lessons learned.

2 Prior work

2.1 Implementation

The Raspberry Pi camera array, when initially implemented by Rafe Denham, achieved poor light field rendering quality (Denham, 2015) (see figure 1). The calibration method used was one proposed by Vaish et al., and involves a *plane + parallax* framework (Vaish, Wilburn, Joshi, & Levoy, n.d.). In Denham's evaluation, a definitive reason for the poor rendering quality is not provided, but it is suggested that it may have been due to an incorrect implementation of Vaish's algorithm.

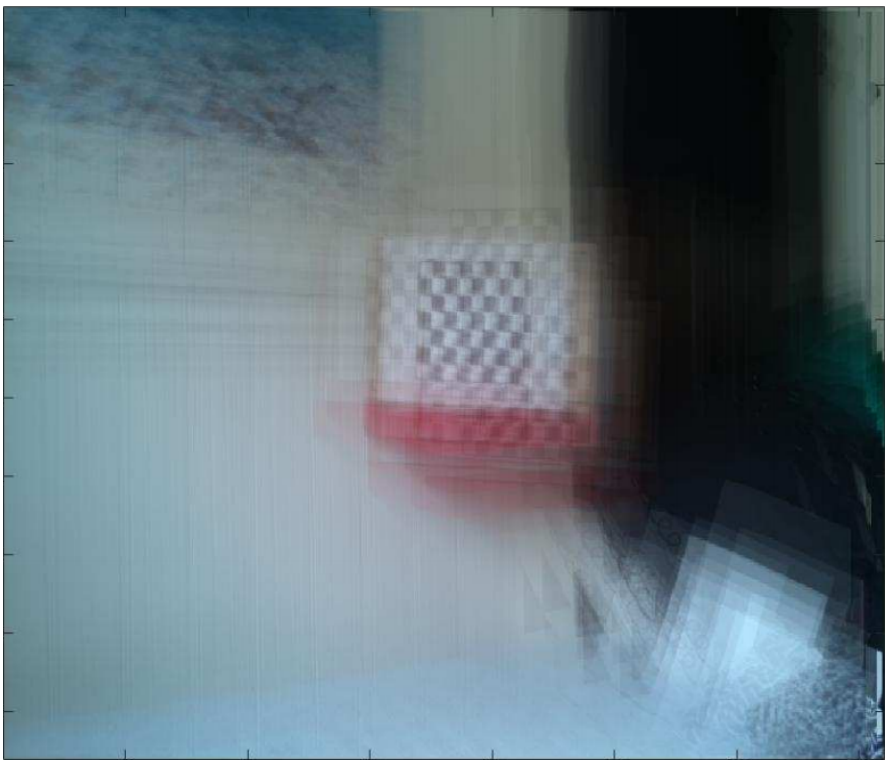


Figure 1: Light field calibrated using Vaish's plane + parallax procedure exhibiting poor focus. Adapted from Denham, R. (2015, June). *Light field array camera* (Tech. Rep.). Queensland University of Technology.

After some testing of the original, we discovered that although the cameras appeared to be planar with a uniform viewing direction, in reality they exhibited significant arbitrary rotation, which Vaish's method does not account for. The lack of conformity in poses may have been at least partially related to the original medium-density fibreboard front-plate, which has since been replaced (see section 3). However, even after replacing the front-plate, the cameras continue to retain significant arbitrary rotation (see figure 2).

Figure 2: Images taken from two horizontally adjacent cameras in our camera array. The red lines illustrate the variance in camera orientation. This variance must be accounted for if calibrated light fields are to be generated.

2.2 Calibration work

2.2.1 Identifying the research gap

Calibration procedures that facilitate light field acquisition can be categorised as either metric or non-metric. Metric calibration procedures recover precise camera positions and orientations. These methods are conceptually complex and take time to implement. Non-metric procedures, while simple, only recover camera positions to some unknown scale. However, it turns out that this is acceptable for most light field applications including 3D reconstruction, synthetic aperture imaging, light field rendering and space-time view interpolation.

Vaish's *plane + parallax* calibration procedure is the only non-metric light field camera calibration procedure that we are aware of. The procedure is one of the simplest procedures for light field acquisition, and it is currently used

by Stanford University for their multi-camera array. The procedure requires each camera to be arranged on a plane. Images can then be projected onto a reference plane parallel to the camera plane, by applying a homography matrix. The x, y positions of cameras (to an unknown scale) can then be recovered by measuring the parallax of a single feature point that does not lie on the reference plane. A limitation of this procedure is that it requires all cameras to lie on a plane and have a uniform viewing direction. As discussed, our cameras exhibit arbitrary rotation, which cannot be accounted for by this procedure (see figure 2). Vaish et al's experiments indicate that their non-metric method actually yields results better qualitative results than those acquired by a full metric calibration.

Perhaps the most appropriate existing calibration procedure for our camera array is one proposed by Xu et al (Xu, Maeno, Nagahara, & Taniguchi, 2014). The procedure extends Zhang's single-camera calibration (Zhang, 2000) to provide a metric calibration for mobile camera arrays. This directly solves the camera pose problem, because the procedure works with camera arrays irrespective of their planarity or camera poses. However, this is a metric procedure that goes beyond our needs, and is conceptually complex - which is what Vaish's method, while inflexible, intended to address (with the addition of providing better qualitative results).

For our camera array, we have a need for a new calibration procedure that bridges the gap between Vaish's method and Xu's method - one that is flexible to orientation like Xu's, yet maintains the non-metric simplicity of Vaish's.

2.2.2 Leading the way to panoramic calibration

A *monocular co-registration* procedure described by Donald Dansereau can be extended and formalised as a novel calibration (Dansereau, 2014). The procedure relates conventional 2D images via the plenoptic function, since any conventional image can be considered some subset of the function. Dansereau states that given a set of images captured by colinear cameras, it is possible to reproject the images into a common parametrisation, so long as the images overlap sufficiently. If we determine the geometric transforms separating images via some reference plane in advance, we can exploit this parametrisation to align images and construct light fields (see figure 3 and figure 4).

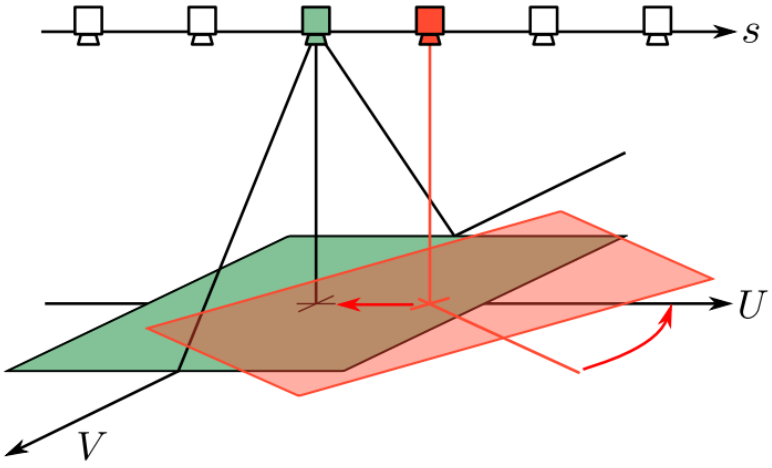


Figure 3: The simplified case of a planar scene and co-linear cameras having only rotations about the principal axes and translations parallel to the scene. Adapted from Dansereau, D. G. (2014). *Plenoptic signal processing for robust vision in field robotics* (Unpublished doctoral dissertation). The University of Sydney.

