CS 5330 Project 4 : Calibration and Augmented Reality Report

Name: Hang Zhao
Email:zhao.hang1@northeastern.edu
Team member: Hang Zhao

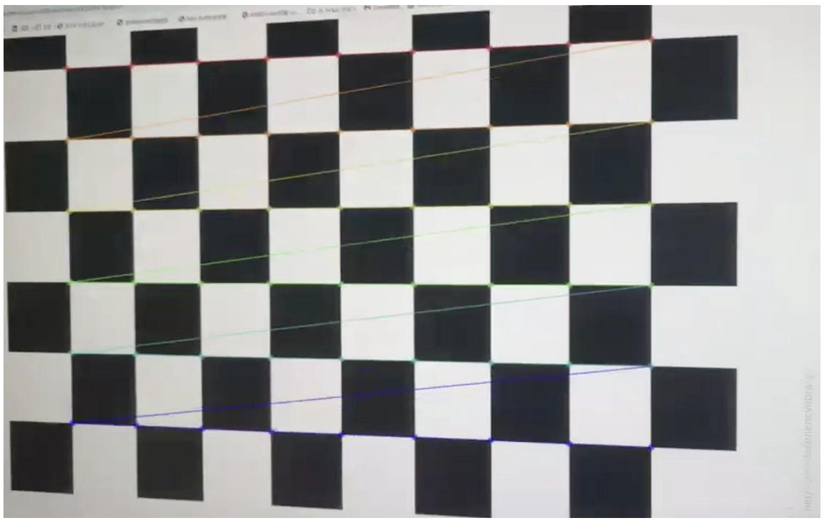
Project Description:

This project is all about calibrating a camera and using that calibration to drop virtual objects into real-world footage. We start by detecting something like a checkerboard pattern and pinpoint its corners. Then, after collecting a few images from different angles, we run a calibration step to figure out the camera's internal settings and distortion. With that info, the software can track how the camera moves and overlay a virtual object—like a 3D model—so it appears naturally in the scene. The end result is a neat little AR demo where the virtual and real worlds line up smoothly.

Images and description

Task 1 Detect and Extract Target Corners

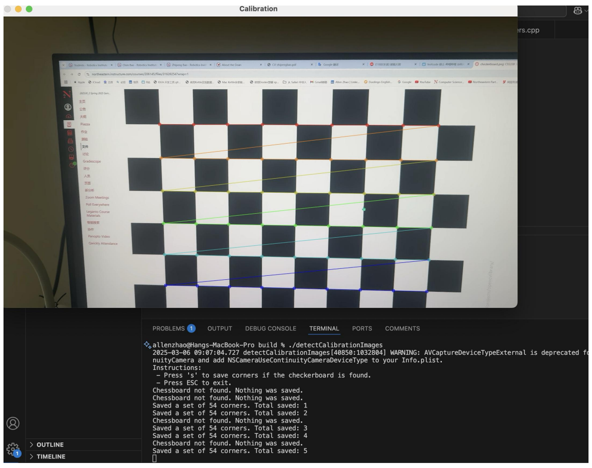
I use checkerboard.png as a target. Demo video https://youtu.be/Bgc8jsYPNME



Detect and Extract Target Corners

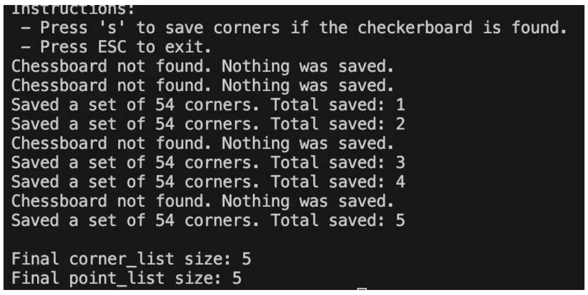
Task 2 Select Calibration Images

In the task "Select Calibration Image", I first took several chessboard images at different angles and distances to obtain diverse image data and ensure that the corners are detected as accurately as possible in a bright and uniform environment. Each time the chessboard is successfully detected, the program will save the detected 2D pixel coordinates (stored in `corner_list`) and the pre-calculated 3D world coordinates (stored in `point_list`) correspondingly, so that each `corner_list` entry can be matched one by one with a `point_list` entry. In order to intuitively demonstrate the detection effect, I drew marks on one of the 9×6 chessboard images with `drawChessboardCorners` and took a screenshot as an example. During the acquisition process, if the user presses the `s` key when the chessboard is not detected, the program will remind you that it cannot be saved, and when the detection is successful, the corner point and its corresponding 3D coordinates are stored. Finally, I collected 5 valid checkerboard images, each containing 54 inner corner points, which can provide sufficient multi-view data for subsequent camera calibration (via `calibrateCamera`). In this way, I can more accurately obtain the camera intrinsic parameters and distortion coefficients in the subsequent steps, making the subsequent augmented reality process more stable.



```
allenzhao@Hangs-MacBook-Pro build % ./detectCalibrationImages
2025-03-06 09:07:04.727 detectCalibrationImages[40850:1032804] WARNING: AVCaptureDeviceTypeExternal is deprecated fo
nuityCamera and add NSCameraUseContinuityCameraDeviceType to your Info.plist.
Instructions:
 - Press 's' to save corners if the checkerboard is found.
 - Press ESC to exit.
Chessboard not found. Nothing was saved.
Chessboard not found. Nothing was saved.
Saved a set of 54 corners. Total saved: 1
Saved a set of 54 corners. Total saved: 2
Chessboard not found. Nothing was saved.
Saved a set of 54 corners. Total saved: 3
Saved a set of 54 corners. Total saved: 4
Chessboard not found. Nothing was saved.
Saved a set of 54 corners. Total saved: 5
```

