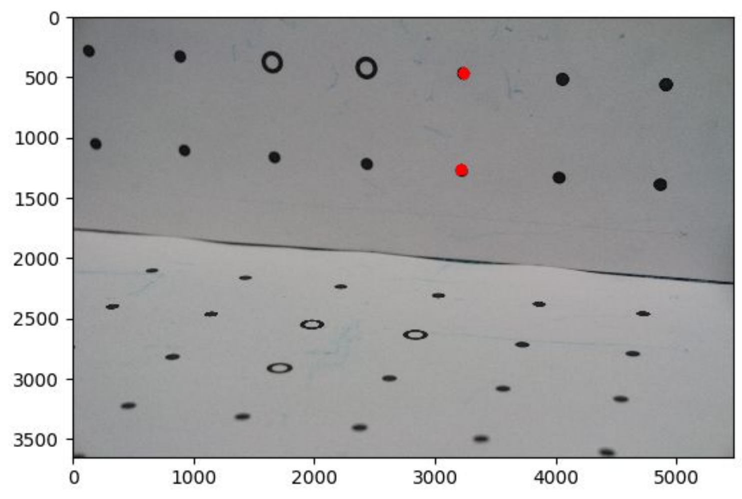
Report: Assignment - 1

Chris Andrew
2018701019

1. DLT camera calibration

- For DLT based calibration, we consider 6 points from an image and their corresponding world coordinates to estimate the camera projection matrix. To do this, we first solve the system of linear equations U = MX, where U is the image coordinates, X is the world coordinates and M is the camera projection matrix. Since the following equation may not always have a solution, we find the Least Squared Error(LSE) solution for this equation. To do this we use SVD to decompose the combined matrixed [X| U] and then solve for M.
- **<u>Challenges faced:</u>**
    - Understanding DLT algorithm.
    - Reforming the equations in a usable form that can be solved using SVD
    - Using SVD to find the LSE solution
- **<u>Results:</u>**
    - For the image provided, I manually measured the points in the image and their world coordinates and used them for computing the projection matrix. The projection matrix is then decomposed into **K[R | t]** to get the intrinsic and extrinsic properties of the camera. We use 6 points to estimate these parameters and then try and predict the remaining points using the parameters found.

- The projected points are plotted on the image below:



- It is observed that only a few points are predicted correctly and the remaining fall out of the image boundary.
- The obtained projection matrix is decomposed into K[R | t] and the individual values are analysed.

  - The obtained K matrix is:

  [-2.20803200e+00 -4.02576157e-01 -2.23531143e-01]
  [0.00000000e+00 2.10059939e+00 -7.52497629e-01]
  [0.00000000e+00 0.00000000e+00 1.37041865e-03]

  - R matrix is:

  [0.88795044 0.06053788 0.45593768]
  [0.11023221 -0.99041945 -0.0831755]
  [0.44653428 0.12411474 -0.88611662]

  - t vector is:

  [-47.91541011 292.80454521 840.35159163]

  It is observed that in the K matrix focal length is negative. This can be solved however by changing R values such that the focal lengths are positive without changing the final projection matrix.

