# Surface anomaly detection on dirt trails

By Ken Wolf,
Computer Vision course
School of Computer Science
Carnegie Mellon University
December 3, 2023

## Introduction

The ideas for this paper came from my passion for running. My favorite runs are on dirt trails, usually in parks. When running on trails, it is clear that the detection of surface anomalies informs foot placement decisions. A general goal when trail running is to avoid surface anomalies, such as acorns, roots, sticks, rocks, divots, and buildup. Stepping on surface anomalies can reduce landing/push-off surface area, stability, traction, and momentum. All of which can contribute to disrupted running cadence and injuries from twisted ankles and loss of balance.

One reason that I am so interested in the topic of surface anomaly detection is that avoiding running injuries is very important to me. Since running provides me with enjoyment, stress relief, and physical health, my ability to run is an important part of my quality of life.

Another reason I am so interested in the topic of surface anomaly detection is that I have been fascinated for a long time, with my body's ability to subconsciously, repeatedly, and quickly, make complicated split-second foot placement decisions. I have realized that looking at the trail (a few meters in front of my location of course) as I run, significantly improves my foot placement decisions, even if I am consciously thinking about a completely different topic. I do not need to consciously observe the trail – I can completely 'space out', but somehow the visual input is gathered and used for foot placement decisions. This is very intriguing.

This paper describes detecting surface anomalies from photographs of a flat dry section of a dirt trail using feature detection (class module 3) and two-view geometry (class module 8) computer vision techniques. The intent is to use pairs of photographs to build a three-dimensional model of the trail, and then identify surface anomalies from model points furthest from the z-value-mean.

# Method Steps

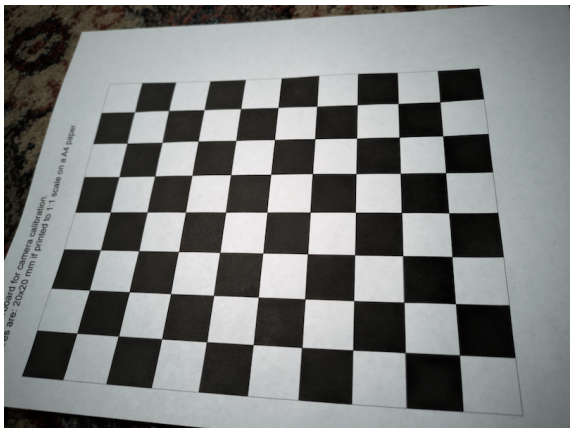
Step 1 - Derive ‘Intrinsic’ Matrix
Step 2 - Capture Images
Step 3 – Load and Prepare Images
Step 4 – Detect Features
Step 5 – Match Key Points
Step 6 – Refine Key Point Matches
Step 7 – Derive ‘Fundamental’ Matrix
Step 8 – Derive ‘'Essential' Matrix
Step 9 - Derive ‘Extrinsic’ Matrix
Step 10 - Derive ‘Camera Projection’ Matrices
Step 11 – Derive 3D Model using Triangulation
Step 12 - 3D Model Analysis

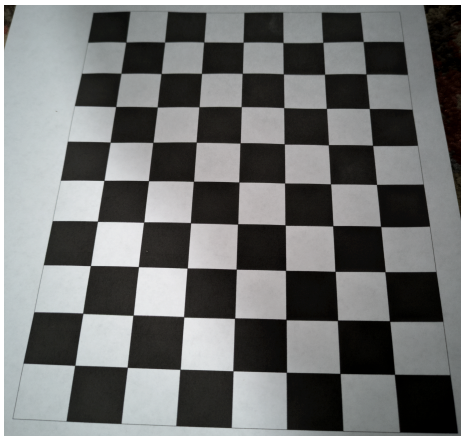
# Step 1 - Derive ‘Intrinsic’ Matrix

The camera’s ‘Instrinsic’ matrix is derived using a process of ‘Camera Calibration’.

Camera calibration process included use of an eight by ten checkerboard pattern printed from the URL https://www.mrpt.org/downloads/camera-calibration-checker-board_9x7.pdf

Images used for calibration included these …





The OpenCV v4.8 functions findChessboardCorners, cornerSubPix, and calibrateCamera are used.