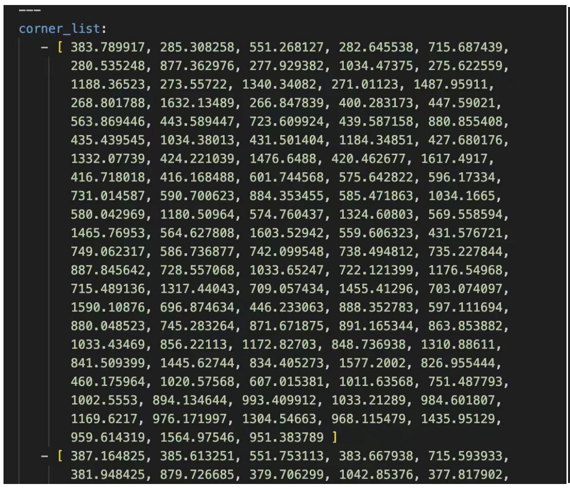
runtime screenshot



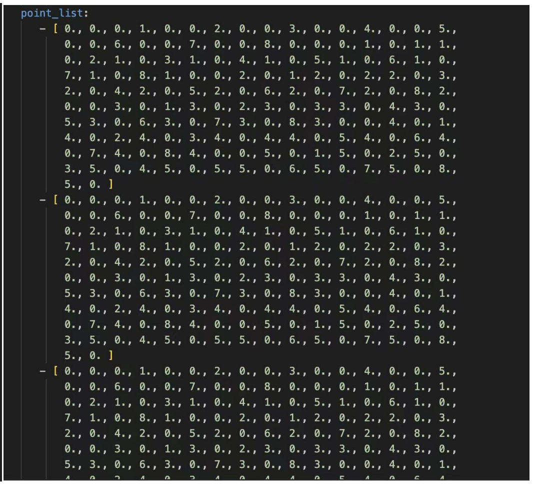
```
Instructions:
 - Press 's' to save corners if the checkerboard is found.
 - Press ESC to exit.
Chessboard not found. Nothing was saved.
Chessboard not found. Nothing was saved.
Saved a set of 54 corners. Total saved: 1
Saved a set of 54 corners. Total saved: 2
Chessboard not found. Nothing was saved.
Saved a set of 54 corners. Total saved: 3
Saved a set of 54 corners. Total saved: 4
Chessboard not found. Nothing was saved.
Saved a set of 54 corners. Total saved: 5

Final corner_list size: 5
Final point_list size: 5
```

runtime screenshot



```
---
corner_list:
  - [ 383.789917, 285.308258, 551.268127, 282.645538, 715.687439,
      280.535248, 877.362976, 277.929382, 1034.47375, 275.622559,
      1188.36523, 273.55722, 1340.34082, 271.01123, 1487.95911,
      268.801788, 1632.13489, 266.847839, 400.283173, 447.59021,
      563.869446, 443.589447, 723.609924, 439.587158, 880.855408,
      435.439545, 1034.38013, 431.501404, 1184.34851, 427.680176,
      1332.07739, 424.221039, 1476.6488, 420.462677, 1617.4917,
      416.718018, 416.168488, 601.744568, 575.642822, 596.17334,
      731.014587, 590.700623, 884.353455, 585.471863, 1034.1665,
      580.042969, 1180.50964, 574.760437, 1324.60803, 569.558594,
      1465.76953, 564.627808, 1603.52942, 559.606323, 431.576721,
      749.062317, 586.736877, 742.099548, 738.494812, 735.227844,
      887.845642, 728.557068, 1033.65247, 722.121399, 1176.54968,
      715.489136, 1317.44043, 709.057434, 1455.41296, 703.074097,
      1590.10876, 696.874634, 446.233063, 888.352783, 597.111694,
      880.048523, 745.283264, 871.671875, 891.165344, 863.853882,
      1033.43469, 856.22113, 1172.82703, 848.736938, 1310.88611,
      841.509399, 1445.62744, 834.405273, 1577.2002, 826.955444,
      460.175964, 1020.57568, 607.015381, 1011.63568, 751.487793,
      1002.5553, 894.134644, 993.409912, 1033.21289, 984.601807,
      1169.6217, 976.171997, 1304.54663, 968.115479, 1435.95129,
      959.614319, 1564.97546, 951.383789 ]
  - [ 387.164825, 385.613251, 551.753113, 383.667938, 715.593933,
      381.948425, 879.726685, 379.706299, 1042.85376, 377.817902,
```



