CS 5330 Pattern Recognition and Computer Vision — Spring 2024
Ronak Bhanushali and Rucha Pendharkar — 17th March 2024

Project IV - Calibration and Augmented Reality

Introduction

The goal of this project is to understand and perform camera calibration. The calibration parameters are then used to create a virtual object. The result is a program that is able to recognize a target and place a virtual object relative to the target, even if the target or camera is rotated or translated

Broadly this project has three parts. Part I focuses on detecting the target, selecting appropriate images for calibration and performing camera calibration. Intrinsic camera parameters are saved to a XML file. Part II focuses on calculating the pose of the target using camera parameters and corner locations. A virtual object is built and displayed relative to the target. Part III is about detecting robust features in a video stream.

Setup

A mobile phone was used to stream live video. The IP Webcam app proved to be very useful for this. The camera calibration was performed using a checkerboard. The virtual object was built using ARuco markers

Tasks:

1. Detect and Extract Target Corners
2. Select Calibration Images
3. Calibrate the Camera
4. Calculate Current Position of the Camera
5. Project Outside Corners or 3D Axes
6. Create a Virtual Object
7. Detect Robust Features
8. Extensions

Task 1: Detect and Extract Target Corners

The 9x6 checkerboard target was used for this task. The corners were detected and extracted using OpenCV functions such as *findChessboardCorners*, *cornerSubPix*. *drawChessboardCorners* was used to draw the detected corners on the target.

One small limitation we observed was that it was difficult to obtain corner points near the extreme edges of the checkerboard.

Task 2: Select Calibration Images

A functionality was added such that frame, corners and points get stored in a vector when the user presses the 's' key. The following is an example of the images used for calibration.

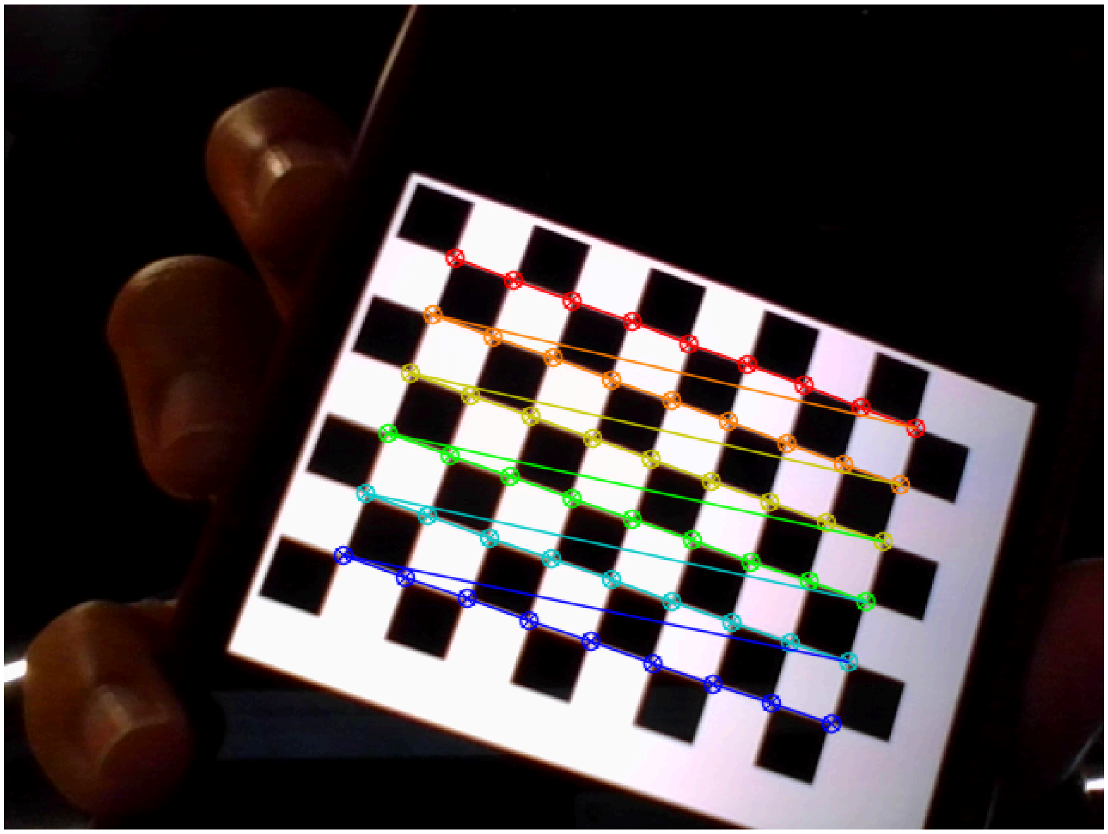
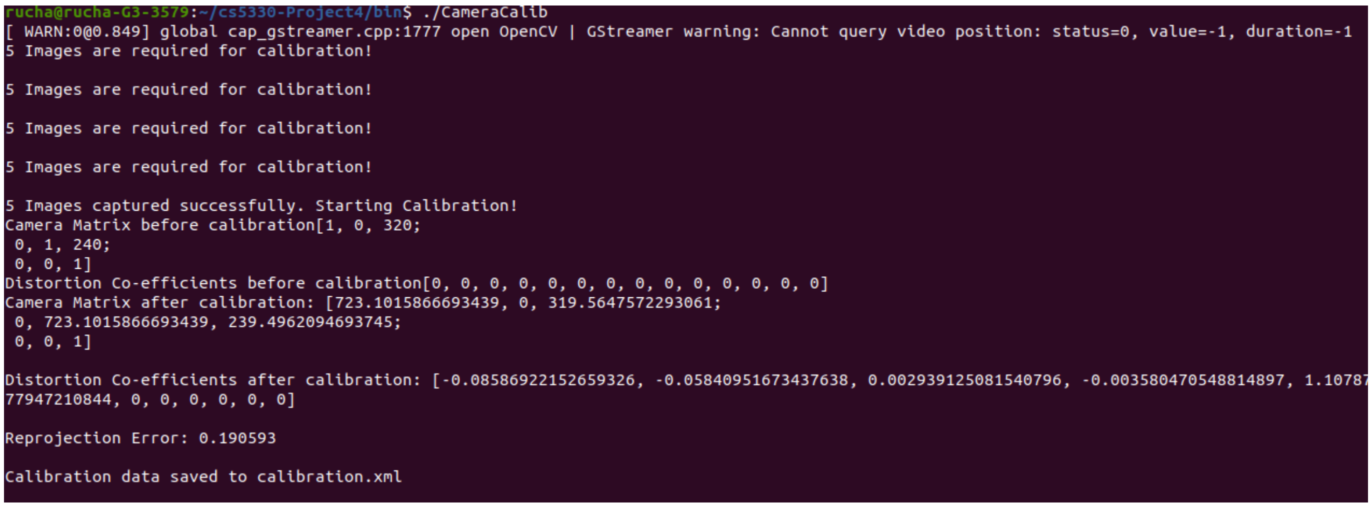