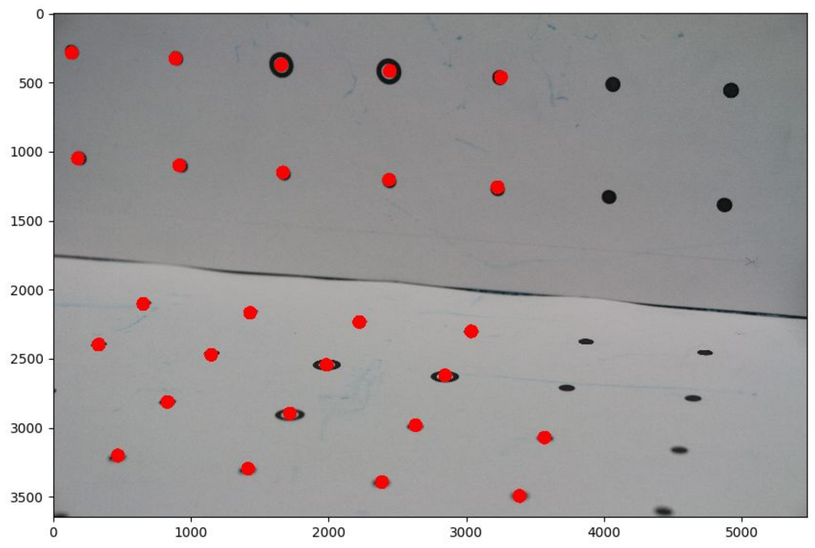
- **Code(DLT)Code(images to video)**

```
import numpy as np
import pdb
from scipy import linalg

def decompose_projection(P):
    """Decompose the projection matrix into K, R, C"""
    M = P[:, 0:3]
    MC = -1 * P[:, 3]
    C = np.dot(np.linalg.inv(M), MC)
    K, R = linalg.rq(M)
    return K, R, C


def reconstruct_projection(K, R, C):
    """Create the projection matrix to be multiplied in camera
equation."""
    M = np.dot(K, R)
    MC = -1 * np.dot(M, C)
    P = np.hstack((M, MC))
    return P


def DLT_method(image_points, world_points):
    assert image_points.shape[0] == world_points.shape[0]
    A = np.zeros((image_points.shape[0]*2, 11))
    U = np.zeros((image_points.shape[0]*2, 1))
    i = 0
    for i_points, w_points in zip(image_points, world_points):
        u, v = i_points
        x, y, z = w_points
        A[i] = [x, y, z, 1, 0, 0, 0, 0, -u*x, -u*y, -u*z]
        U[i] = -u
        A[i+1] = [0, 0, 0, 0, x, y, z, 1, -v*x, -v*y, -v*z]
        U[i+1] = -v

        i += 2
```

```python
    # Solving using SVD
    H = np.hstack((A, U))
    u_, D, v_T = np.linalg.svd(H)
    P = (v_T[-1]).reshape((-1, 1))
    divider = P[-1, 0]
    P = P/divider
    P = P.reshape(3, 4)
    return P


def find_projection_error(P, image_points, world_points):
    """Find the error in predicted points for a projection matrix."""
    total_error = 0
    for img_point, (x, y, z) in zip(image_points, world_points):
        A = np.array([[x], [y], [z], [1]])
        i_pred = np.dot(P, A)
        i_pred = i_pred.reshape(1, -1)[0]
        i_pred = i_pred[:-1] / i_pred[-1]
        error = np.linalg.norm(i_pred - img_point, ord=2)
        total_error += error
    return total_error

if __name__ == "__main__":
    world_points = np.array([[36, 0, 0], [0, 0, 36], [0, 36, 0],
                             [36, 36, 0], [36, 0, 36], [0, 0, 72]])
    image_points = np.array([[396.505, 209.674], [473.951, 246.394],
[486.636, 138.904],
                             [402.514, 132.227], [385.822, 237.047],
[465.94, 277.77]])
    projection_matrix = DLT_method(image_points, world_points)
    K, R, C = decompose_projection(projection_matrix)
    print(K)
    print(R)
    print(C)
```

2. RANSAC camera calibration

- For RANSAC based calibration, we use the RANSAC algorithm to find the optimal camera projection matrix. 6 points and their corresponding world coordinates are randomly sampled from the image. The camera projection matrix is then computed using DLT with these 6 points. The image coordinates of the remaining known points are predicted using their world coordinates and the projection matrix. An error is computed to see how accurate these predictions are to the correct values. This process is repeated for a number of iterations and the least error solution is chosen.
- **<u>Challenges faced:</u>**
    - Understanding RANSAC algorithm.
    - Using DLT to predict image coordinates.
    - Calculating error and finding the best projection matrix.
- **<u>Results:</u>**
    - For the image provided, I manually measured the points in the image and their world coordinates and used them for computing the projection matrix. The projection matrix is then decomposed into **K[R | t]** to get the intrinsic and extrinsic properties of the camera. The algorithm is run for 500 iterations, with 6 points randomly selected at every iteration and the remaining points used to calculate the error.
    - The projected points are plotted on the image below:



    - It is observed that more points are predicted correctly as compared to DLT based methods.

- The obtained projection matrix is decomposed into K[R | t] and the individual values are analysed.
  - The obtained K matrix is:
  [-2.26008894e+01 -2.04168029e-01 5.36494998e+00]
  [0.00000000e+00 2.24534025e+01 4.72210387e+00]
  [0.00000000e+00 0.00000000e+00 1.77268431e-03]
- **- R matrix is:**
  [0.97777124 -0.00752716 0.20953937]
  [-0.08896921 -0.91982255 0.38211381]
  [0.1898628 -0.39226244 -0.90004572]
- **- t vector is:**
  [-32.78197198 199.8035223 532.76931481]
  It is observed that in the K matrix focal length is negative. This can be solved however by changing R values such that the focal lengths are positive without changing the final projection matrix.