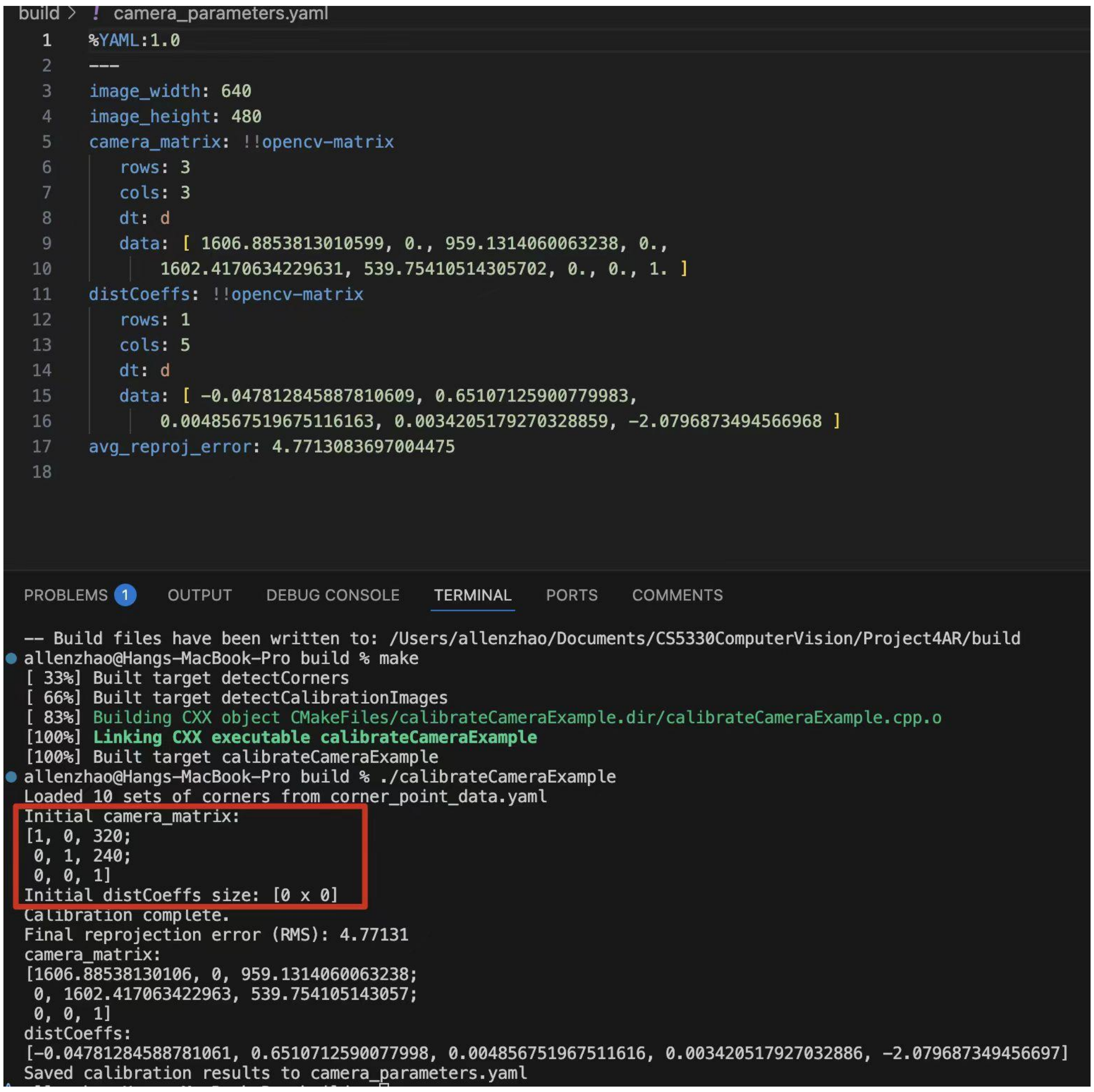
```
point_list:
  - [ 0., 0., 0., 1., 0., 0., 2., 0., 0., 3., 0., 0., 4., 0., 0., 5.,
      0., 0., 6., 0., 0., 7., 0., 0., 8., 0., 0., 0., 1., 0., 1., 1.,
      0., 2., 1., 0., 3., 1., 0., 4., 1., 0., 5., 1., 0., 6., 1., 0.,
      7., 1., 0., 8., 1., 0., 0., 2., 0., 1., 2., 0., 2., 2., 0., 3.,
      2., 0., 4., 2., 0., 5., 2., 0., 6., 2., 0., 7., 2., 0., 8., 2.,
      0., 0., 3., 0., 1., 3., 0., 2., 3., 0., 3., 3., 0., 4., 3., 0.,
      5., 3., 0., 6., 3., 0., 7., 3., 0., 8., 3., 0., 0., 4., 0., 1.,
      4., 0., 2., 4., 0., 3., 4., 0., 4., 4., 0., 5., 4., 0., 6., 4.,
      0., 7., 4., 0., 8., 4., 0., 0., 5., 0., 1., 5., 0., 2., 5., 0.,
      3., 5., 0., 4., 5., 0., 5., 5., 0., 6., 5., 0., 7., 5., 0., 8.,
      5., 0. ]
  - [ 0., 0., 0., 1., 0., 0., 2., 0., 0., 3., 0., 0., 4., 0., 0., 5.,
      0., 0., 6., 0., 0., 7., 0., 0., 8., 0., 0., 0., 1., 0., 1., 1.,
      0., 2., 1., 0., 3., 1., 0., 4., 1., 0., 5., 1., 0., 6., 1., 0.,
      7., 1., 0., 8., 1., 0., 0., 2., 0., 1., 2., 0., 2., 2., 0., 3.,
      2., 0., 4., 2., 0., 5., 2., 0., 6., 2., 0., 7., 2., 0., 8., 2.,
      0., 0., 3., 0., 1., 3., 0., 2., 3., 0., 3., 3., 0., 4., 3., 0.,
      5., 3., 0., 6., 3., 0., 7., 3., 0., 8., 3., 0., 0., 4., 0., 1.,
      4., 0., 2., 4., 0., 3., 4., 0., 4., 4., 0., 5., 4., 0., 6., 4.,
      0., 7., 4., 0., 8., 4., 0., 0., 5., 0., 1., 5., 0., 2., 5., 0.,
      3., 5., 0., 4., 5., 0., 5., 5., 0., 6., 5., 0., 7., 5., 0., 8.,
      5., 0. ]
  - [ 0., 0., 0., 1., 0., 0., 2., 0., 0., 3., 0., 0., 4., 0., 0., 5.,
      0., 0., 6., 0., 0., 7., 0., 0., 8., 0., 0., 0., 1., 0., 1., 1.,
      0., 2., 1., 0., 3., 1., 0., 4., 1., 0., 5., 1., 0., 6., 1., 0.,
      7., 1., 0., 8., 1., 0., 0., 2., 0., 1., 2., 0., 2., 2., 0., 3.,
      2., 0., 4., 2., 0., 5., 2., 0., 6., 2., 0., 7., 2., 0., 8., 2.,
      0., 0., 3., 0., 1., 3., 0., 2., 3., 0., 3., 3., 0., 4., 3., 0.,
      5., 3., 0., 6., 3., 0., 7., 3., 0., 8., 3., 0., 0., 4., 0., 1.,
```