One camera was used to gather images and it’s determined intrinsic matrix, K, is …

| | | |
|---|---|---|
| 1073.87531 | 0 | 487.802829 |
| 0 | 1102.79033 | 551.208199 |

| | | |
|---|---|---|
| 0 | 0 | 1 |

K can also be described as

| | | |
|---|---|---|
| fx | s | cx |
| 0 | fy | cy |
| 0 | 0 | 1 |

The focal length, in pixels, is …
(fx,fy) = (1073.87531, 1102.79033) ~= (1074,1103)

The pixel coordinates of the principal point (where all rays converge) is …
(cx,cy) = (487.802829, 551.208199) ~= (488, 551)

The skew factor, s, is zero. "The skew parameter will be zero for most normal cameras." - *Multiple View Geometry in Computer Vision by Richard Hartley and Andrew Zisserman (as quoted on https://forum.opencv.org/t/about-the-camera-principal-axis-skew-factor-s/7773/2 )*

# Step 2 - Capture Images

## Third-Party Data

A suitable existing dataset was not identified.
Images in https://zenodo.org/records/7755648 are too low resolution and path itself is a small and varied percent of the images.
Images in kaggle.com/eyantraiit/semantic-segmentation-datasets-of-indian-roads include a footpath but it is too clean and groomed to be useful for anomaly detection.

## Data Capture

Five pairs of images were captured, using a Pixel7a phone camera, on Meadows Canyon Trail, in Tilden Regional Park, in Berkeley, California.

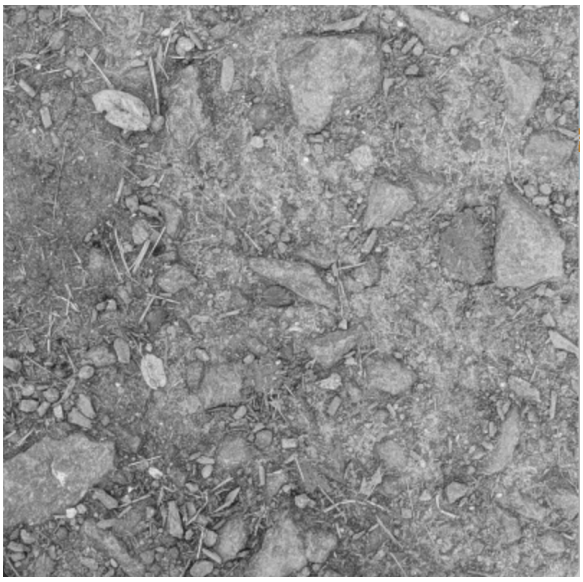
The images were saved as raw .dng files. Each file was approximately 24mb. Images were transferred from phone to computer via email as attachments.

Each scene was photographed twice. The camera was 3 feet above the trail for both pictures. The photographer took a picture, took a 3-foot step to the right, and then took another picture.

For the photographer, an L-square ruler was used to orient the images along with the Pixel7a camera grid lines. Pixel7a camera grid lines are visible to the photographer but do not appear in captured images. The L-square ruler was cropped out of the images before they computer vision techniques were applied to the images.

## Step 3 – Load and Prepare Images

Images of an original photograph and the used cropped and rotated version …



The L-square ruler was cropped out of the images before they computer vision techniques were applied to the images.

Another motivation for cropping was to reduce the size of the image to improve processing time and make increase the relative size of surface anomalies.

The OpenCV (Open Source Computer Vision Library) v4.8 function imread is used to load the color images as greyscale, and for convenience, the image rotation is done using the OpenCV v4.8 function rotate

## Step 4 – Detect Features

Feature descriptors are identified using the Scale Invariant Feature Transform (SIFT) method. We learned about SIFT and how it works in Module 3 of this course. The OpenCV (Open Source Computer Vision Library) v4.8 function SIFT_create is used to create a SIFT class instance. Then the SIFT class's inherited method detectAndCompute is invoked twice - once for each image in a pair of images - to determine and return feature descriptors.

## Step 5 – Match Key Points

Matches between key points on the respective images are determined using the OpenCV (Open Source Computer Vision Library) v4.8 class BFMatcher . We use this class's inherited method knnMatch to determine matched pairs of points by processing the two image's respective feature descriptors. This class is a "Brute-force descriptor matcher. For each descriptor in the first set, this matcher finds the closest descriptor in the second set by trying each one." - https://docs.opencv.org/4.8.0/d3/da1/classcv_1_1BFMatcher.html

The BFMatcher can be configured to run with different 'norm types' which is to say to use different criteria for determining individual matches. The default is NORM_L2 . The Euclidean norm (L2 norm) refers to the square root of the sum of squares, which can be used to determine distance.

