COMPSCI 773: Assignment 2 Phase 1: Distortion Removal and Calibration

Tony Wang/Shah Heidari/Patrice Delmas

The Department of Computer Science, University of Auckland

2025

Assignment 2 Phase 1

This assignment combines the knowledge acquired in the lecture's content of weeks 1-6 and weeks 7-8. You should be able to complete A2 phase 1 part 1 from Wednesday, 30 April, and part 2 from Wednesday, 7 May. How to best prepare for this assignment:

1. Review COMPSCI 373 knowledge as listed in the COMPSCI 773 webpage.
2. Review COMPSCI 773 knowledge on transforms (rotation, translation, scaling, homography), RANSAC, and feature detection.
3. Review this week's tutorial content and the associated recordings on line fitting.
4. Review any programming provided for the above knowledge.

Part 1: Distortion Removal

Through assignment 2, Phase 1, Part 1, you will deepen your understanding of radial distortion removal. This is an essential preparation task towards camera calibration. In this assignment, you will complete the following tasks on an image acquired from Hero3+ Black edition stereo camera (H3):

- First, we will provide the corner list for an image acquired from H3 stereo cameras.
- Our goal is to ensure that straight lines in the real world remain straight in the image. Therefore, you must group the corners on the horizontal chessboard lines.
- Then, you will use the line fitting algorithm provided to find the optimal radial distortion coefficient (κ_1) that can be used to change the corner coordinates to be on straight lines.
- Once you find the best κ_1, you need to undistort the whole image.

Part 2: Camera Calibratio

In Part 2, we will deepen our understanding of the full stereo vision pipeline. You will complete the following tasks on both Fujifilm W3 camera (W3) and Hero3+ Black edition stereo camera (H3) images:

1. Calibrate a single camera system using Tsai calibration.
2. Calibrate a stereo-camera system using Tsai calibration.
3. Estimate single and stereo systems calibration errors.

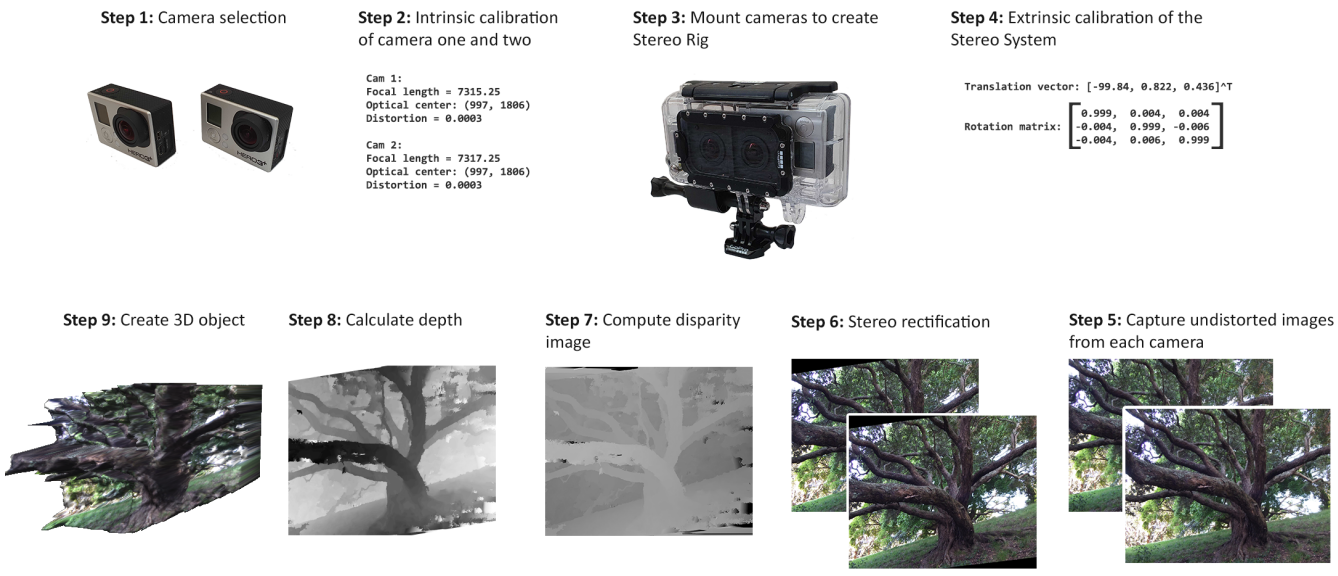


Figure: Overview of the steps in the stereo pipeline.