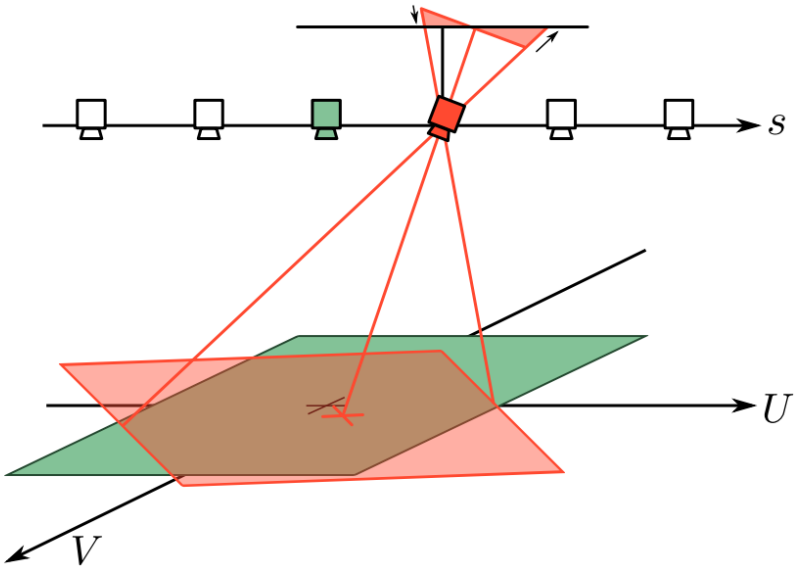


Figure 4: In the case of cameras having arbitrary rotations, images can be reprojected onto a plane parallel with the U, V plane, based on an estimate of the camera's orientation. Adapted from Dansereau, D. G. (2014). *Plenoptic signal processing for robust vision in field robotics* (Unpublished doctoral dissertation). The University of Sydney.

3 Adjustments to the implementation

Our camera array's casing was originally built using medium-density fibreboard. After some testing, it was clear that the material significantly reduced the potential for a consistent calibration. This is because medium-density fibreboard is simply too bendy to maintain its form. This likely contributed to the poor quality of the original light field renders. Fortunately, we have been able to address this by designing, building, and installing a sturdier aluminium front-plate (see figure 5). The design specifications for the new front-plate are provided in appendix 9.1.

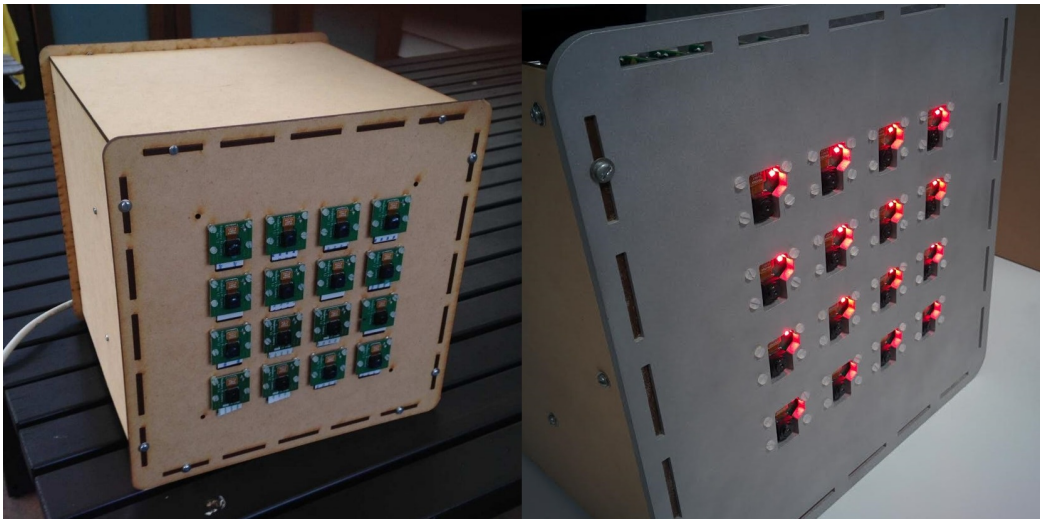


Figure 5: The original camera array (left) and the camera array with the new aluminium front-plate installed (right).

Although the front-plate is now sufficiently sturdy and maintains its form, the cameras themselves still retain some arbitrary rotation (see figure 2). This rotation may be due to the nylon nuts and screws used to mount the cameras, which do not fit as tightly as stainless steel materials would. Stainless steel nuts and screws were considered, but were ultimately rejected because they may have damaged the cameraboards or caused short-circuits. The cameras themselves may also have some slight variance if they are not built to be absolutely identical. Fortunately, this rotation can be addressed by our novel calibration procedure.

4 Procedure

4.1 Calibration

The monocular co-registration procedure introduced in section 2.2.2 can be extended and formalised as a novel non-metric calibration procedure suitable for our camera array. The key part of the procedure lies in the estimation of geometric transforms that relate image pairs captured by the camera array, relative to some reference plane. Our first step will be to capture a full array image of a calibration pattern positioned at the reference plane. This image set will be called the *calibration set*, and it will will inform our estimation.

In order to estimate the geometric transforms between images in the calibration set, we need to identify and match features that are present in image pairs. Therefore, the reference plane for our calibration set should be one that provides sufficient overlap across camera views (see figure 6).

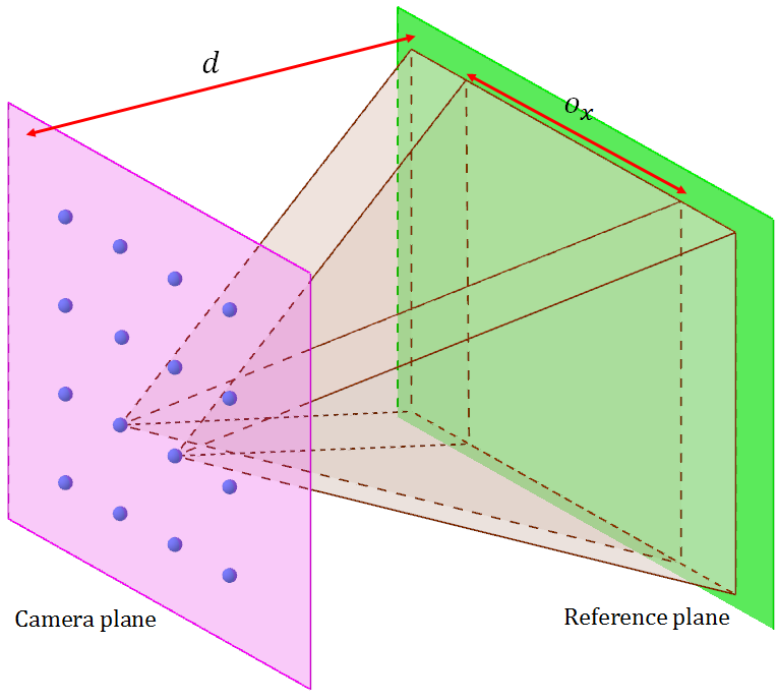


Figure 6: Diagram of our camera array and reference plane. Two camera views are projected onto the reference plane, which is set up at distance d from the camera plane. Significant overlap in the x direction is illustrated as o_x.