## Step 6 – Refine Key Point Matches

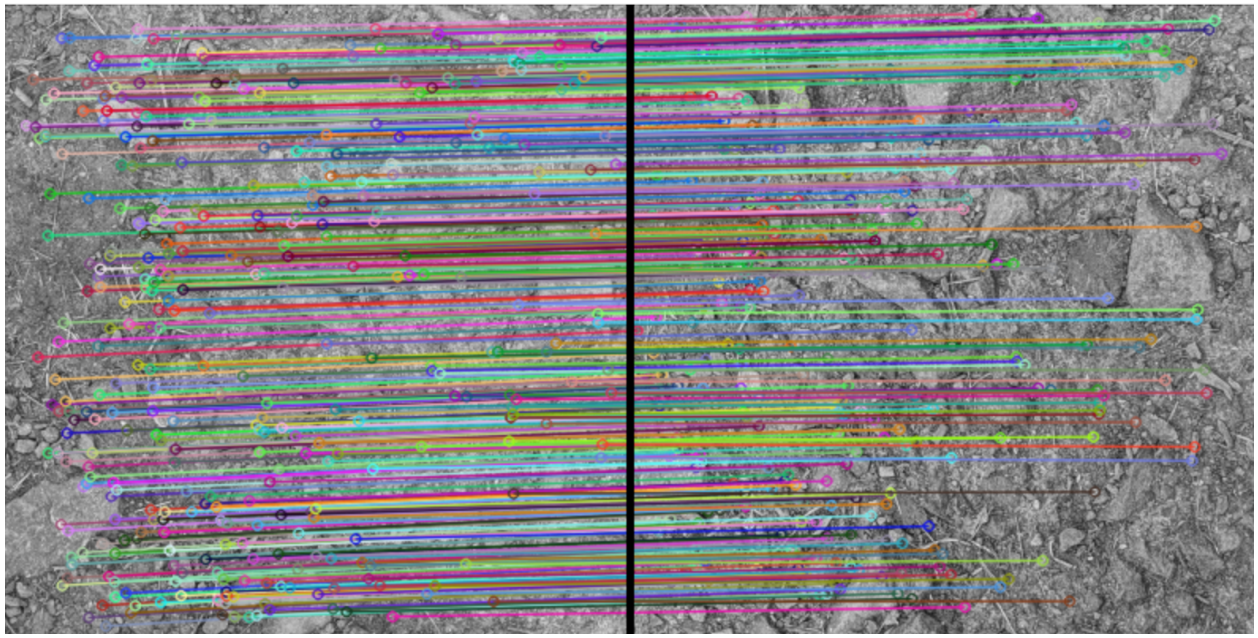
Then we use a ratio test to identify the 'good' matches. The determination uses the distance attribute of instance of the OpenCV (Open Source Computer Vision Library) v4.8 class DMatch. This distance attribute values will partially depend on the 'norm types' used by the BFMatcher, but regardless of that they still represents the same concept. So, for each matched set of points, we compare their distance. When the first image's distance is a smaller percent of the second image's distance then the match is 'better'.

For this paper, multiple ratio constants were tested. With the visualizations below, it is easy when there are too few lines, as well as to see when there are too many because valid matched lines are at least generally parallel to the other match lines.