Assignment goals

This assignment requires that you:

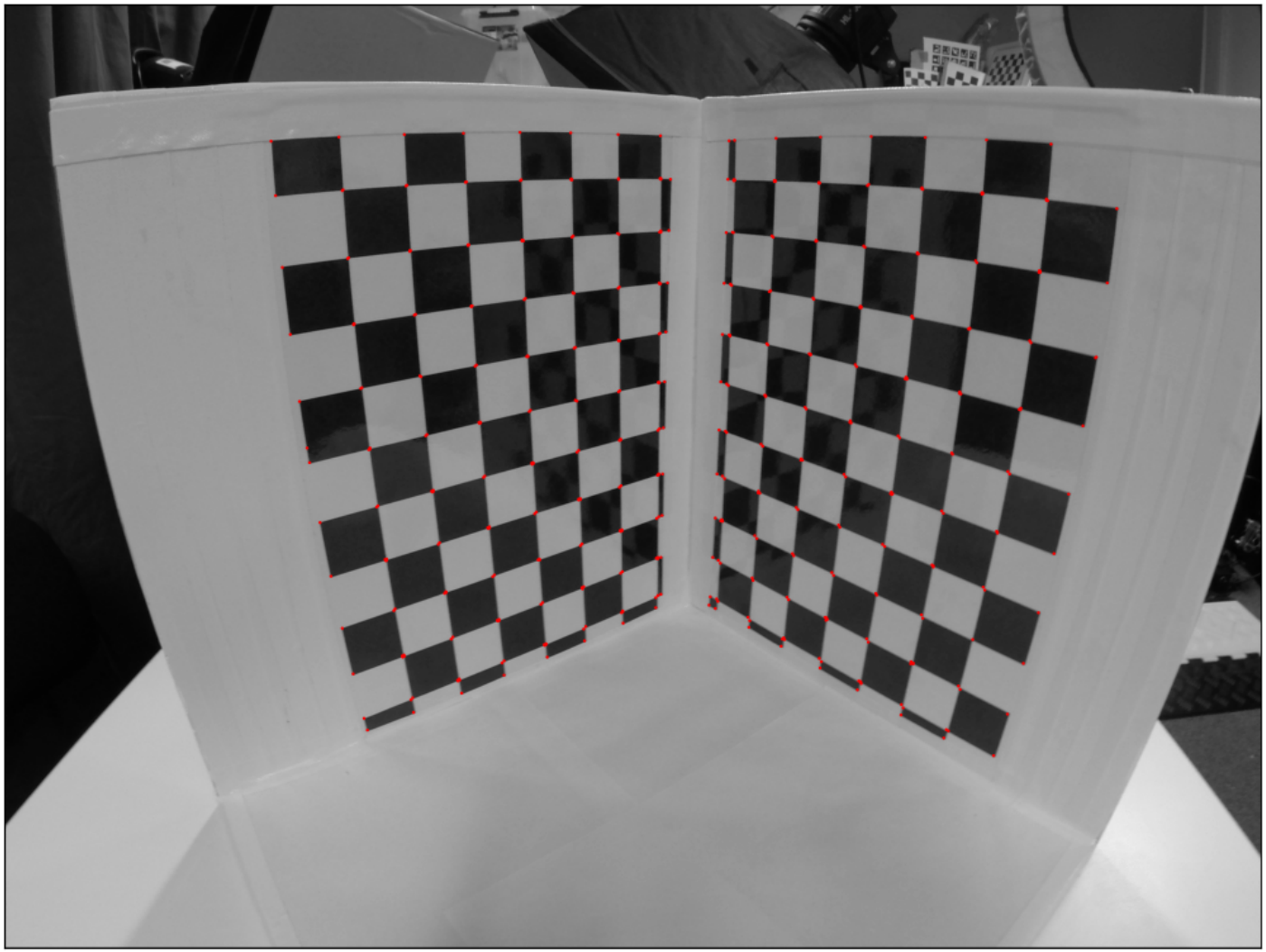
1. Adapt the Python code we provided to compute the outputs of tasks from the above slides.
2. All groups need to go through the following extension:
   - Extend your κ_1 estimation code to estimate κ_2. Compare the undistorted images, the distortion removal error (as the distance of undistorted point positions to the line they belong to).
   - Add an extension of your choice.
3. Groups of 2 require one extension (above) while groups of 3 need two extensions, for example:
   - Construct a calibration object using blob detection instead of corner detection.
   - Use another radial distortion model.
   - introduce the tangential distortion model.
   - Introduce strategies to improve your initial camera calibration results by reducing calibration errors.

Assignment goals (Cont.)

4. Write a short report detailing your assignment implementation (including extensions), results and analysis:
   - Outline your radial distortion removal implementation.
   - The optimal radial distortion coefficient(s) κ_1 (, and κ_2 if you implemented Extension).
   - Outline your single camera and stereo camera calibration and extension details.
   - Single camera and stereo camera calibration parameters and estimated errors.
   - Your extensions.
   - Analysis of your results.
   - Supplement your results with tables, graphs, and figures.