It is clear that the degree of overlap between any two camera views is a function of each camera's field of view, viewing direction, rotation about the principal axis, as well as translation between cameras and the distance d from each camera to the reference plane.

We achieve 100% overlap for any two cameras if the angles between the outer bounds of both projections are at most 180°, with the distance to the reference plane d approaching ∞, and with a calibration pattern that approaches dimensions of $\infty \times \infty$. Thus we should consider that as d increases, so does the required size of the calibration pattern. Additionally, geometric transforms estimated using very large d compared to the distance of captured scenes of interest will result in increased sensitivity when attempting to adjust light field focus. Clearly, the most practical calibration setup will depend on the resources available, the camera setup, and the intended use of the camera. For our calibration setup, see section 5.1.

In order to estimate geometric transforms relating images via the reference plane, we must be able to identify common features between images. To ensure this is possible, our calibration pattern must meet certain requirements. Most calibration procedures involve a checkerboard pattern as part of calibration, but this is inappropriate for our procedure since we rely on the detection of *unique* features. Checkerboards and other repeating patterns have non-unique surface features which can easily be mismatched with other identical features. Therefore, the calibration pattern must be a relatively detailed, non-repeating planar pattern, positioned parallel to the camera plane at a distance providing significant overlap. The colorful impressionistic paintings of Leonid Afremov have proven effective for our setup.

Once we have a calibration set that meets the outlined requirements, we can begin to develop our transformation set for rectification. For each image in the calibration set, we can identify surface features automatically via the Speeded Up Robust Feature detection algorithm (SURF) (Bay, Tuytelaars, & Van Gool, 2006). SURF has been shown to identify sufficient features quickly and effectively. We can then identify unique matching surface features between successive image pairs, and thus estimate their relative geometric transforms. Estimating geometric transforms can be achieved by applying a *RANdom SAmple Consensus* algorithm (RANSAC). We have opted to use *M-estimator SAmple and Consensus* (MSAC), which additionally evaluates the quality and likelihood of the consensus set (Torr & Zisserman, 2000). We can exploit this to enforce a minimum confidence level required to produce a positive match. A MATLAB implementation of our calibration procedure

has been provided in appendix 9.4.1.

4.1.1 Measuring calibration accuracy

To quickly assess the accuracy of our transformation set, we can use the transformation set to project all the images from the *calibration set* onto a new image plane, effectively constructing a panorama. It turns out that constructing such a panorama provides an effective means to visually assess the quality of the transformation set (see figure 7). A good transformation set will result in a smoothly stitched panorama.

Figure 7: Jagged panorama of a calibration set (left) and smooth panorama of a calibration set (right). The calibration set used to construct the smooth panorama will produce better results for light field applications. The jagged panorama was built using a poor calibration set which was not of a fully planar scene. Ignore any clear differences in color across the panoramas - this is due to automatic color balancing and gamma correction in the camera modules.

To more accurately assess the quality of a transformation set, we can evaluate the positional consistency of surface features across images in the panorama, one image at a time. The closer each surface feature is to having a uniform position across images, the better the transformation set is for light field applications.

4.2 Rectification

A transformation set built using the procedure described in the previous section (section 4.1) can be used to rectify any set of images captured by the camera array. Rectification will bring images into light field alignment and enable popular light field applications such as synthetic aperture focusing.

In section 4.1.1, we suggested that projecting calibration images onto a panorama plane is useful as a quick way to assess the accuracy of a transformation set. This is very similar to the rectification process. To rectify an image, we can simply project only the image of interest onto the panorama plane using the appropriate transformation (see figure 8). Repeating this for all images in a set will rectify the set for light field acquisition and rendering. A MATLAB implementation of the rectification procedure is provided in appendix 9.4.2.

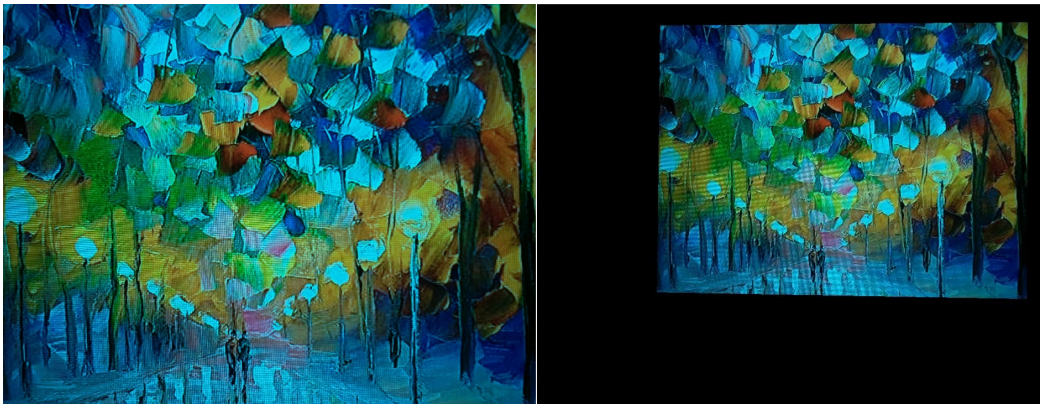


Figure 8: Original image (left) and rectified image (right).

Rectified images have a side effect of having potentially large areas of blackness. This may be inconvenient or significantly affect the file size of large sets. It is possible to crop rectified images, but this also requires that the relative positions of cameras be known. This is because the translations between cropped portions must correspond to the relative camera positions associated with each image. Cropped images must also be of uniform dimensions. Figure 9 illustrates both possibilities. Our method only allows relative camera positions to be recovered as an approximation, and only if all cameras have a uniform viewing direction.