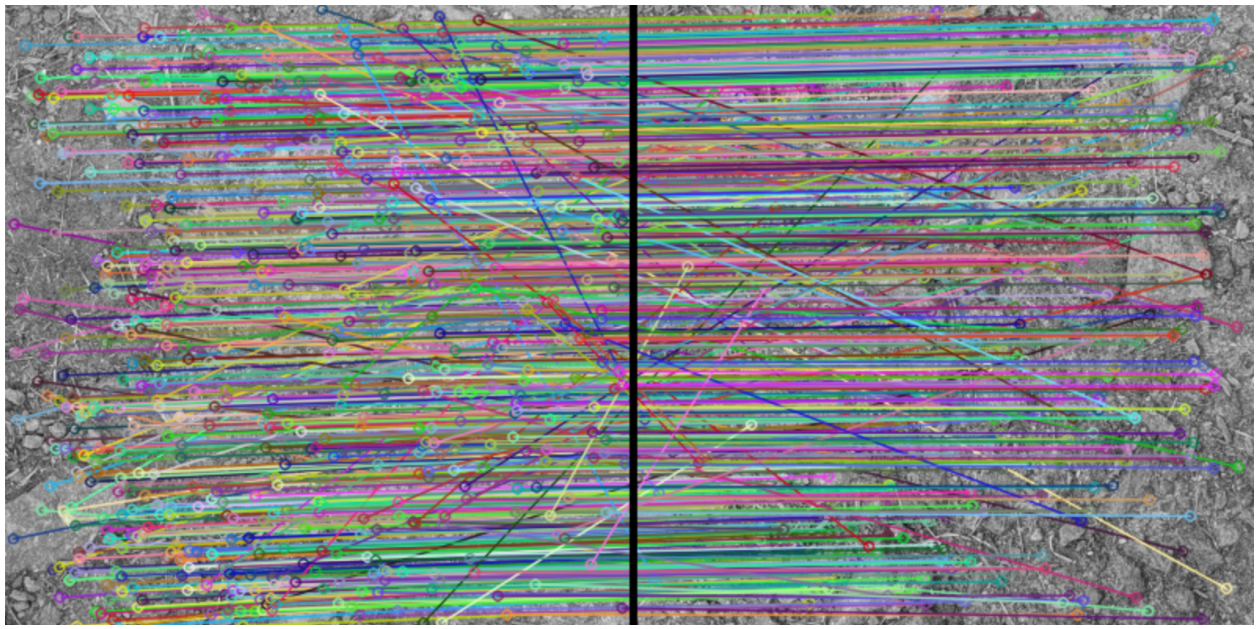
With m and n being DMatch class instances from the respective photographs in a matched pair, when (m.distance < Q * n.distance) is true, a match is considered to be a good match. Here is the result of some experiments with various Q values for one of pairs of images…

| Q Value | visualization includes match point lines that are not parallel. | % of matches that are identified as good matches |
|---|---|---|
| 0.2 | none | 0% |
| 0.3 | none | 0.4% |
| 0.4 | none | 1.8% |
| 0.5 | none | 6.5% |
| 0.6 | none | 11.7% |
| 0.7 | none | 17.6% |
| 0.8 | A few | 23% |
| 0.9 | A significant number | 33% |
| 1.0 and above | many | 100% |

Visualization of good match lines using a multiplier value of 0.7:



Visualization of good match lines using a multiplier value of 0.9, with many lines that are neither horizontal nor parallel with most of the other lines:



# Step 7 - Derive 'Fundamental' Matrix

## Summary

The 'Fundamental' matrix, F, is determined from the good matched points.

The 8-point algorithm and RANSAC (Random Sample Consensus) algorithm, two common ways to determine F, were both considered.

The 8-point algorithm was selected for reasons discussed below.

## 8-point algorithm

The 8-point algorithm is a reasonable option because it is computationally efficient, there are more than 8 vectors of data, and because the good match data is clean with few outliers and little noise, having already distilled the matched points down to the set of good matched points.

For the 8-point algorithm code was used from this course's programming assignment #4 from class module 8.

This 8-point algorithm code uses the SciPy (an open-source software for mathematics, science, and engineering) v1.11.3 function optimize.fmin_powell to minimize F, a function of multiple variables, using Powell's method. This is

done after the good matched points are normalized, A has been determined, and SVD (Singular Value Decomposition) of A is used to determine the values in F. It is done before F is 'singularized' ( using SVD on F ( U, S, V = np.linalg.svd(F) ) and then recomposing F ( S[-1] = 0 and F = U.dot(np.diag(S).dot(V)) ) and denormalized.

## RANSAC algorithm

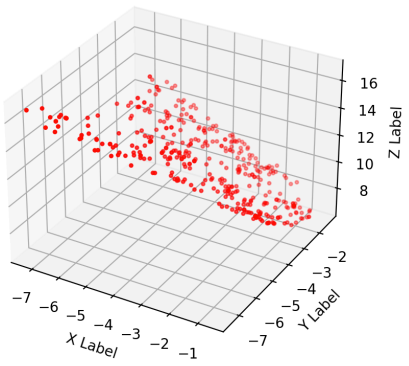
The RANSAC may also be a reasonable choice because it is iterative, so regardless of data quality, it could, in theory, drill down to a reasonable F value.