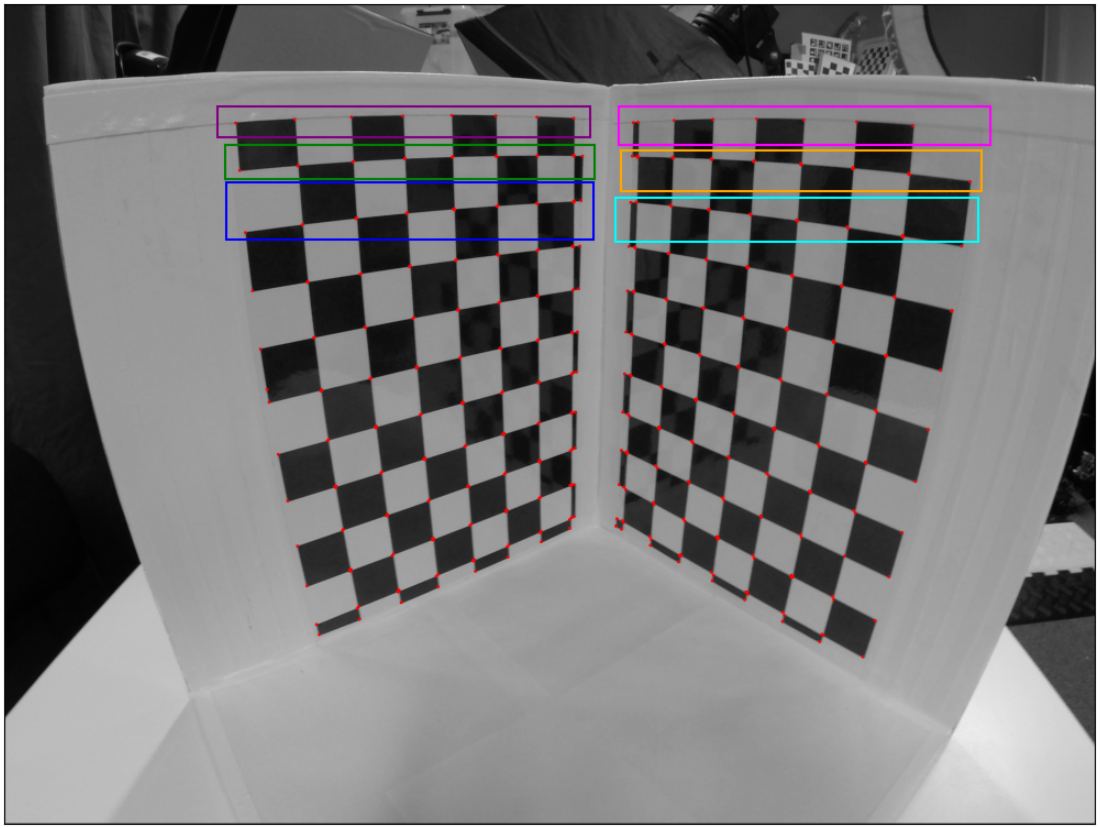
Distortion removal details

1. Given a chessboard image, load all the corner points we provided.



Distortion removal details

2. For the first three horizontal lines on the left chessboard, group the corresponding corners. And, do the same for the right chessboard. You can use the provided Python function to perform corner points grouping.



Distortion removal details

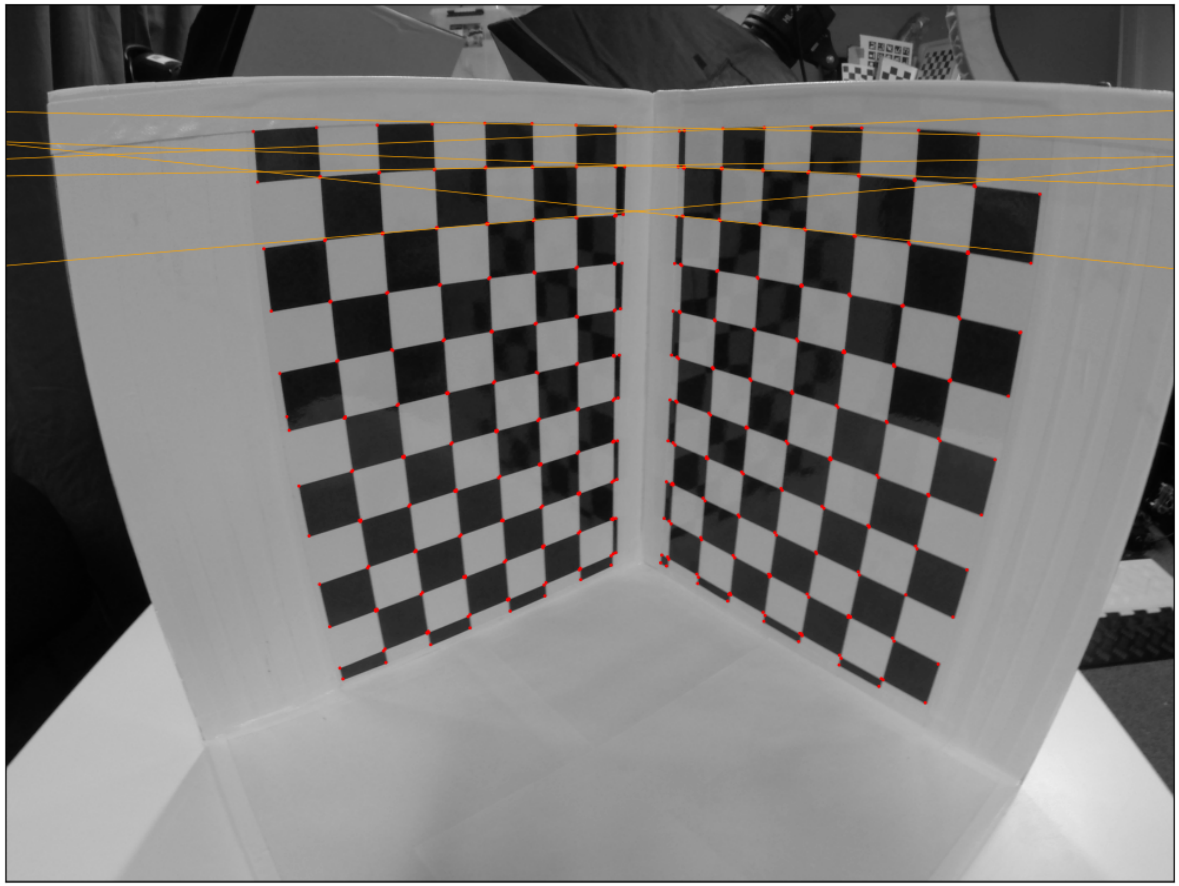
3. Initialise κ_1 as zero (only for the first iteration), and compute the new corner coordinate (x_u, y_u) using the following equations and κ_1 value.