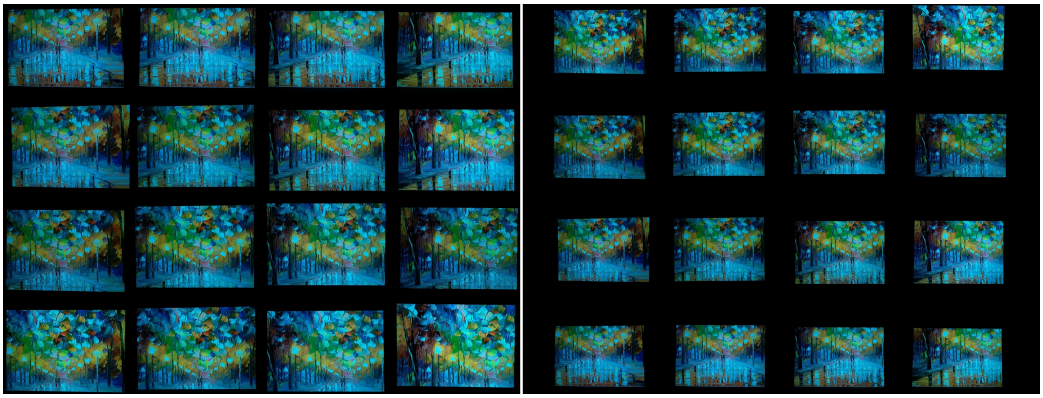


Figure 9: Cropped set (left) vs. uncropped set (right). In the uncropped set, each image is the size of a full panorama.

5 Results

This section presents results obtained using the proposed calibration procedure with the Raspberry Pi camera array.

5.1 Our implementation

So far, we have found that the most effective calibration patterns are detailed images such as paintings (see figure 10).

We have positioned our Raspberry Pi camera array parallel to a TV displaying such a pattern (see figure 11). This setup makes it easy to change the pattern while keeping the same reference plane distance.

Figure 10: A calibration pattern that achieved good results for our camera. The image is of Leonid Afremov's *Farewell to Anger*.

Figure 11: Our calibration setup. We have displayed our calibration image on a TV to ensure planarity, and aligned by visual inspection.

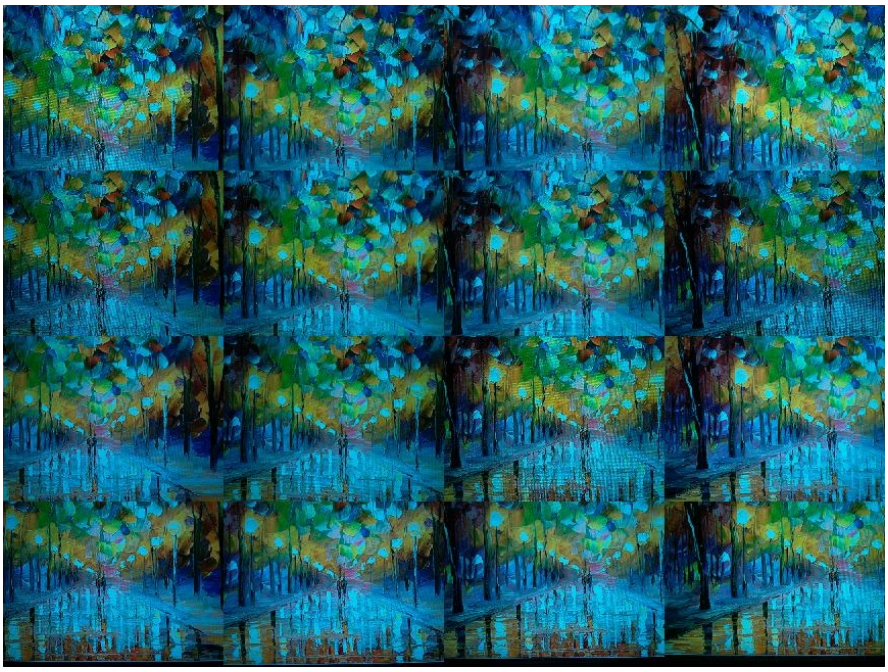


Figure 12: Mosaic of our calibration set

5.1.1 Overlap measurement

For our 4x4 camera array, we have camera translations of $t = 34.5$ mm, camera fields of view $\phi = (53.5°, 41.41°)$, and a distance to the reference plane $d \approx 300$ mm. Using basic geometry, and by assuming a precisely co-planar camera array and perpendicular viewing directions, we have:

$$\begin{aligned} \text{Projection width } P_x &= 2d\tan(\frac{\phi_x}{2}) \\ &= 2 \times 300 \times \tan(\frac{53.5}{2}) \\ &= 302.424 \text{ mm} \end{aligned} \tag{1}$$

$$\begin{aligned} x \text{ overlap} &= \frac{P_x - t}{P_x} \\ &= \frac{302.424 - 34.5}{302.424} \\ &= 88.59\% \end{aligned} \tag{2}$$

So we have approximately 88.59% overlap between camera views in the x direction. Applying the same equations in the y direction, we calculate approximately 84.78% overlap. This degree of overlap along with the chosen calibration pattern has demonstrated good results for synthetic aperture focusing applications.

5.2 Calibration accuracy

In section 4.1.1, we suggest that calibration accuracy can be tested by comparing the relative coordinates of features across rectified images. For a planar scene (such as the calibration set scene), perfect calibration is achieved when coordinates of features match precisely. A MATLAB implementation of this assessment procedure is provided in appendix 9.4.3.

We have calculated such pixel inconsistencies over multiple passes of the calibration procedure (see table 1). Multiple passes are achieved by estimating and transforming already rectified images successive times.

# of passes	**x-inconsistency (pixels)**	**y-inconsistency (pixels)**	**Overall inconsistency (%)**
1	5.092	5.155	0.23
2	1.178	0.884	0.043
3	1.184	0.917	0.043
4	1.160	0.925	0.042

Table 1: Average pixel inconsistencies of reference plane features detected across rectified images

It is clear that after the second pass, we see significant diminishing returns. For our implementation, we have therefore opted for two passes. We achieve an overall final inconsistency of 0.043%, from 0.23% with only one pass (an improvement of a factor of 5.35). Calibrated light fields can therefore be considered to be 99.957% accurate in terms of their alignment and resultant focus.

It is critical to note that the average difference in feature positions does not directly translate to reprojection error. Reprojection error is a common accuracy measure used in metric calibration procedures. Like Vaish et al's plane + parallax procedure, our procedure is non-metric, so we cannot calculate reprojection error. Vaish points out that non-metric procedures are not calculating the same intrinsic and extrinsic camera parameters, or making the same assumptions as metric calibration procedures (Vaish et al., n.d.).

5.3 Light field rendering

Light fields can be rendered by shifting rectified images or *light field slices* to a common depth, then adding the slices together to yield a single 2D output. Dansereau provides an implementation of this in his *Light Field Toolbox* for MATLAB as `LFFiltShiftSum`. Light fields will initially need be loaded via `LFReadGantryArray`, which can take a rectified image set as input.

We can demonstrate the effectiveness of our calibration procedure by attempting to render a planar scene. We juxtapose an uncalibrated light field render with its rectified counterpart (see figure 13). Note the increased clarity after rectification.

Figure 13: Uncalibrated images rendered as a light field (left) vs. light field calibrated using our procedure (right).