save corner_list data

save point_list data

Task 3 Calibrate the Camera

I collected 10 images of a 9×6 checkerboard from various angles and distances. After detecting the checkerboard corners and matching them with their 3D coordinates, I used OpenCV's `calibrateCamera` on these sets. This process estimates the camera's focal length, principal point, and distortion coefficients, yielding a final reprojection error (RMS). Ideally, that RMS should be close to one pixel or less. When calibration finished, I saved the resulting `camera_matrix` and `distCoeffs` to a file for later use in undistorting images or performing augmented reality.

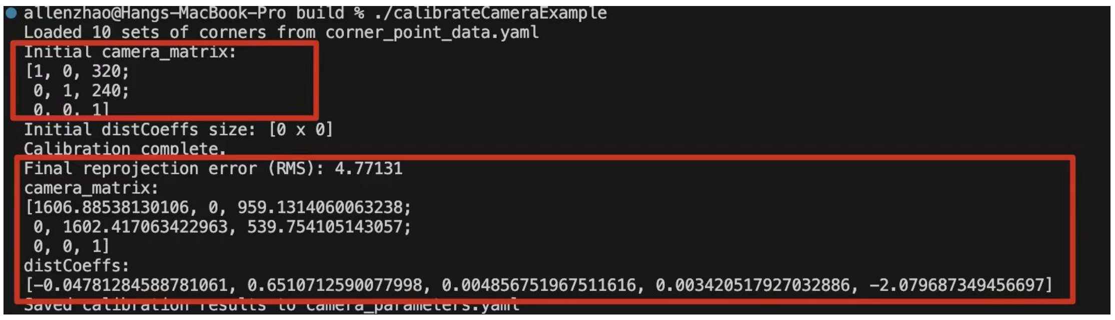


```
build > ! camera_parameters.yaml
%YAML:1.0
---
image_width: 640
image_height: 480
camera_matrix: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [ 1606.8853813010599, 0., 959.1314060063238, 0.,
       1602.4170634229631, 539.75410514305702, 0., 0., 1. ]
distCoeffs: !!opencv-matrix
   rows: 1
   cols: 5
   dt: d
   data: [ -0.047812845887810609, 0.65107125900779983,
       0.0048567519675116163, 0.0034205179270328859, -2.0796873494566968 ]
avg_reproj_error: 4.7713083697004475
```

PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMMENTS

```
-- Build files have been written to: /Users/allenzhao/Documents/CS5330ComputerVision/Project4AR/build
allenzhao@Hangs-MacBook-Pro build % make
[ 33%] Built target detectCorners
[ 66%] Built target detectCalibrationImages
[ 83%] Building CXX object CMakeFiles/calibrateCameraExample.dir/calibrateCameraExample.cpp.o
[100%] Linking CXX executable calibrateCameraExample
[100%] Built target calibrateCameraExample
allenzhao@Hangs-MacBook-Pro build % ./calibrateCameraExample
Loaded 10 sets of corners from corner_point_data.yaml
Initial camera_matrix:
[1, 0, 320;
 0, 1, 240;
 0, 0, 1]
Initial distCoeffs size: [0 x 0]
Calibration complete.
Final reprojection error (RMS): 4.77131
camera_matrix:
[1606.88538130106, 0, 959.1314060063238;
 0, 1602.417063422963, 539.754105143057;
 0, 0, 1]
distCoeffs:
[-0.04781284588781061, 0.6510712590077998, 0.004856751967511616, 0.003420517927032886, -2.079687349456697]
Saved calibration results to camera_parameters.yaml
```

camera_matrix



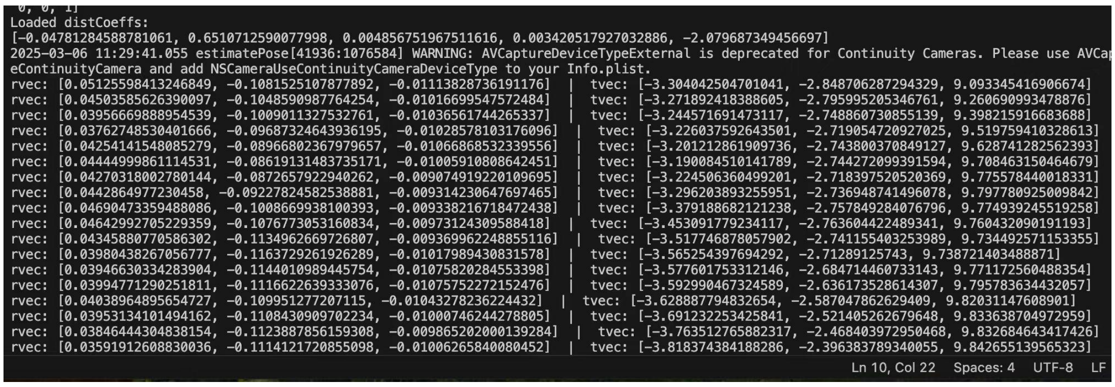
```
allenzhao@Hangs-MacBook-Pro build % ./calibrateCameraExample
Loaded 10 sets of corners from corner_point_data.yaml
Initial camera_matrix:
[1, 0, 320;
 0, 1, 240;
 0, 0, 1]
Initial distCoeffs size: [0 x 0]
Calibration complete.
Final reprojection error (RMS): 4.77131
camera_matrix:
[1606.88538130106, 0, 959.1314060063238;
 0, 1602.417063422963, 539.754105143057;
 0, 0, 1]
distCoeffs:
[-0.04781284588781061, 0.6510712590077998, 0.004856751967511616, 0.003420517927032886, -2.079687349456697]
Saved calibration results to camera_parameters.yaml
```

calibration matrix and re-projection error

Task 4 Calculate Current Position of the Camera

I used the estimatePose.cpp code to complete this task. In my program, I continuously print the rotation vector (rvec) and the translation vector (tvec) to the console in real time. As I move the camera from left to right, I can see the translation vector’s x-component shift accordingly, going

positive or negative depending on the direction. When I tilt the camera, rvec updates to reflect the angular change. These movements align with what I observe in real life: moving closer lowers the z-value in the translation vector, sliding the camera sideways adjusts the x-value, and turning the camera causes the rotation vector to vary. In all, the changes in rvec and tvec are consistent with my actual camera motions.
Demo video: https://youtu.be/S_PdgzGmXdw