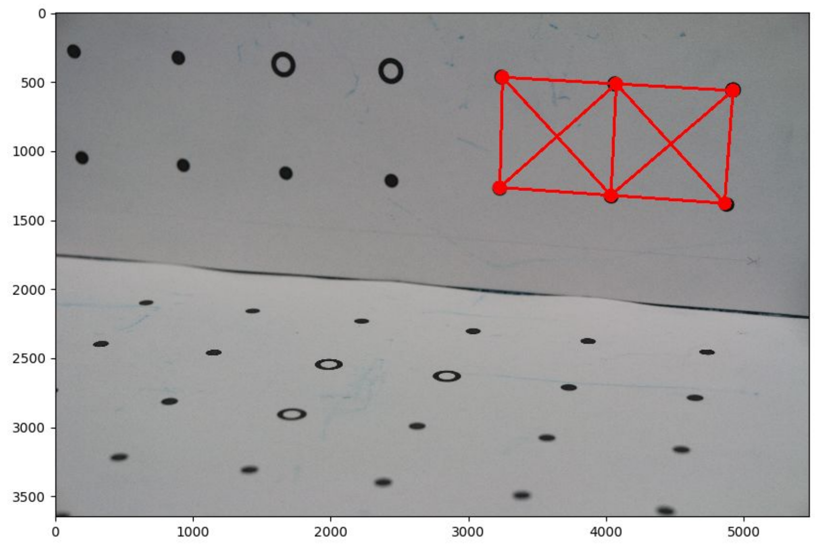
- **Code(RANSAC)**

```
import numpy as np

from dlt import DLT_method, find_projection_error,
decompose_projection


def ransac_estimation(image_points, world_points, iter=500):
    """Use the RANSAC algorithm with DLT to find best projection
matrix"""
    n_points = image_points.shape[0]
    best_P = None
    min_error = np.inf
    for i in range(iter):
        print("Iteration number {}".format(i))
        indices = np.random.permutation(n_points)
        X, Y = image_points[indices[0:6]], world_points[indices[0:6]]
        X_test, Y_test = image_points[indices[6::]],
world_points[indices[6::]]

        projection_matrix = DLT_method(X, Y)
        error = find_projection_error(projection_matrix, X_test,
```

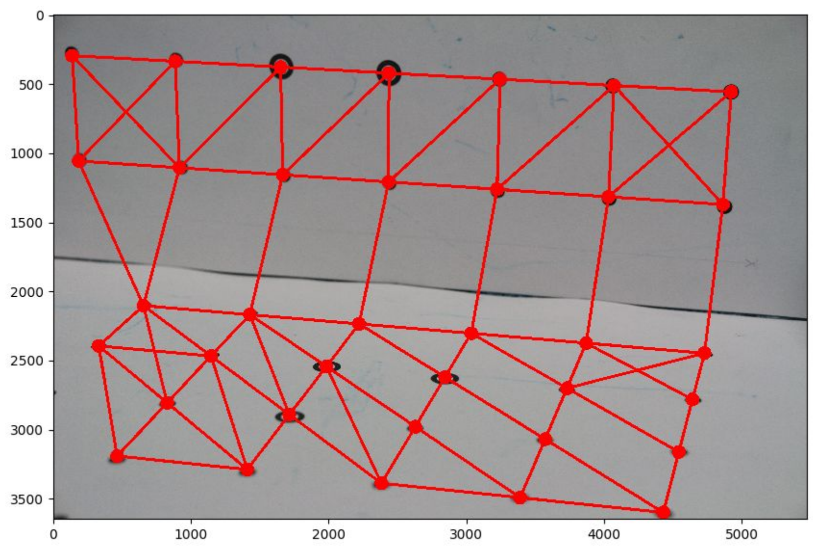
```python
Y_test)

        if error < min_error:
            min_error = error
            best_P = projection_matrix

    return best_P


if __name__ == "__main__":
    from points import image_points, world_points
    projection_matrix = ransac_estimation(image_points, world_points)
    K, R, C = decompose_projection(projection_matrix)
    print(K)
    print(R)
    print(C)
```