For use of the RANSAC algorithm, the code that was provided in this course's programming assignment #4 from class module 8 was used initially, but it was generating errors in nested 3rd party library code in the scikit-image (open source python image processing library) v0.20.0 ransac function. It is worth noting that an error from that ransac function, when it was run with a residual_threshold parameter of 0.3, was that it needed 8 points, which implies that the 8-point algorithm may be used internally, possibly as an estimator. So, after spending significant time on the ransac function errors, I gave up on that code and, instead, used the OpenCV (Open Source Computer Vision Library) v4.8 function findFundamentalMat.

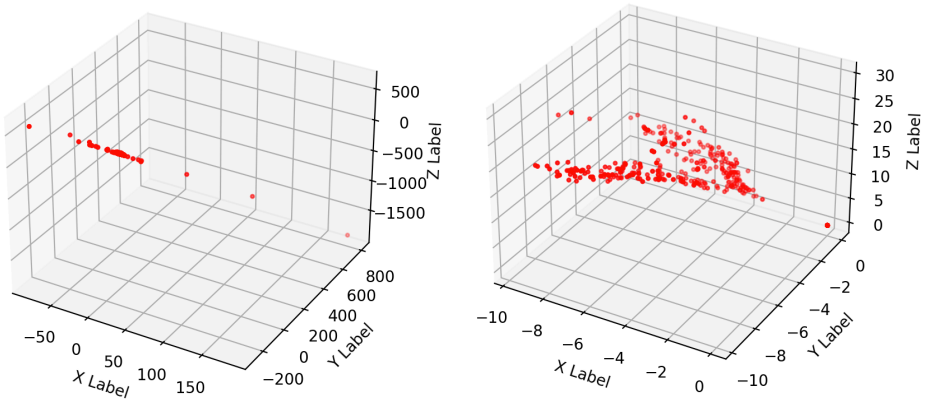
## Comparison

The two algorithms returned significantly different values for F, with the RANSAC data consisting of significant outliers, as shown in the following three example graphs of resulting triangulated data from one pair of photos:

8-point Algorithm




RANSAC, before and after trimming 3D data outliers (~20% of the data)




As a result of this consideration of the two algorithms, the 8-point algorithm was used because it did not require discarding of a significant portion of the data.

## Step 8 - Derive ‘Essential’ Matrix

The 'Essential' matrix, E, is determined using the formula $E = K1^T \times F \times K2$. K is the intrinsic matrix for a camera, that includes information about the camera optical point, focal distance, and skew. One camera was used for this paper so K = K1 = K2. K was determined by the camera calibration. So, for this paper, E is calculated using the formula $E = K^T \times F \times K$

## Step 9 - Derive ‘Extrinsic’ Matrix

The Extrinsic’ matrix, M2, is determined using the M2 = [R|T]

The camera two’s relative Rotation matrix and Translation matrix are determined from the 'Essential' matrix E, the camera two’s intrinsic matrix K, and the goods matches, by the OpenCV (Open Source Computer Vision Library) v4.8 function recoverPose . For this paper it was configured to use the RANSAC technique.

I used an OpenCV function for determining the ‘Extrinsic’ matrix M2 because the code from the Unit 8 programming assignment #4 would intermittently suggest a set of M2s from which none of the individual member M2 would satisfy the criteria for the ‘best’ option. I spent some time on this but at some point I needed to move on so I used the third party option.

## Step 10 - Derive ‘Camera Projection’ Matrices

The camera projection matrices are determined using the equations

M1 = [I|0] = [[1, 0, 0, 0],[0, 1, 0, 0],[0, 0, 1, 0]].

C1 = K x M1
C2 = K x M2

Here is a quick review of how we derived the camera projection matrices. In this summary, the good matching points are referred to as MP… First, we determined MP and F respectively from the images and K independently. Second, we determined E from K and F. Third, we determined R and T from MP, K, and E. Fourth, we determined M2 from R and T. Fifth, we determined C2 from K and M2. This approach is based on what we learned in Unit 8 of this course.

## Step 11 – Derive 3D Model using Triangulation

For this paper, for the sake of comparison, the 2D points were triangulated twice, once by code from this course programming assignment #4 from class module 8 and once by OpenCV (Open Source Computer Vision Library) v4.8 function triangulatePoints which internally uses the DLT method.

For both approaches the inputs to the triangulation function were the good matches and the two camera projection matrices C1 and C2.