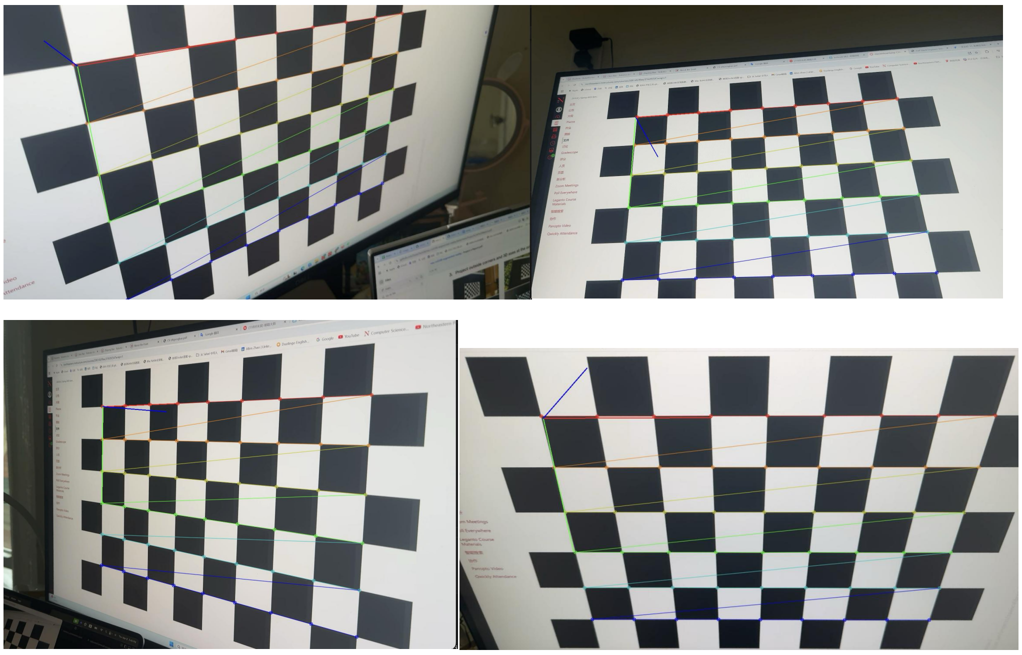


```
 0, 0, 1]
Loaded distCoeffs:
[-0.04781284588781061, 0.6510712590077998, 0.004856751967511616, 0.003420517927032886, -2.079687349456697]
2025-03-06 11:29:41.055 estimatePose[41936:1076584] WARNING: AVCaptureDeviceTypeExternal is deprecated for Continuity Cameras. Please use AVCap
eContinuityCamera and add NSCameraUseContinuityCameraDeviceType to your Info.plist.
rvec: [0.05125598413246849, -0.1081525107877892, -0.01113828736191176]  |  tvec: [-3.304042504701041, -2.848706287294329, 9.093345416906674]
rvec: [0.04503585626390097, -0.1048590987764254, -0.01016699547572484]  |  tvec: [-3.271892418388605, -2.795995205346761, 9.260690993478876]
rvec: [0.03956669888954539, -0.1009011327532761, -0.01036561744265337]  |  tvec: [-3.244571691473117, -2.748860730855139, 9.398215916683688]
rvec: [0.03762748530401666, -0.09687324643936195, -0.01028578103176096]  |  tvec: [-3.226037592643501, -2.719054720927025, 9.519759410328613]
rvec: [0.04254141548085279, -0.08966802367979657, -0.01066868532339556]  |  tvec: [-3.201212861909736, -2.743800370849127, 9.628741282562393]
rvec: [0.04444999861114531, -0.08619131483735171, -0.01005910808642451]  |  tvec: [-3.190084510141789, -2.744272099391594, 9.708463150464679]
rvec: [0.04270318002780144, -0.0872657922940262, -0.009074919220109695]  |  tvec: [-3.224506360499201, -2.718397520520369, 9.775578440018331]
rvec: [0.0442864977230458, -0.09227824582538881, -0.009314230647697465]  |  tvec: [-3.296203893255951, -2.736948741496078, 9.797780925009842]
rvec: [0.04690473359488086, -0.1008669938100393, -0.009338216718472438]  |  tvec: [-3.379188682121238, -2.757849284076796, 9.774939245519258]
rvec: [0.04642992705229359, -0.1076773053160834, -0.00973124309588418]  |  tvec: [-3.453091779234117, -2.763604422489341, 9.760432090191193]
rvec: [0.04345880770586302, -0.1134962669726807, -0.009369962248855116]  |  tvec: [-3.517746878057902, -2.741155403253989, 9.734492571153355]
rvec: [0.03980438267056777, -0.1163729261926289, -0.01017989430831578]  |  tvec: [-3.565254397694292, -2.71289125743, 9.738721403488871]
rvec: [0.03946630334283904, -0.1144010989445754, -0.01075820284553398]  |  tvec: [-3.577601753312146, -2.684714460733143, 9.771172560488354]
rvec: [0.03994771290251811, -0.1116622639333076, -0.01075752272152476]  |  tvec: [-3.592990467324589, -2.636173528614307, 9.795783634432057]
rvec: [0.04038964895654727, -0.109951277207115, -0.01043278236224432]  |  tvec: [-3.628887794832654, -2.587047862629409, 9.82031147608901]
rvec: [0.03953134101494162, -0.1108430909702234, -0.01000746244278805]  |  tvec: [-3.691232253425841, -2.521405262679648, 9.833638704972959]
rvec: [0.03846444304838154, -0.1123887856159308, -0.009865202000139284]  |  tvec: [-3.763512765882317, -2.468403972950468, 9.832684643417426]
rvec: [0.03591912608830036, -0.1114121720855098, -0.01006265840080452]  |  tvec: [-3.818374384188286, -2.396383789340055, 9.842655139565323]
                                                                                          Ln 10, Col 22   Spaces: 4   UTF-8   LF
```