$$x_u = (x_d - c_x)(1 + \kappa_1((x_d - c_x)^2 + (y_d - c_y)^2) + c_x$$
$$y_u = (y_d - c_y)(1 + \kappa_1((x_d - c_x)^2 + (y_d - c_y)^2) + c_y$$

where (x_d, y_d) is the distorted corner coordinate found in the corner detection step, and (c_x, c_y) is the centre of the given chessboard image.

Distortion removal details

4. Next, you need to use the line-fitting algorithm provided to you, to fit a line through the new coordinates computed in step three. Here, the red pixels are the original corner coordinates.

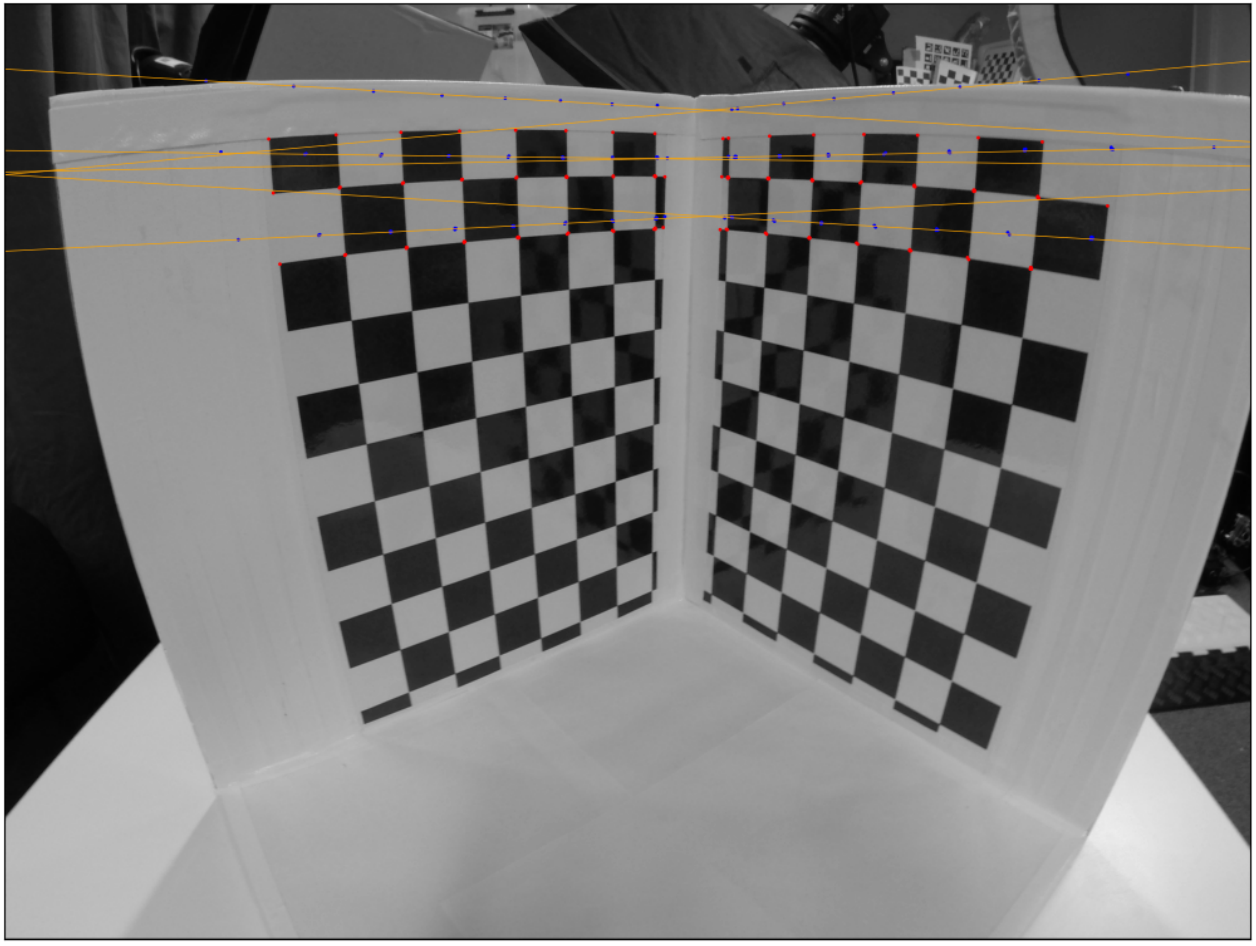


Distortion removal details

5. Once you fit the lines, you need to compute, for each line, the mean squared error (MSE) between the original distorted corners (x_d, y_d) and the corresponding coordinates on the fitted line. Then, you compute the average error E_{avg} concerning all six horizontal lines in the chessboard image fitted in Step 4.
6. If E_{avg} is less than a threshold ($E_{avg} = 4$), stop; otherwise, increase κ_1 by 10^{-9} and go to Step 3. (Hint: run the loop 200 times)

Distortion removal details

The image shows the distorted (red) and undistorted (blue) corner coordinates and the corresponding fitted lines based on the estimated κ_1 after brute forcing κ_1.