5.3.1 Synthetic aperture focusing and robustness to occlusion

We can demonstrate our calibration procedure's appropriateness for synthetic aperture focusing applications by adjusting light field focus for non-planar scenes. A scene demonstrating synthetic focus of three depth levels has been provided (see figure 14).

Figure 14: Three focus levels for a non-planar scene. Focus on backdrop (left), focus on person (middle), focus on fist holding ruler (right).

An application of particular interest in plenoptic imaging is occlusion removal. A basic form of occlusion removal can be demonstrated by synthetic aperture focusing. An example illustrating robustness to occlusion has been provided (see figure 15 and figure 16).

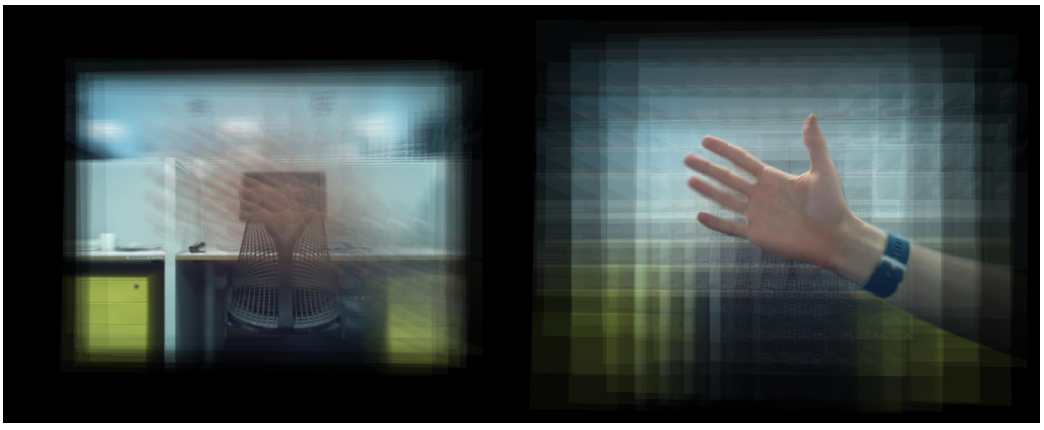


Figure 15: Two focus levels of a non-planar scene with a significantly occluding hand. Focus on occluded area containing office chair and computer monitor (left) and focus on hand (right).

Figure 16: Rectified image set used to construct the rendered light fields in figure 15. Note that in each camera view, a significant portion of the monitor or the office chair is occluded.

5.3.2 Light field video

Light field video is an area of particular interest, as little work has been done in plenoptic imaging that exploits the temporal axis. We have been able to construct light field video by extracting each frame of video, and constructing a light field for each frame.

The Raspberry Pi camera array has been able to capture synchronised video with < 1 centisecond of synchronisation disparity. This has been achieved by recording a stopwatch across all cameras and commanding video capture by sending synchronised SSH (see figure 17).

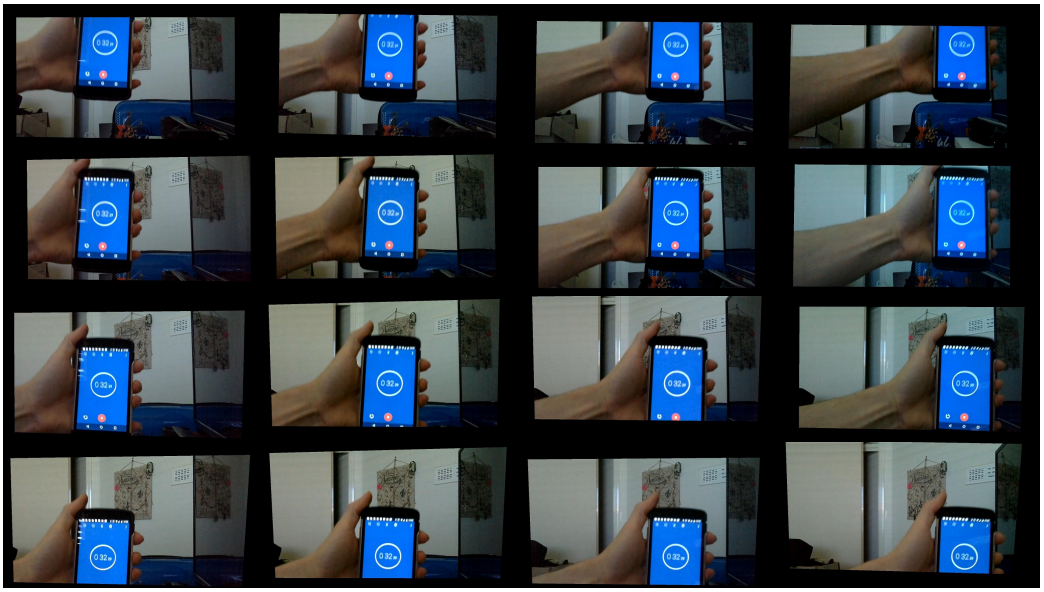


Figure 17: Recorded video has been shown to be synchronised to less than one centisecond. The camera modules exhibit significant motion blur over smaller timespans due to the shutter speed.

6 Challenges and lessons learned

Several significant challenges were faced throughout the project, which we reflect on in this section as a reference for potential future work on the Raspberry Pi camera array.

6.1 Camera array power setup

A significant challenge which presented itself early in the project was the camera array's power setup. Initially, not all Raspberry Pi devices were powering on. This meant that at the start of the project, the cause of the failing Raspberry Pis needed to be identified and fixed. The power setup and lack of cable management meant that identifying the cause was a significant problem itself (see figure 18). A failing Raspberry Pi was also noted in Denham's technical report (Denham, 2015).