3. Wireframe:

- Once we have the prediction matrix using any of the three algorithms, we can use this matrix to predict the image coordinates for points using their known world coordinates. We then construct a wireframe using these predicted image coordinates and then join them together using lines.
- **Challenges faced:**
    - Learning OpenCV draw functions to draw on an image.
    - Finding the points that needed to be connected based on distance.
- **Results:**
    - Wireframe using DLT based predictions



- Wireframe using RANSAC based predictions:



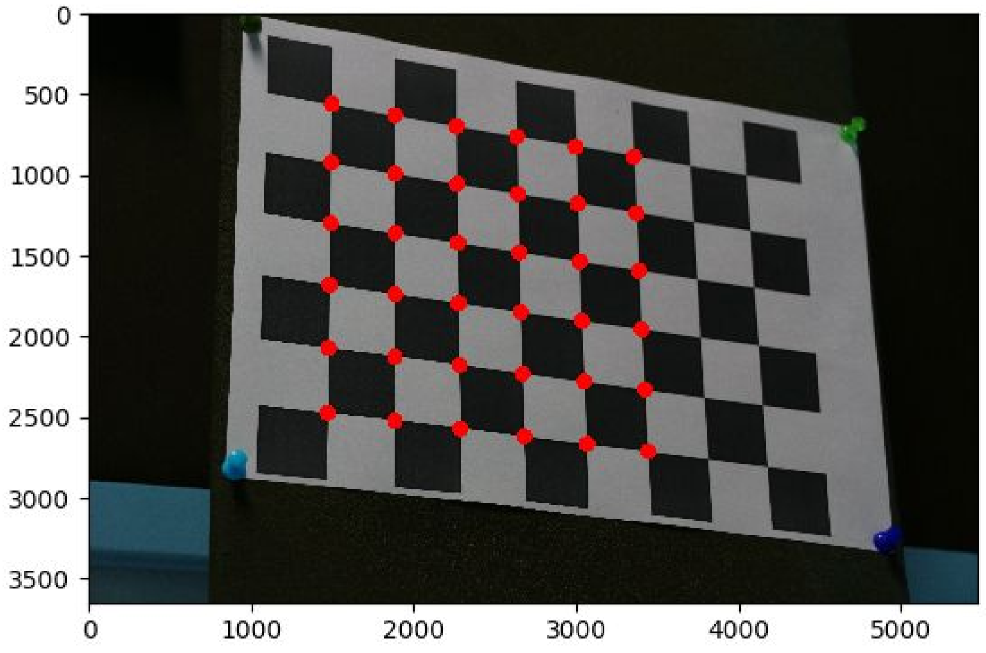
- **<u>Code(Wireframe):</u>**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
import pdb

from dlt import DLT_method
from ransac import ransac_estimation
from zhang import zhang_calibration, predict_points_zhang


def predict_points(projection_matrix, world_points):
    """Use the projection matrix to predict image_points"""
    points = []
    for (x, y, z) in world_points:
        A = np.array([[x], [y], [z], [1]])
        i_pred = np.dot(projection_matrix, A)
        i_pred = i_pred.reshape(1, -1)[0]
        i_pred = i_pred[:-1] / i_pred[-1]
        points.append(i_pred)
    points = np.array(points, dtype=np.int)
    return points


def plot_wireframe(image, points):
    img = image.copy()
    for (x, y) in points:
        img = cv2.circle(img, (x, y), 50, (255, 0, 0), -1)

    for p in points:
        distance = np.sum((points - p)**2, axis=1)
        min_index = np.argsort(distance)[1:5]
        for nearest_point in points[min_index]:
            img = cv2.line(img, tuple(p), tuple(nearest_point), (255,
0, 0), 20)
    return img


def plot_points(image, points):
    img = image.copy()
    for (x, y) in points:
        if x < 0 or x >= img.shape[0] or y < 0 or y >= img.shape[1]:
            continue
        img = cv2.circle(img, (x, y), 50, (255, 0, 0), -1)
    return img


if __name__ == "__main__":
    from points import image_points, world_points

    image = cv2.imread('Assignment1_Data/IMG_5455.JPG')
    projection_matrix = DLT_method(image_points, world_points)

    pred_points = predict_points(projection_matrix,
world_points[0:6])
    img = plot_points(image, pred_points)

    plt.imshow(img)
    plt.show()