Distortion removal details

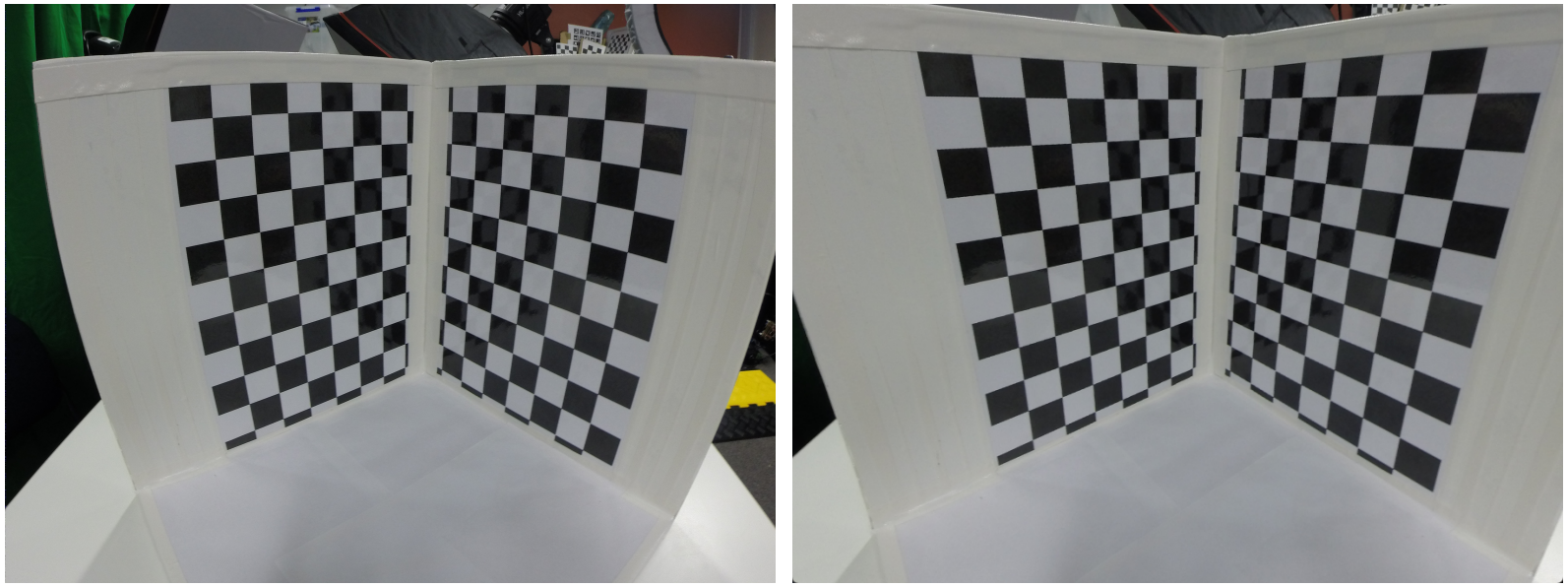
6. Once the optimal κ_1 is estimated, you will use the undistortion equations to undistort all the coordinates in the original distorted image.

$$x_u = (x_d - c_x)(1 + \kappa_1((x_d - c_x)^2 + (y_d - c_y)^2) + c_x$$
$$y_u = (y_d - c_y)(1 + \kappa_1((x_d - c_x)^2 + (y_d - c_y)^2) + c_y$$

7. Output the optimal radial distortion coefficient(s) κ_1 (, and κ_2 if you implemented Extension). As well, the smallest MSE you found during the distortion removal process.
8. Finally, you use the provided Python script to do nearest-neighbour interpolation for picking the corresponding colours.

Final results for distortion removal

- Left image: original image with visible radial distortion
- Right image: warped image after removing distortion and interpolation



Skeleton code explanation

CS773A2Ph1.py:

```
############################################################
### Removing Distortion
############################################################
distortion_remover = DistortionRemover()
smallest_MSE, kappa_1, kappa_2, average_kappa_value = distortion_remover.remove(group_corners_left_right)
print(f"Smallest MSE: {smallest_MSE}")
print(f"kappa 1: {kappa_1}")
print(f"kappa 2: {kappa_2}")
print(f"Average kappa value: {average_kappa_value}")
print("================================================\n\n\n\n")
```

We have given you the code for printing the smallest MSE, kappa 1, kappa 2, and average kappa value.