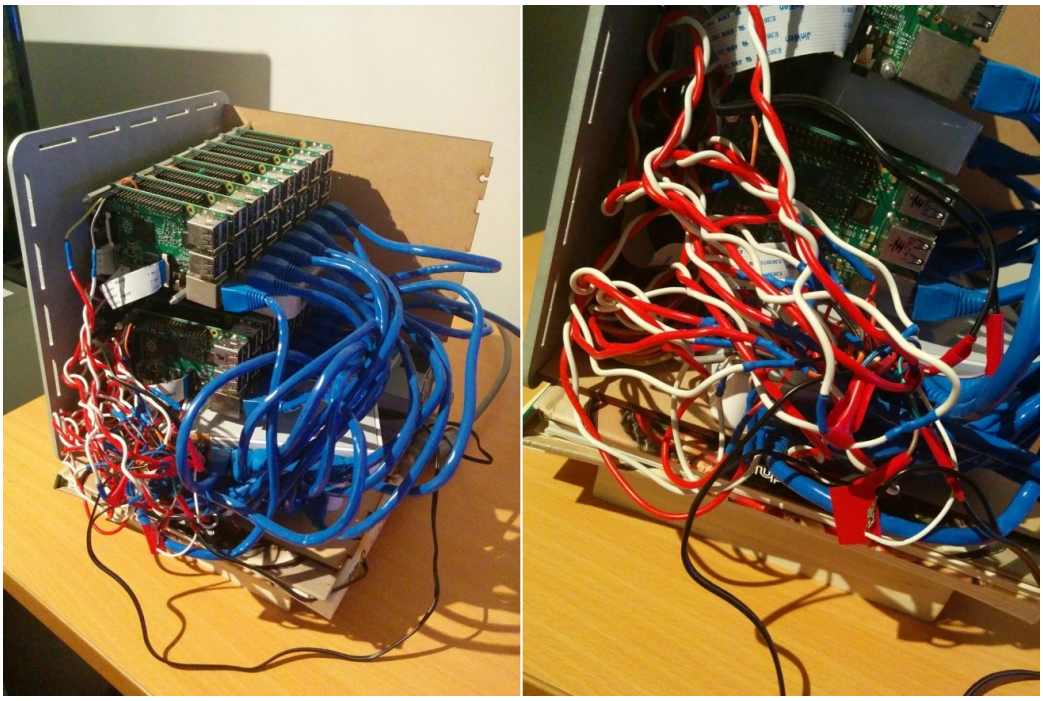


Figure 18: Rasperry Pi camera array power setup. Notice the poor cable management.

Solving the problem required the camera array to be pulled apart and reconstructed, ensuring all cables were plugged in securely. Unfortunately, the power setup was also extremely temperamental - power cables frequently came loose, and often required resoldering. As soldering was a new skill, training and practice was required. Even after all Raspberry Pis were powering on, they would often lose power after simply moving the camera array, which would then require reconstruction or resoldering.

Once all cameras were powering on and had the right software, such issues were relatively infrequent. For future work, it may prove beneficial to redesign the power setup of the array, so that cables are organised, safe and secure.

6.2 Camera array hardware setup

The way Raspberry Pis are positioned within the enclosure makes it difficult to access SD cards, power cables and cameraboard cables (see figure 19). If SD cards need re-imaging, or power or cameraboard cables need resitting, the entire array may need to be pulled apart.

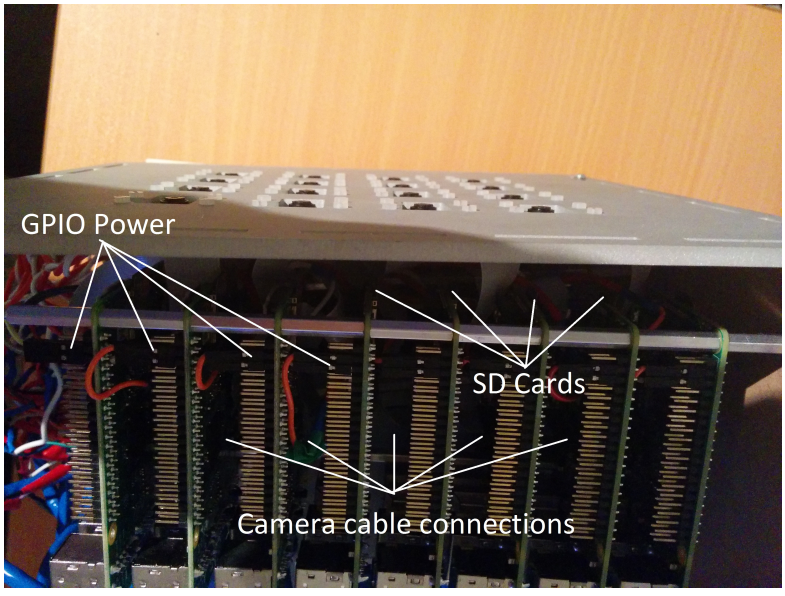


Figure 19: Top-down view of the camera array enclosure with the top removed. Key areas requiring regular access are labelled. It is difficult to access these areas without deconstructing the array of Raspberry Pis.

Again, once the camera array has all the necessary software and all cables are secure, these issues do not cause serious problems. Nevertheless, we highly recommend redesigning the enclosure so that Raspberry Pis are not situated as they are presently.

6.3 Raspberry Pi software setup

In the early stages of the project, we wanted to demonstrate camera function via a live video stream to each Raspberry Pi camera module. It was discovered that the MATLAB support package for Raspberry Pi devices could achieve this. However, this requires Raspberry Pis to be set up with MATLAB's Raspbian distribution, since MathWorks does not provide the streaming software on its own. Setting up the new OS was a tedious process, because it required that the entire camera array be pulled apart so that SD cards could be retrieved, imaged, and put back into place.

After successfully testing camera functionality via live streams through MATLAB, our aim was to capture synchronised images and video. In Denham's original report, CompoundPi is suggested for synchronised capture. Installing the CompoundPi software package presented another challenge, since software needed to be downloaded and configured across Raspberry Pi devices. Since only one device can be connected to WiFi at a time (we only have one WiFi dongle), we attempted to install and configure CompoundPi on one device to start with. At this point, it was discovered that CompoundPi was incompatible with MATLAB's Raspbian distribution, so we decided to revert to a plain Raspbian distribution. Once we had one Raspberry Pi working with pure Raspbian and CompoundPi, we pulled apart the array to copy the image to the other 15 devices and reconstruct the array.

Later, we discovered that CompoundPi does not support synchronised video well. After some exploration, we found that the *SuperPutty* software package for Windows could synchronise commands to many devices via SSH. This could effectively replace CompoundPi so that we can capture synchronised images and video using the default capture commands `raspistill` and `raspivid`, without requiring any third-party software. This is our current software recommendation.

7 Conclusions

We have developed a non-metric calibration for light field acquisition, suitable for camera arrays, and flexible to viewing direction and planarity. The procedure is conceptually simple, yet demonstrates excellent results for light field applications, provided transform estimates are sufficiently accurate. Synthetic aperture focusing results compare well with other such setups (see figure 20).

Figure 20: Vaish et al's synthetic aperture focusing demonstration (top) and ours (bottom). The top images were adapted from Vaish, V., Wilburn, B., Joshi, N., & Levoy, M. (n.d.). Using plane + parallax for calibrating dense camera arrays. In *Computer vision and pattern recognition, 2004. cvpr 2004. proceedings of the 2004 ieee computer society conference on* (Vol. 1, pp. I–2).

It is important to recognise that the calibration procedure is non-metric, thus there is no inherent baseline for a correct result or reprojection error. However, our calibration is also unique in that it is a non-metric calibration whose accuracy can be easily calculated to a fine degree, by measuring positional consistency of features on the reference plane. We have achieved such a positional consistency of 99.957%.

The Raspberry Pi camera array is now significantly more capable as a light field camera, since camera poses are now retained by a new aluminium front-plate. Additionally, the camera array has demonstrated good performance with the new calibration procedure and synthetic aperture focusing applications. The camera array's potential for explorations in light field video has also been demonstrated, as synchronised capture and rendering of light field video has been achieved.