    projection_matrix = ransac_estimation(image_points, world_points)
    pred_points = predict_points(projection_matrix, world_points)
    img = plot_points(image, pred_points)
    plt.imshow(img)
    plt.show()
```

4. Zhangs Method:

- In Zhangs method for calibration, a known pattern such as a checkerboard is used to estimate the calibration matrix. The checkerboard world coordinates are known treating one corner as the origin and computing the rest. The image coordinates are then taken from the image of the checkerboard and the projection matrix is estimated using these. To make this process more robust the coordinates are calculated using multiple images and combining the results. Zhangs method for calibration does not require physical measurements of the world coordinates to estimate the camera parameters. Zhangs method gives the intrinsic and extrinsic parameters of the camera as seprate matrices/vectors.
- **<u>Challenges faced:</u>**
    - Understanding Zhangs method
    - Understanding the OpenCV implementation of Zhangs method and using it.

- **Results:**
    - The predicted points using Zhangs algorithm are plotted:



    - The intrinsic parameters estimated are (K):
      [1.36415094e+04 0.00000000e+00 3.31635881e+03]
      [0.00000000e+00 1.36632517e+04 1.50037396e+03]
      [0.00000000e+00 0.00000000e+00 1.00000000e+00]

    - This is a consistent result because the focal lengths are positive here. In the case of DLT, the focal lengths were negative since we did not get the intrinsic and extrinsic properties separately and had to decompose the projection matrix to get them.

- **Code(Zhangs):**

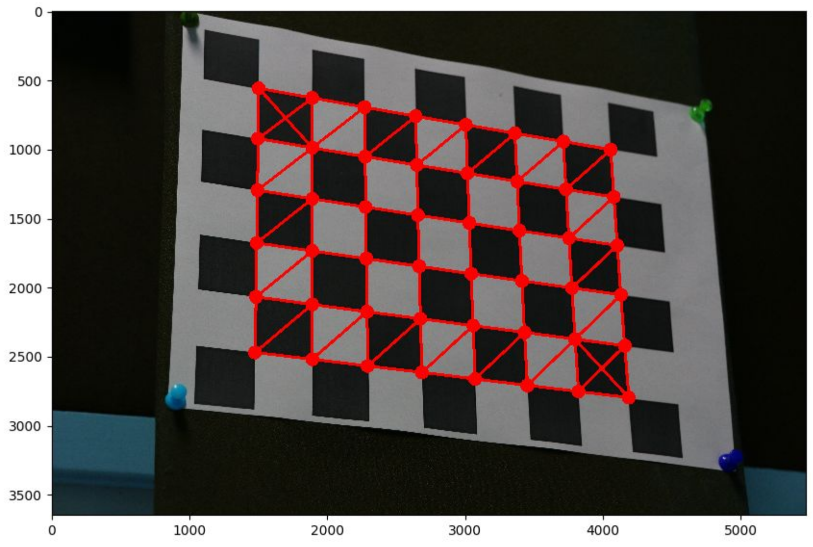
```python
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt

from wireframe import plot_points


def find_img_points(img, size=(8, 6)):
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,
30, 0.001)
    objp = np.zeros((size[0]*size[1], 3), np.float32)
    objp[:, :2] = np.mgrid[0:size[0], 0:size[1]].T.reshape(-1, 2)

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, size, None)
    if ret:
        corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1,
-1), criteria)
        return objp, corners2, gray.shape
    return None, None, None


def zhang_calibration(images, size):
    objpoints = []
    imgpoints = []
    for i, img in enumerate(images):
        print("Processing image ", i)
        objp, imgp, shape = find_img_points(img, size)
        if objp is not None and imgp is not None:
            objpoints.append(objp)
            imgpoints.append(imgp)
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,
imgpoints, shape[::-1], None, None)
    return ret, mtx, dist, rvecs, tvecs, objpoints, imgpoints
```

```
def predict_points_zhang(rvecs, tvecs, mtx, dist, world_points):
    img_points = cv2.projectPoints(world_points, rvecs, tvecs, mtx,
dist)
    return img_points


if __name__ == "__main__":
    folder = "Assignment1_Data/"
    image_names = ["IMG_5456.JPG", "IMG_5457.JPG", "IMG_5458.JPG",
"IMG_5459.JPG", "IMG_5460.JPG",
                   "IMG_5461.JPG", "IMG_5462.JPG", "IMG_5463.JPG",
"IMG_5464.JPG", "IMG_5465.JPG",
                   "IMG_5466.JPG", "IMG_5467.JPG", "IMG_5468.JPG",
"IMG_5469.JPG", "IMG_5470.JPG"]
    images = [cv2.imread(os.path.join(folder, x)) for x in
image_names]

    ret, mtx, dist, rvecs, tvecs, objpoints, imgpoints =
zhang_calibration(images, size=(8, 6))
    pred_points = predict_points_zhang(rvecs[-1], tvecs[-1], mtx,
dist, objpoints[-1])[0].reshape(48, 2)
    img = plot_points(images[-1], pred_points)

    plt.imshow(img)
    plt.show()
    print(mtx)
```

5. Wireframe using Zhangs:

- We draw the wireframe using the points predicted by Zhangs algorithm on the checkerboard.

- **Results:**



- **Code:**