Skeleton code explanation

distortion_remover.py:

```python
class DistortionRemover():
    def __init__(self) -> None:
        pass

    def remove(self, distorted_corner_groups):
        '''
        This should be a list of grouped corner points.
        For example:
        undistorted_corner_groups = [
            [[242, 972], [2383, 483], [298, 2394]],  # group 1
            [[483, 734], [293, 781], [982, 129]], # group 2
        ]
        '''
        undistorted_corner_groups = None

        # A helper function to compute slope and y-intercept for each point in the group
        line_fitting_alg = LineFittingAlgorithm()
        best_slope_list, best_intercept_list = line_fitting_alg.run(undistorted_corner_groups)









        # Assign your computed values here!
        # If you choose to not estimate kappa 2, you can leave kappa_2 and average_kappa_value as they are.
        # But DO NOT delete them below!
        smallest_MSE = None
        kappa_1 = None
        kappa_2 = None
        average_kappa_value = None
        return smallest_MSE, kappa_1, kappa_2, average_kappa_value   # DO NOT CHANGE THIS LINE!!!
```

Tsai calibration details

1. Find coordinates (x_u, y_u) in the undistorted image from the lattice coordinates (u, v). Assuming that (c_x, c_y) is the image centre.

$$x_u = s_x d_x (u - c_x)$$

$$y_u = d_y (c_y - v)$$

- Pixel size in the image sensor in GoPro H3 is **0.00155** *mm* in both x and y direction.
- Pixel size in the image sensor in W3 is **0.001691** *mm* in x direction and **0.001663** *mm* in y direction.
- c_x and c_y can be computed using the image size.
- s_x is initially set to 1.

Tsai calibration details

2. Compute vector L using calculated list of coordinates (x_u, y_u) and Moore-Penrose inverse.

$$\underbrace{\begin{pmatrix} y_u^0 X^0 & y_u^0 Y^0 & y_u^0 Z^0 & y_u^0 & -x_u^0 X^0 & -x_u^0 Y^0 & -x_u^0 Z^0 \\ y_u^1 X^1 & y_u^1 Y^1 & y_u^1 Z^1 & y_u^1 & -x_u^1 X^1 & -x_u^1 Y^1 & -x_u^1 Z^1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ y_u^n X^n & y_u^n Y^n & y_u^n Z^n & y_u^n & -x_u^n X^n & -x_u^n Y^n & -x_u^n Z^n \end{pmatrix}}_{n \times 7} \overbrace{\begin{pmatrix} s_x r_{11}/t_y \\ s_x r_{12}/t_y \\ s_x r_{13}/t_y \\ s_x t_x/t_y \\ r_{21}/t_y \\ r_{22}/t_y \\ r_{23}/t_y \end{pmatrix}}^{L}_{7 \times 1} = \underbrace{\begin{pmatrix} x_u^0 \\ x_u^1 \\ \vdots \\ x_u^n \end{pmatrix}}_{n \times 1}$$

Let $\mathrm{L} = [a_1, a_2, a_3, a_4, a_5, a_6, a_7]^T$

Tsai calibration details

3. Compute t_y without considering the sign.

$$|t_y| = \frac{1}{\sqrt{a_5^2 + a_6^2 + a_7^2}}$$

4. Compute s_x using $|t_y|$.

$$s_x = |t_y| \sqrt{(a_1^2 + a_2^2 + a_3^2)}$$

Tsai calibration details

5. Find the sign of t_y.
   - Choose the reference 3D point whose image position is the most distant from the principal point (image centre).
   - Use pixel coordinates relative to centre, and image centre must be (0,0).
   - Let this 3D point be defined as (X, Y, Z) and its corresponding 2D point be as (x, y).
   - Compute these parameters:

$$r_{11} = a_1 t_y; r_{12} = a_2 t_y; r_{13} = a_3 t_y;$$
$$r_{21} = a_5 t_y; r_{22} = a_6 t_y; r_{23} = a_7 t_y;$$
$$t_x = a_4 t_y$$

   - Define new 2D coordinates as:

$$\begin{cases} x_{new} = r_{11}X + r_{12}Y + r_{13}Z + t_x \\ y_{new} = r_{21}X + r_{22}Y + r_{23}Z + t_y \end{cases}$$

   - Compare signs of x_{new} and x, and signs of y_{new} and y. If they are not the same flip sign of t_y.

Tsai calibration details

6. Recalculate these parameters:

$$r_{11} = a_1 \frac{t_y}{s_x}; r_{12} = a_2 \frac{t_y}{s_x}; r_{13} = a_3 \frac{t_y}{s_x};$$
$$r_{21} = a_5 t_y; r_{22} = a_6 t_y; r_{23} = a_7 t_y;$$
$$t_x = a_4 \frac{t_y}{s_x}$$

7. Let

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

8. Compute r_{31}, r_{32}, r_{33} using the cross product of its first two rows.
9. Now you have a complete R matrix.

Tsai calibration details

10. Estimate focal length f and t_z by using Moore-Penrose inverse again.

$$\underbrace{\begin{pmatrix} r_{21}X^0 + r_{22}Y^0 + r_{23}Z^0 + t_y & -y_u^0 \\ r_{21}X^1 + r_{22}Y^1 + r_{23}Z^1 + t_y & -y_u^1 \\ \vdots & \vdots \\ r_{21}X^n + r_{22}Y^n + r_{23}Z^n + t_y & -y_u^n \end{pmatrix}}_{n \times 2} \underbrace{\begin{pmatrix} f \\ t_z \end{pmatrix}}_{2 \times 1} = \underbrace{\begin{pmatrix} y_u^0 r_{31}X^0 + y_u^0 r_{32}Y^0 + y_u^0 r_{33}Z^0 \\ y_u^1 r_{31}X^1 + y_u^1 r_{32}Y^1 + y_u^1 r_{33}Z^1 \\ \vdots \\ y_u^n r_{31}X^n + y_u^n r_{32}Y^n + y_u^n r_{33}Z^n \end{pmatrix}}_{n \times 1}$$