8 Future work

For any future work with the Raspberry Pi camera array, we highly recommend improvements be made in the design of the device before additional light field work takes place. Specifically, the power method and/or setup should be adjusted, so that power cables are secure, relatively accessible, and less exposed. Additionally, the enclosure should be redesigned so that SD cards, cameraboard ribbon cables, and power cables are accessible without pulling apart the device.

The camera array has successfully demonstrated effectiveness in light field applications such as synthetic aperture focusing and light field video, and is therefore fit for future work in plenoptic imaging. An area of particular interest is occlusion removal that exploits the temporal axis.

The nature of the Raspberry Pi camera array also means that there may be potential for parallel processing. Perhaps using OpenCV across devices, images could be rectified before being sent to a host machine for final processing into a light field. If the current calibration procedure is still used, it too could be implemented using OpenCV and run on each Raspberry Pi. This would allow the device to self-calibrate without the need for a host machine.

References

Bay, H., Tuytelaars, T., & Van Gool, L. (2006). Surf: Speeded up robust features. In *European conference on computer vision* (pp. 404–417).

Dansereau, D. G. (2014). *Plenoptic signal processing for robust vision in field robotics* (Unpublished doctoral dissertation). The University of Sydney.

Denham, R. (2015, June). *Light field array camera* (Tech. Rep.). Queensland University of Technology.

Torr, P. H., & Zisserman, A. (2000). Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, *78*(1), 138–156.

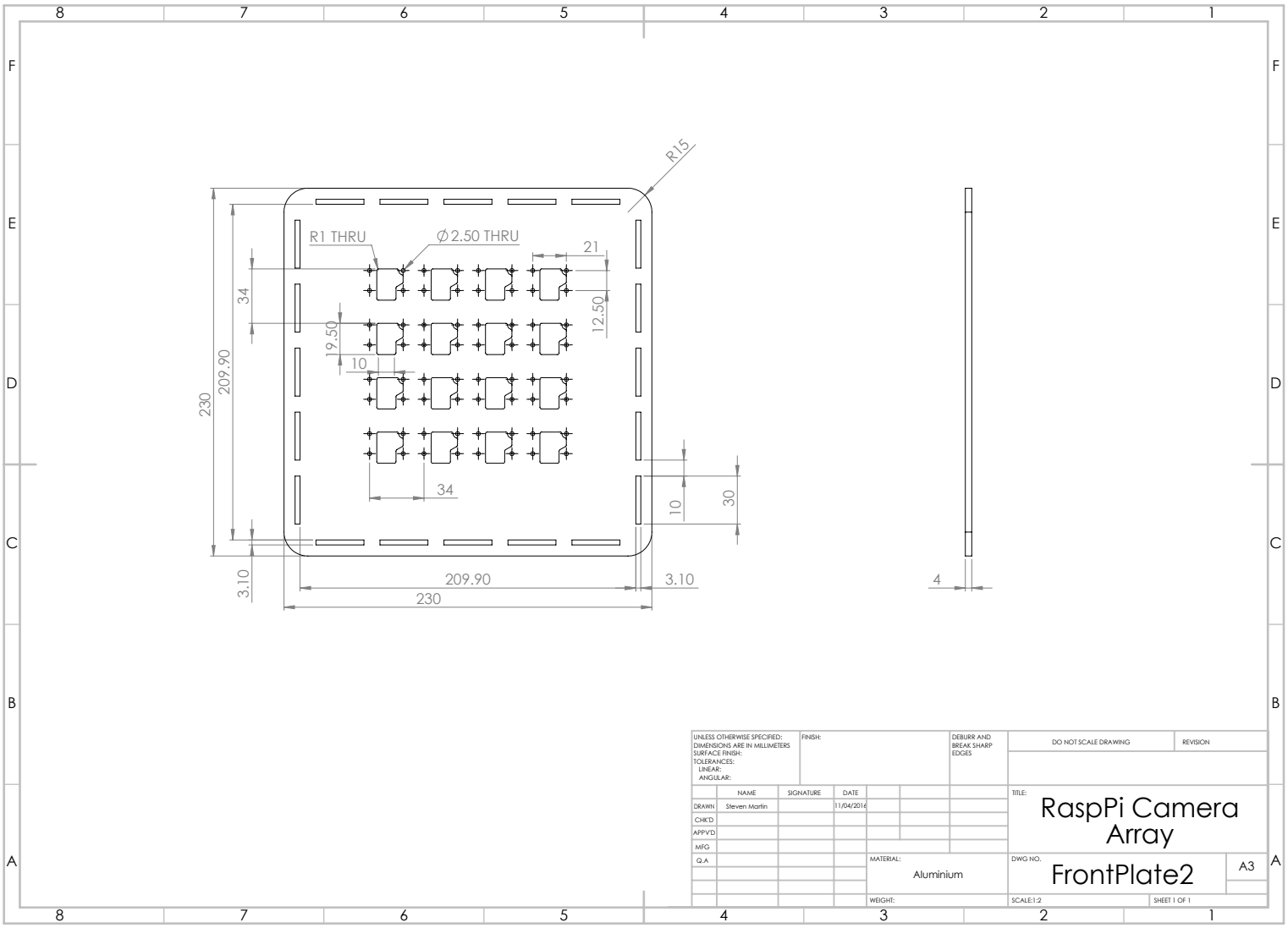
Vaish, V., Wilburn, B., Joshi, N., & Levoy, M. (n.d.). Using plane + parallax for calibrating dense camera arrays. In *Computer vision and pattern recognition, 2004. cvpr 2004. proceedings of the 2004 ieee computer society conference on* (Vol. 1, pp. I–2).

Xu, Y., Maeno, K., Nagahara, H., & Taniguchi, R.-i. (2014). Mobile camera array calibration for light field acquisition. *arXiv preprint arXiv:1407.4206*.