rvec and tvec data

Task 5 Project Outside Corners or 3D Axes

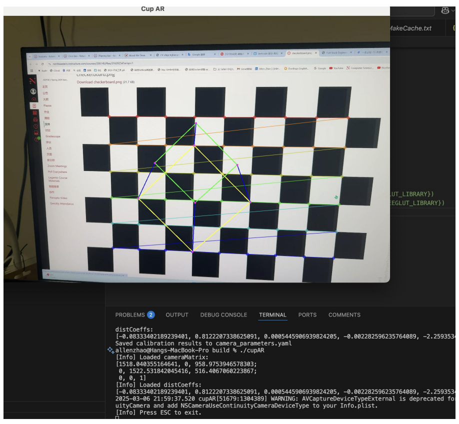
I took the rotation (rvec) and translation (tvec) vectors from the pose estimation step, then used OpenCV's projectPoints to transform selected 3D points (like the axes) into the image plane. By drawing lines where those projected points landed, I could visually confirm that the axes remained locked to the checkerboard as either the camera or the board moved. Essentially, this demonstrated how to overlay 3D content in real time, making it appear correctly oriented and positioned in the scene.



The 3D axes (red = X, green = Y, blue = Z) projected onto the checkerboard using solvePnP and projectPoints. As the camera or board moves, the axes remain aligned with the checkerboard, confirming correct pose estimation.

Task 6 Create a Virtual Object

In this task, I created a 3D virtual cup and made it appear as if it was floating above a detected chessboard in real time. The cup was designed using a set of 3D points forming a wireframe cylinder, with 4 points for the bottom circle, 4 for the top, and vertical lines connecting them. Using `solvePnP`, I estimated the camera's pose, and then applied `projectPoints` to map the cup onto the 2D image. As the camera moved, the cup maintained its correct orientation relative to the board, creating an augmented reality effect. I adjusted the cup's size and position to ensure it looked natural in the scene. To document the results, I took screenshots and videos, demonstrating how the virtual cup stayed properly aligned with the board as the camera moved.



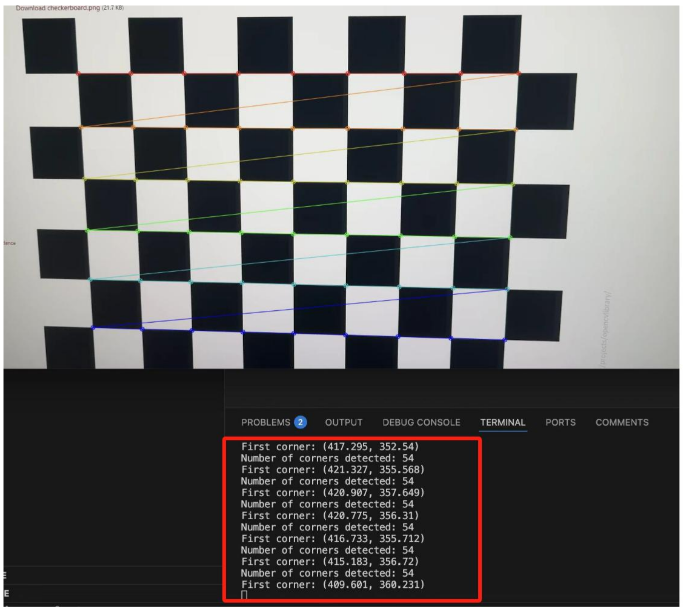
```
distCoeffs:
[-0.08333402189239401, 0.8122207338625091, 0.0005445906939824205, -0.002282596235764089, -2.259353
Saved calibration results to camera_parameters.yaml
allenzhao@Hangs-MacBook-Pro build % ./cupAR
[Info] Loaded cameraMatrix:
[1518.040355164641, 0, 958.9753946578303;
 0, 1522.531842045416, 516.4067060223867;
 0, 0, 1]
[Info] Loaded distCoeffs:
[-0.08333402189239401, 0.8122207338625091, 0.0005445906939824205, -0.002282596235764089, -2.259353
2025-03-06 21:59:37.520 cupAR[51679:1304389] WARNING: AVCaptureDeviceTypeExternal is deprecated fo
uityCamera and add NSCameraUseContinuityCameraDeviceType to your Info.plist.
[Info] Press ESC to exit.
```

a floating cup

Demo video: https://youtu.be/UKmH5aOy9iY

Task 7 Detect Robust Features

This task detects Harris Corners in a live video stream and highlights them on a pattern of choice. The user can experiment with different threshold values to see how feature detection changes. The detected feature points could later be used as anchor points for augmented reality applications.