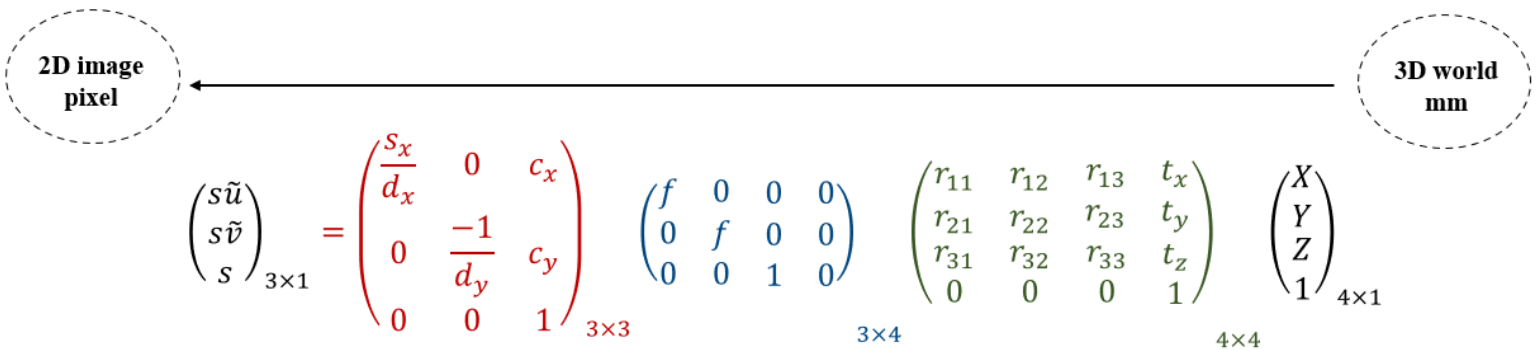
- X, Y, Z are the 3D point, and corresponding y_u were calculated from Step 1.

Tsai calibration details

11 Combine matrix R with t_x, t_y, t_z to form matrix Rt.

$$Rt = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

12 Project 3D coordinates to 2D pixels.



Camera calibration detailed specifications of inputs and outputs

Inputs:

- Use provided undistorted images and corresponding corner points to gather all X_w, Y_w, Z_w, u, v coordinates of calibration cube feature points in *mm* and the coordinates of their traces in the image in pixels for each of the images provided.

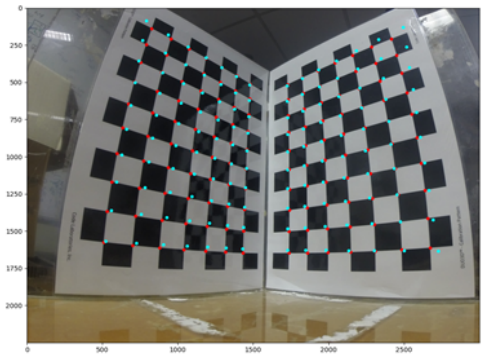
Outputs:

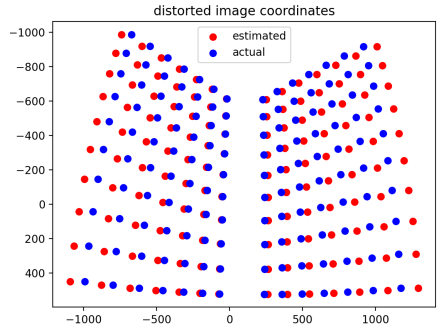
- For each (stereo) camera(s):
  - Tsai calibration parameters for each of the H3 and W3 cameras as a print out of your code and in a table in your report.
  - Parameters: a_1, a_2, a_3, a_4, a_5, a_6, a_7, s_x, t_x, t_y, t_z, f, matrix Rt.
  - Tsai calibration errors (in the cube, image for single camera calibration; 3D errors for stereo camera calibration) as a print out of your code and in a table in your report.

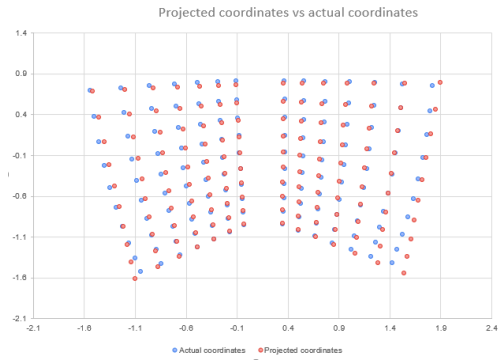
What kind of outputs/results should we produce for camera calibration?