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

9 Appendices

9.1 Replacement front plate



9.2 Project Proposal

Removing occlusions from light field data
Research project proposal

Ashley Stewart

October 28, 2016

9.3 Literature Review

Removing occlusions from light field data
Literature Review

Ashley Stewart

October 28, 2016

9.4 MATLAB Code

9.4.1 BuildTransformMatrixFromCalibrationImages

```
%%
% This file is an adaptation of MATLAB's 'Feature Based
% Panoramic Image Stitching' demonstration.
%
% Ashley Stewart
% Queensland University of Technology
% Australia
% 06/11/2016

%%
% BuildTransformMatrixFromCalibrationImages(calibrationSetDir)
%   calibrationSetDir:
%           The directory containing the calibration set
%
% Returns a series of geometric transformations that describe
% the relative transforms of images in a calibration set
% captured by a camera array, according to Ashley Stewart's
% panoramic calibration procedure for light field acquisition.
%
% The transformation matrix returned can be used to rectify
% future image sets captured by the same camera array.
%
% The transformation matrix will contain transforms in
% alphabetical order according to the calibration set
% filenames.

function tforms = BuildTransformMatrixFromCalibrationImages...
    (calibrationSetDir)

% Load calibration images
images = imageSet(calibrationSetDir);

% Initialise the first image and detect features
I = read(images, 1);
grayImage = rgb2gray(I);
points = detectSURFFeatures(grayImage);
```

```
[features, points] = extractFeatures(grayImage, points);

% Initialise transforms to the identity matrix
tforms(images.Count) = projective2d(eye(3));

% Iterate over remaining image pairs
for n = 2:images.Count

    % Store points and features for I(n-1)
    pointsPrevious = points;
    featuresPrevious = features;

    % Read I(n)
    I = read(images, n);

    % Detect and extract SURF features for I(n)
    grayImage = rgb2gray(I);
    points = detectSURFFeatures(grayImage);
    [features, points] = extractFeatures(grayImage, points);

    % Find correspondences between I(n) and I(n-1)
    indexPairs = matchFeatures(features, ...,
        featuresPrevious, 'Unique', true);
    matchedPoints = points(indexPairs(:,1), :);
    matchedPointsPrev = pointsPrevious(indexPairs(:,2), :);

    % Estimate the transformation between I(n) and I(n-1)
    tforms(n) = estimateGeometricTransform(matchedPoints,...
        matchedPointsPrev, 'projective',...
        'Confidence', 99.9, 'MaxNumTrials', 2000);

    % Compute T(1) * ... * T(n-1) * T(n)
    tforms(n).T = tforms(n-1).T * tforms(n).T;
end

%%
% At this point, all the transformations in |tforms| are
% relative to the first image.
%
% Using the first image as the initial reference does not
% produce the best result because it tends to distort most of
```

```
% the images. An improved result can be achieved by modifying
% the transformations such that the center of the scene is the
% least distorted. This is accomplished by inverting the
% transform for the center image and applying that transform to
% all the others.
%
% Start by using the |projective2d| |outputLimits| method to
% find the output limits for each transform. The output limits
% are then used to automatically find the image that is roughly
% in the center of the scene.
imageSize = size(I);  % all the images are the same size

% Compute the output limits  for each transform
for i = 1:numel(tforms)
    [xlim(i,:), ylim(i,:)] = outputLimits(tforms(i), ...
        [1 imageSize(2)], [1 imageSize(1)]);
end

%%
% Next, compute the average X limits for each transforms and
% find the image that is in the center. Only the X limits are
% used here because the scene is known to be horizontal. If
% another set of images are used, both the X and Y limits may
% need to be used to find the center image.

avgXLim = mean(xlim, 2);
[~, idx] = sort(avgXLim);
centerIdx = floor((numel(tforms)+1)/2);
centerImageIdx = idx(centerIdx);

%%
% Finally, apply the center image's inverse transform to all
% the others.
Tinv = invert(tforms(centerImageIdx));

for i = 1:numel(tforms)
    tforms(i).T = Tinv.T * tforms(i).T;
end

end
```

9.4.2 RectifyImagesViaTransforms

```
%%
% Ashley Stewart
% Queensland University of Technology
% Australia
% 06/11/2016

%%
% RectifyImagesViaTransforms(tforms, originalsDir, rectifiedDir)
%   tforms:
%       A transformation set built by
%       BuildTransformMatrixFromCalibrationImages
%   originalsDir:
%       The directory of the original images
%   rectifiedDir:
%       The directory to save rectified images to
%
% Rectifies an image set according to a series of
% transformations determined by analysing a calibration set
% via BuiltTransformMatrixFromCalibrationImages.
%
% Rectified image sets can be loaded via LFReadGantryArray.

function RectifyImagesViaTransforms(tforms, originalsDir, ...
    rectifiedDir)

% Changable parameters
scaleFactor = 0.33;
outputFormat = 'jpg';

% Load images
imageDir = fullfile(originalsDir);
images = imageSet(imageDir);

% Find the minimum and maximum output limits
imageSize = size(read(images, 1));
for i = 1:numel(tforms)
    [xlim(i,:), ylim(i,:)] = outputLimits(tforms(i), ...
        [1 imageSize(2)], [1 imageSize(1)]);
```

```
end

xMin = min([1; xlim(:)]);
xMax = max([imageSize(2); xlim(:)]);

yMin = min([1; ylim(:)]);
yMax = max([imageSize(1); ylim(:)]);

% Final dimensions of each image
width  = round(xMax - xMin);
height = round(yMax - yMin);
imageDimensions = [height width];

% Calculate how many digits there are in the number of images
digits = numel(num2str(images.Count));

for imageNum = 1:images.Count
    I = read(images, imageNum);

    panoramaView = imref2d(imageDimensions, [xMin xMax], ...
        [yMin yMax]);
    warpedImage = imresize(imwarp(I, tforms(imageNum),...
        'OutputView', panoramaView), scaleFactor);

    imwrite(warpedImage,strcat(rectifiedDir, '/',...
        sprintf(strcat('%0', num2str(digits), 'd'),...
        imageNum), '.', outputFormat));
end

end
```

9.4.3 CalculateRectifiedSetAccuracy

```
%%
% This file is an adaptation of MATLAB's 'Feature Based
% Panoramic Image Stitching' demonstration.
%
% Ashley Stewart
% Queensland University of Technology
% Australia
% 06/11/2016

%%
% CalculateRectifiedSetError(rectifiedSetDir)
%   rectifiedSetDir:
%           The directory containing the rectified set
%
% Returns the average pixel error calculated in the
% x and y directions for a rectified image set.
function [avgPixelError, pixelError] = ...
    CalculateRectifiedSetAccuracy(rectifiedSetDir)

% Load rectified images
images = imageSet(rectifiedSetDir);

% Initialise the first image and detect features
I = read(images, 1);
grayImage = rgb2gray(I);

points = detectSURFFeatures(grayImage);
[features, points] = extractFeatures(grayImage, points);

% Pixel error
pixelError = {};

% The pixel error beyong which we remove as an outlier
outlierError = 50;

% Iterate over remaining image pairs
for n = 2:images.Count
```

```
    % Store points and features for I(n-1)
    pointsPrevious = points;
    featuresPrevious = features;

    % Read I(n)
    I = read(images, n);

    % Detect and extract SURF features for I(n)
    grayImage = rgb2gray(I);
    points = detectSURFFeatures(grayImage);
    [features, points] = extractFeatures(grayImage, points);

    % Find correspondences between I(n) and I(n-1)
    indexPairs = matchFeatures(features, ...,
        featuresPrevious, 'Unique', true);
    matchedPoints = points(indexPairs(:,1), :);
    matchedPointsPrev = pointsPrevious(indexPairs(:,2), :);

    % Calculate pixel error
    thisError = abs(matchedPoints.Location - ...
        matchedPointsPrev.Location);

    % Remove outliers
    thisError(thisError(:,1) > outlierError,:) = [];
    thisError(thisError(:,2) > outlierError,:) = [];

    pixelError{n-1} = thisError;
end

% Find the average pixel error in x and y directions
avgPixelError = mean(vertcat(pixelError{:}));

end
```