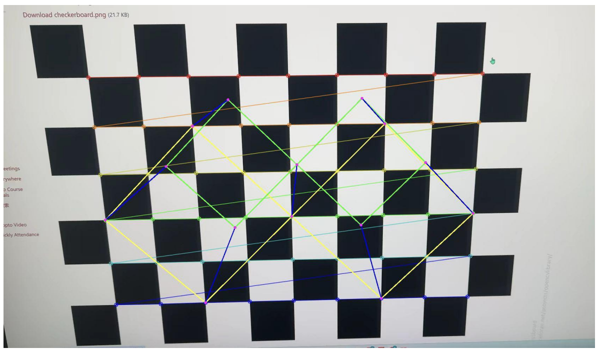
```
First corner: (417.295, 352.54)
Number of corners detected: 54
First corner: (421.327, 355.568)
Number of corners detected: 54
First corner: (420.907, 357.649)
Number of corners detected: 54
First corner: (420.775, 356.31)
Number of corners detected: 54
First corner: (416.733, 355.712)
Number of corners detected: 54
First corner: (415.183, 356.72)
Number of corners detected: 54
First corner: (409.601, 360.231)
```

When the camera is moved, the program prints the detected Harris corners to the console.

Extensions:

1. **Get your system working with multiple targets in the scene.**

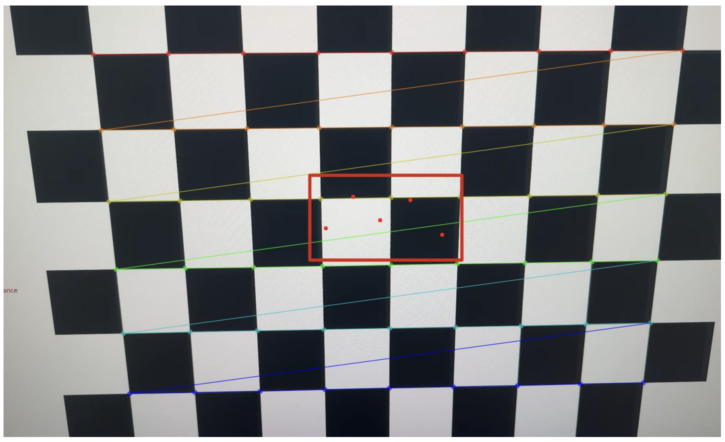
In this task, I modified the AR system to detect a single checkerboard and project two 3D wireframe cups onto it at different positions. Using OpenCV, the program detects the 9×6 checkerboard, estimates its pose with solvePnP, and then projects two sets of 3D cup points using projectPoints. The first cup is positioned at (2,3) and the second at (6,3) on the board. Each cup is drawn as a wireframe cylinder using 8 points (4 for the bottom and 4 for the top) and connected with lines. As the camera moves, both cups maintain their correct positions and orientations relative to the board. This demonstrates how multiple AR objects can be accurately placed within the same scene using camera pose estimation.



2 cups on one chessboard

2. Uber extension: Figure out how to get the 3D points on an object in the scene if the scene also has a calibration target. Show the 3-D point cloud.

I implemented a 3D point cloud reconstruction system using a checkerboard as a calibration target. The program detects the chessboard, estimates camera pose (rvec, tvec) using solvePnP, and projects 3D object points onto the image using projectPoints. This creates a real-time 3D point cloud visualization, adjusting dynamically as the camera moves.



3D point cloud

Reflection:

I really enjoyed learning how camera calibration works at a deeper level. Grabbing multiple checkerboard shots made me think carefully about angles and lighting. Once the data was in, using functions like solvePnP to lock virtual objects onto the marker felt like a fun reward for all that setup. Watching a virtual shape remain fixed as I moved the camera around gave me a “wow” moment—it reminded me that math and coding can create very real illusions. Overall, this project helped me appreciate both the power of OpenCV’s built-in functions and the importance of careful calibration steps.

Acknowledgement:

I would like to thank our teaching assistants and classmates for their valuable support and guidance throughout this project. Their help in troubleshooting technical issues and providing feedback was essential in completing our work.

I also want to express our gratitude to our professor for providing clear explanations and insightful examples that deepened our understanding of computer vision. The resources and tutorials shared during the course were incredibly helpful in learning OpenCV and applying it effectively to this project.