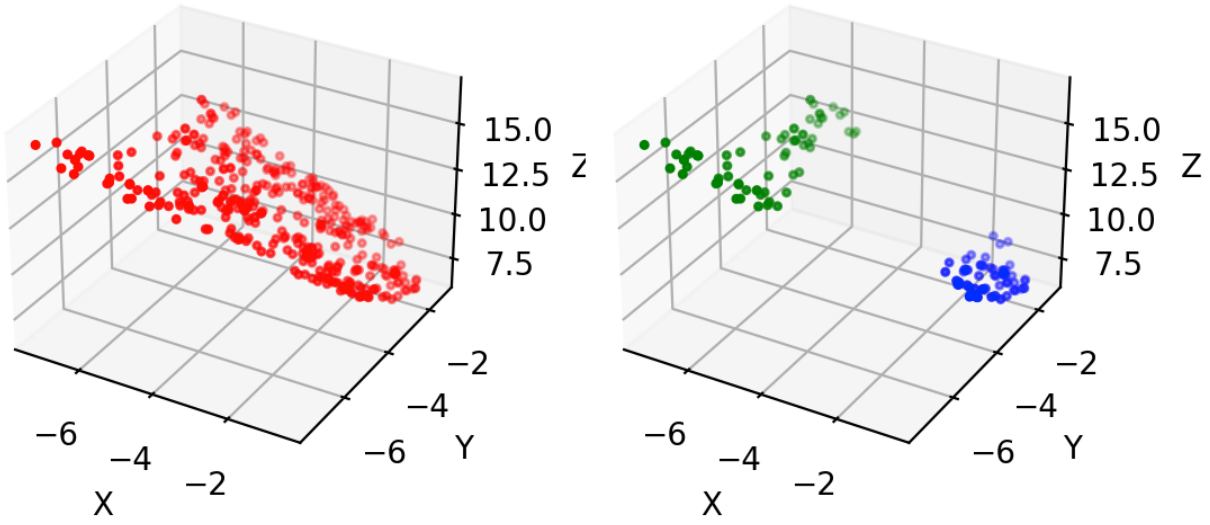
The two triangulation functions return similar results.

For example, for one pair of photographs…

Rounded to 5 decimals, 90.4% of the values where not identical.
Rounded to 4 decimals, 90.0% of the values where not identical.
Rounded to 3 decimals, 87.1% of the values where not identical.
Rounded to 2 decimals, 68.5% of the values where not identical.
Rounded to 1 decimal, 26.1% of the values where not identical.
Rounded to 0 decimals, all of the values where identical.

## Step 12 - 3D Model Analysis

Below is an example of the 3D data for a pair of photographs. All data is visualized in red. Green data points have a z value more than one standard deviation larger than the mean of z. Blue data points have a z value less than one standard deviation less than the mean of z.




Instead of creating a relatively accurate three dimensional model of the trail surface, the data seems to have simply slanted on the z-axis. It is not possible to detect anomalies from these three dimensional models.

# Conclusion

What worked well is that I was able to successfully capture images from my camera and apply a multi-step computer vision process, using Python, and that my work on the capstone project significantly increased my understanding of class modules 3 and 8.

What did not work well is that the process did not produce useful results, in that it was not possible to detect surface anomalies. Instead of creating a relatively accurate three dimensional model of the trail surface, the data seems to have simply slanted on the z-axis. This was very disappointing.

My primary finding was that a small dataset, such as just a pair of images, for a patch of trail, can be limiting in terms of the value that can be derived.

If there is one thing I would do differently it would be to gather and process a lot more data. My method seems reasonable but processing a larger set of images and joining the derived data could create a more detailed three dimensional model. Additionally a large data set would enable use of CNN to help identify surface anomalies, possibly in maps with contour lines that were derived from the three dimensional models.

# Appendix A –Tools

The following development tools and libraries were used for this paper:

- Misc
  - Python v 3.11.5
  - Jupyter Notebook v6.5.4
  - https://www.mrpt.org/downloads/camera-calibration-checker-board_9x7.pdf

- Open-Source Libraries
  - OpenCV v4.8
  - SciPy v1.11.3
  - scikit-image v0.20.0
  - Numpy v1.24.3
  - Matplotlib v 3.7.2

# Appendix B - Related Articles

- https://rosap.ntl.bts.gov/view/dot/27844
- https://people.idsia.ch/~giusti/forest/web/
- https://www.sciencedirect.com/science/article/pii/S1877042813045151
- https://academic.oup.com/tse/article/4/4/tdac026/6835624
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8494564/
- https://www.hindawi.com/journals/cin/2021/6262194/