```
import cv2
import os

from zhang import zhang_calibration, predict_points_zhang
from wirframe import plot_wireframe

folder = "Assignment1_Data/"
    image_names = ["IMG_5456.JPG", "IMG_5457.JPG", "IMG_5458.JPG",
"IMG_5459.JPG", "IMG_5460.JPG",
                   "IMG_5461.JPG", "IMG_5462.JPG", "IMG_5463.JPG",
"IMG_5464.JPG", "IMG_5465.JPG",
                   "IMG_5466.JPG", "IMG_5467.JPG", "IMG_5468.JPG",
"IMG_5469.JPG", "IMG_5470.JPG"]
    images = [cv2.imread(os.path.join(folder, x)) for x in
image_names]

    ret, mtx, dist, rvecs, tvecs, objpoints, imgpoints =
zhang_calibration(images, size=(8, 6))
    pred_points = predict_points_zhang(rvecs[-1], tvecs[-1], mtx,
dist, objpoints[-1])[0].reshape(48, 2)
    img = plot_wireframe(images[-1], pred_points)

    plt.imshow(img)
    plt.show()
```

6. World Origin:

- In the calibration matrices obtained for the given images with measurements using DLT and RANSAC, the world origin falls on the correct image pixel as is observed in the image. This is result bears out in our observation.
- **Results:**
    -

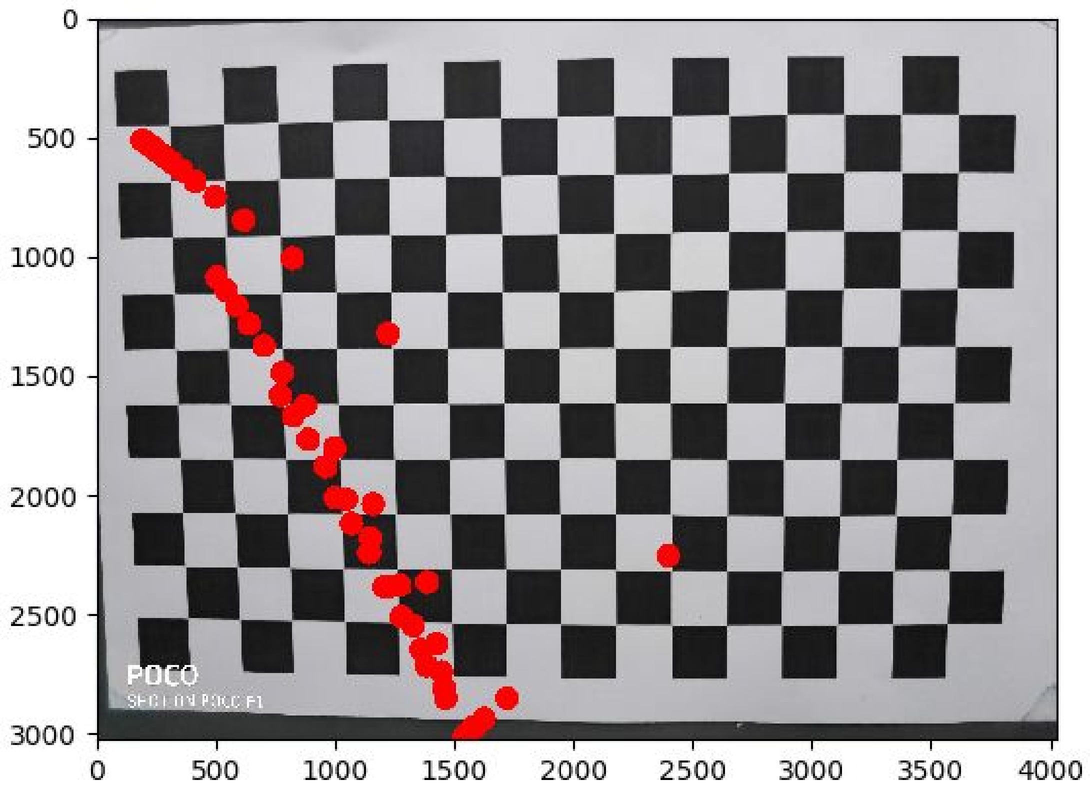
7. Own Images:

- For experiments using our own images, I used a checkerboard image taken with my phone. I chose this image because it would be easier to estimate the world coordinates and the image coordinates from this image without having to physically measure both. I first did Zhangs calibration method to get the world and image coordinates. These coordinates were then used to calibrate using RANSAC and DLT. I plotted the predicted points for all three calibration techniques as well as the camera parameters for them.