- An overall average and standard deviation of mono and stereo calibration errors for each camera.
- Tables of calibration errors (cube, plane) for each calibration point and each camera.
- A curve of calibration errors based on the distance of the traces of calibration points in the image to the center of an image.
- The calibration image with the positions of calibration points in the image and back projected calibration points in the image for each image.
- A 3D plot of all triangulated calibration points after performing stereo-calibration.
- Extension: material demonstrating the improvement on calibration errors.
- Any other clever way of showcasing your results

Visualisation examples







m1	m2	m3	m4	m5	m6	m7
0	35195.69	0	703.9137	0	-12126.318	0
0	60233.96	0	718.7823	0	-29435.364	0
0	86244.37	0	733.3705	0	-55008.157	0
0	113434.3	0	749.236	0	-89232.867	0
0	141969.3	0	766.5729	0	-132348.49	0
0	171849.9	0	784.7029	0	-185580.67	0
0	203202	0	803.8052	0	-249986.07	0
0	27586.93	18648.76	551.7386	0	-12136.983	-8204.600839
0	47132.4	19010.44	562.4392	0	-29306.516	-11820.52779
0	67444.26	19384.49	573.5056	0	-54709.312	-15724.27513

L	
-0.00427	a1
0.004324	a2
0.000216	a3
-0.15696	a4
-0.00193	a5
-0.00158	a6
-0.00509	a7

Table 1:

Camera	**GoPro Hero 3**	**FujiFilm W3**
Image width (px)	3000	3584
Image height (px)	2250	2016
Pixel size	0.00155	0.001691
Estimated focal length (mm)	2.839186016	7.301248868
Estimated $\lvert\kappa_1\rvert$ (px^{-2})	4.2682E-10	1.40763E-10
Average error (mm)	0.032886659	0.0052393727

Estimated $[R\lvert t]$ (GoPro) (mm for t)	-0.704305235	0.709120286	0.033204756	18.78136897
	-0.223085933	-0.18371347	-0.957330678	128.7109817
	-0.67277066	-0.681668848	0.287588631	468.8091167
	0	0	0	1
Estimated $[R\lvert t]$ (W3) (mm for t)	-0.752987677	0.65799239	-0.007454758	22.46895468
	-0.018074251	-0.032481305	-0.999308904	158.2196091
	-0.657779827	-0.752332588	0.036350743	995.2947631
	0	0	0	1

Figure: Calibration steps and results visualisation examples

Skeleton code explanation

Tasi_camera_calibration.py:

```python
class TsaiCameraCalibrator():
    def __init__(self) -> None:
        np.set_printoptions(suppress = True)

    def calibrate_2D(self, corner_list, image, camera_type):
        ################################################################
        ### You can put all your code for camera calibration here!!! ###
        ################################################################




        ################################################################
        ### Insert all parameters into the Python dictionary below!!! ###
        ################################################################
        required_parameters = {
            'a1': 0.0,
            'a2': 0.0,
            'a3': 0.0,
            'a4': 0.0,
            'a5': 0.0,
            'a6': 0.0,
            'a7': 0.0,
            'sx': 0.0,
            'tx': 0.0,
            'ty': 0.0,
            'tz': 0.0,
            'f': 0.0,
            'Rt': np.random.rand(4, 4)
        }

        # Insert your projected 2D coordinates here!
        projected_2D_coordinates = None

        ##########################################
        ### DO NOT CHANGE THE RETURN VARIABLES!!! ###
        ##########################################
        return projected_2D_coordinates, required_parameters
```

Skeleton code explanation

calibration_error_calculator.py:

```python
class CalibrationErrorCalculator():
    def __init__(self) -> None:
        pass

    def calculate_error_2D(self, estimate_coordinates, actual_coordinates):
        ########################################################################
        ### You can put all your code for calculating the errors here!!! ###
        ########################################################################









        ##########################################
        ### Insert all calculated errors here!!! ###
        ##########################################
        RMSE = 0  # Root Mean Squared Error (remember to round it to 4 decimal places)
        mean_abs_error_xy = 0  # Mean Absolute Error
        std_abs_error_xy = 0  # Standard Deviation of Absolute Error
        abs_errors_xy = 0  # A List of Absolute Errors
        std_sqr_error_xy = 0  # Standard Deviation of Squared Error (remember to round it to 4 decimal places)

        ### DO NOT CHANGE THE RETURN VARIABLES!!!
        return RMSE, mean_abs_error_xy, std_abs_error_xy, abs_errors_xy, std_sqr_error_xy
```

Grading

- This assignment has a total (and maximum) of 10 marks.
- Estimating κ_1 for the H3 image provided, and performing single and stereo camera calibration on both H3 and W3 images will earn you four marks.
- Code outline and quality of reporting will earn you three marks
- **Extension 1**: estimating κ_2 for at least one of the images provided, supporting report content and results (such as images, line fitting errors, before\after distortion removal) will earn you a maximum of 1.5 marks.
- **Extension 2** will earn you a maximum of 1.5 marks.
- Due date Sunday 25^{th} May, 23:59 NZ time

Organisation

- All submissions have to be submitted using Github Classroom before due date (use the same repository as A1).
- No Jupyter Notebook is allowed for this assignment.
- Keep the same teams.
- Write a short report on what you did, where you extended, and what the findings of your experiments were (a few A4 pages max, do not repeat material already included in the lecture notes).
- The report must have a title, team members' names, a couple of sentences on who did what, and a clear structure.
- Report should be pushed to Github in a subfolder Reports, and name your report as your group name.
- The focus is on doing some programming to learn the class material and improve your skills in proper (scientific) reporting.
- Good luck!!