- **<u>Challenges faced:</u>**
  - Estimated camera parameters:

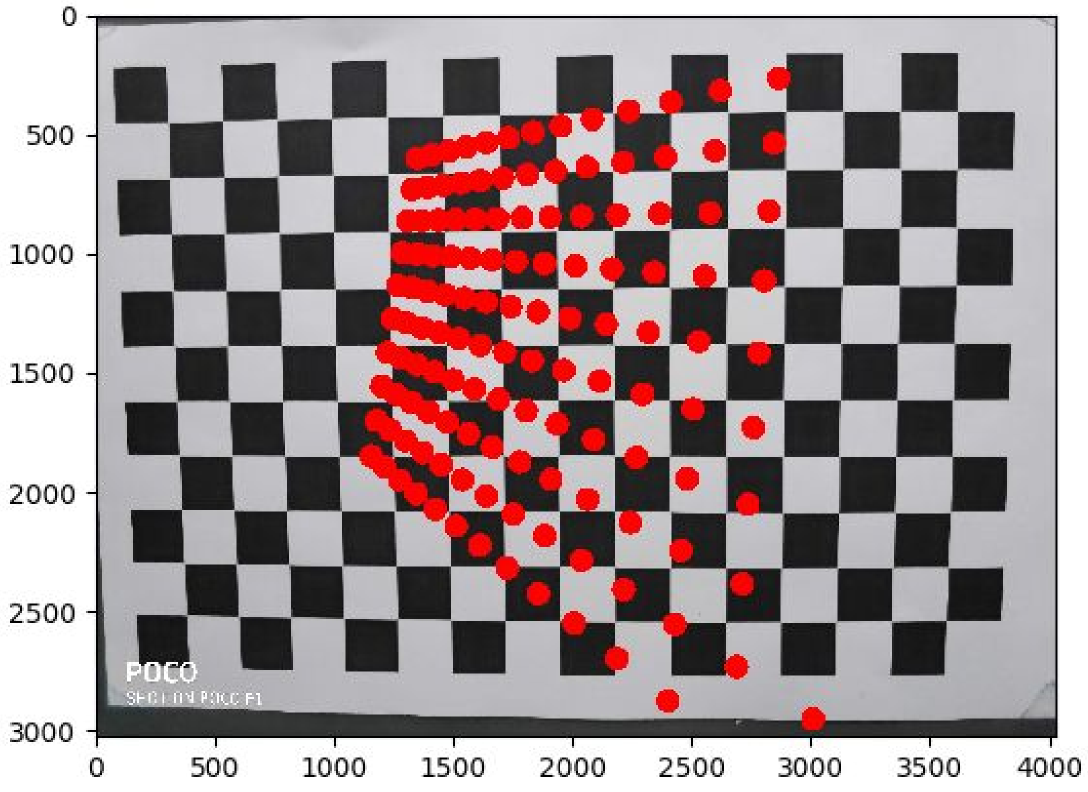
    Zhangs(intrinsic parameters)
    ```
    [2.85900179e+03 0.00000000e+00 2.06236818e+03]
    [0.00000000e+00 2.88219115e+03 1.47109720e+03]
    [0.00000000e+00 0.00000000e+00 1.00000000e+00]
    ```
    DLT(K)
    ```
    [ 1.60191172e+01 -3.85013883e+02 -2.41784425e+15]
    [ 0.00000000e+00 -7.28580800e+02  2.69789521e+18]
    [ 0.00000000e+00  0.00000000e+00 -1.21502447e+34]
    ```
    DLT(R)
    ```
    [ 9.99135568e-01  4.15706301e-02  6.56819071e-36]
    [ 4.15706301e-02 -9.99135568e-01  7.75811072e-36]
    [ 6.88502251e-36 -7.47836054e-36 -1.00000000e+00]
    ```
    DLT(C)
    ```
    [ 5.48095138e+00  4.59277009e-01 -4.80010343e-35]
    ```

    RANSAC(K)
    ```
    [-9.03258545e+00  2.84713324e+01 -1.08530295e+13]
    [ 0.00000000e+00 -1.31496727e+02  3.60576300e+18]
    [ 0.00000000e+00  0.00000000e+00  1.62389129e+34]
    ```
    RANSAC(R)
    ```
    [ 9.53181531e-01  3.02398691e-01 -2.72423266e-36]
    [ 3.02398691e-01 -9.53181531e-01 -3.96157975e-37]
    [-2.71648591e-36 -4.46193924e-37 -1.00000000e+00]
    ```
    RANSAC(C)
    ```
    [ 1.57316321e+02  4.51023200e+01 -3.85891473e-34]
    ```
    All values are in line with our previous observations. DLT and RANSAC give negative intrinsic parameters since we have to decompose the projection matrix ourselves. Whereas in Zhangs method, the intrinsic parameters are obtained directly.

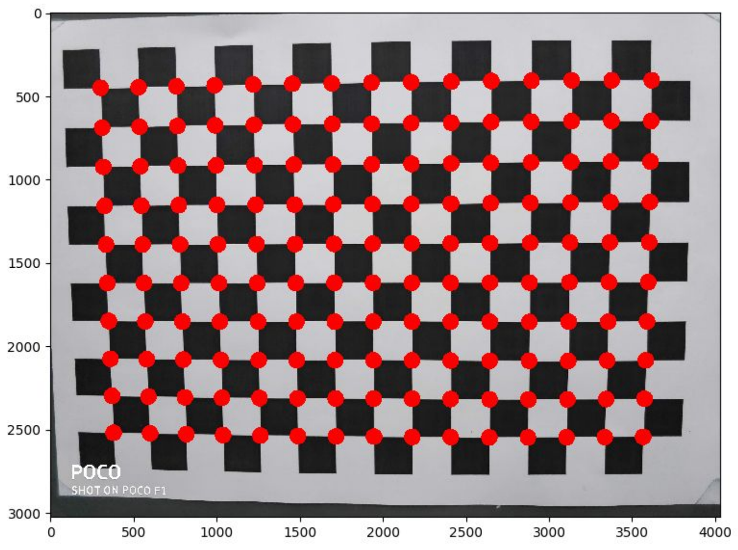
- I also plotted the predicted points on my own images using the three calibration methods:
  DLT:



RANSAC



ZHANG:



- **Code(Own images):**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
import pdb

from dlt import DLT_method, decompose_projection
from ransac import ransac_estimation
from zhang import zhang_calibration, predict_points_zhang
from wireframe import predict_points, plot_points

if __name__ == "__main__":
    # Use Zhangs method to find image points and world points
    folder = "myimages/"
    image_names = ["img_0.jpg", "img_1.jpg", "img_2.jpg",
"img_3.jpg", "img_4.jpg",
                   "img_5.jpg", "img_6.jpg", "img_7.jpg",
"img_8.jpg", "img_9.jpg"]
    images = [cv2.imread(os.path.join(folder, x)) for x in
image_names]

    ret, mtx, dist, rvecs, tvecs, objpoints, imgpoints =
zhang_calibration(images, size=(15, 10))

    pred_points = predict_points_zhang(rvecs[-1], tvecs[-1], mtx,
dist, objpoints[-1])[0].reshape(150, 2)
    img = plot_points(images[-1], pred_points)

    plt.imshow(img)
    plt.show()
    print(mtx)

    image_points = imgpoints[-1].reshape(150, 2).astype(int)
    world_points = objpoints[-1].astype(int)
    projection_matrix = DLT_method(image_points[0:6],
world_points[0:6])
    K, C, R = decompose_projection(projection_matrix)
    print(K)
    print(C)
    print(R)
    pred_points = predict_points(projection_matrix, world_points)
    img = plot_points(images[-1], pred_points)

    plt.imshow(img)
    plt.show()

    projection_matrix = ransac_estimation(image_points, world_points)
    K, C, R = decompose_projection(projection_matrix)
    print(K)
    print(C)
    print(R)
    pred_points = predict_points(projection_matrix, world_points)
    img = plot_points(images[-1], pred_points)
    plt.imshow(img)
    plt.show()
```