Fig1: Checkerboard with corners

Task 3: Calibrate the Camera

Building on task 2, once the vector has been populated with more than 5 images, calibration is performed. The cv::Calibrate function was used for this purpose. The camera matrix and the distortion coefficients are printed before and after calibration. The reprojection error is also printed. These intrinsic parameters are saved in an XML file for later use.



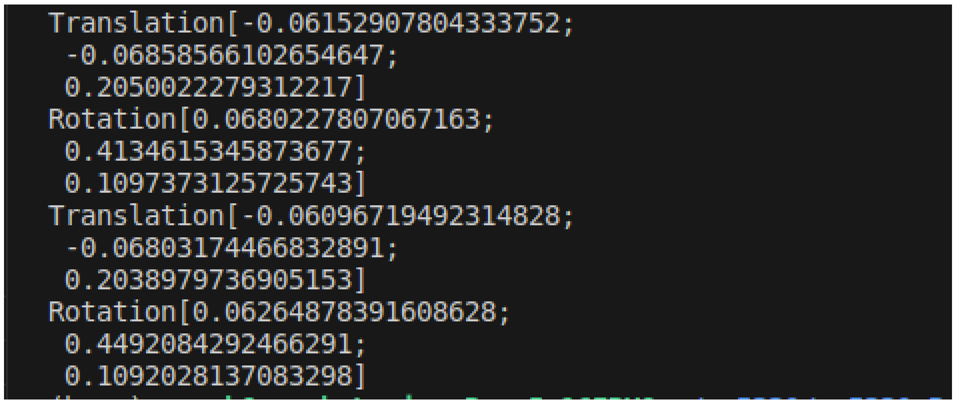
```
rucha@rucha-G3-3579:~/cs5330-Project4/bin$ ./CameraCalib
[ WARN:0@0.849] global cap_gstreamer.cpp:1777 open OpenCV | GStreamer warning: Cannot query video position: status=0, value=-1, duration=-1
5 Images are required for calibration!

5 Images are required for calibration!

5 Images are required for calibration!

5 Images are required for calibration!

5 Images captured successfully. Starting Calibration!
Camera Matrix before calibration[1, 0, 320;
 0, 1, 240;
 0, 0, 1]
Distortion Co-efficients before calibration[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Camera Matrix after calibration: [723.1015866693439, 0, 319.5647572293061;
 0, 723.1015866693439, 239.4962094693745;
 0, 0, 1]

Distortion Co-efficients after calibration: [-0.08586922152659326, -0.05840951673437638, 0.002939125081540796, -0.003580470548814897, 1.10787
77947210844, 0, 0, 0, 0, 0, 0]

Reprojection Error: 0.190593

Calibration data saved to calibration.xml
```

Fig2: Intrinsic Camera Parameters

As observed, the values of u0 and v0 have remained the same before and after calibration. Both the focal lengths are also roughly equal in length
Reprojection error represents the error in projecting the 3D points onto the 2D image plane using the estimated camera parameters. The reprojection error obtained is **0.19853.** This value makes sense, as we used the laptop's webcam for calibration here.

Task 4: Calculate Current Position of the Camera

We first undistort the image using the undistortImage function defined in functions.cpp. solvePnp function was used to calculate the current position of the camera. We used multiple targets, so for this task we use the target with id=0. Following is the printed rotation vector and translation vector from solvePnp



```
Translation[-0.06152907804333752;
 -0.06858566102654647;
 0.2050022279312217]
Rotation[0.0680227807067163;
 0.4134615345873677;
 0.1097373125725743]
Translation[-0.06096719492314828;
 -0.06803174466832891;
 0.2038979736905153]
Rotation[0.06264878391608628;
 0.4492084292466291;
 0.1092028137083298]
```

Fig3: Camera Pose

To test whether the values make sense, the camera was moved closer and away from the target which changed the 3rd element of Translation. Similarly movement in X and Y changed the first and second elements respectively. SImilar pattern was observed for rotation about X,Y, and Z. We used multiple aruco markers from this task onwards. The printed values are with respect to markers with id = 0. Following is our setup

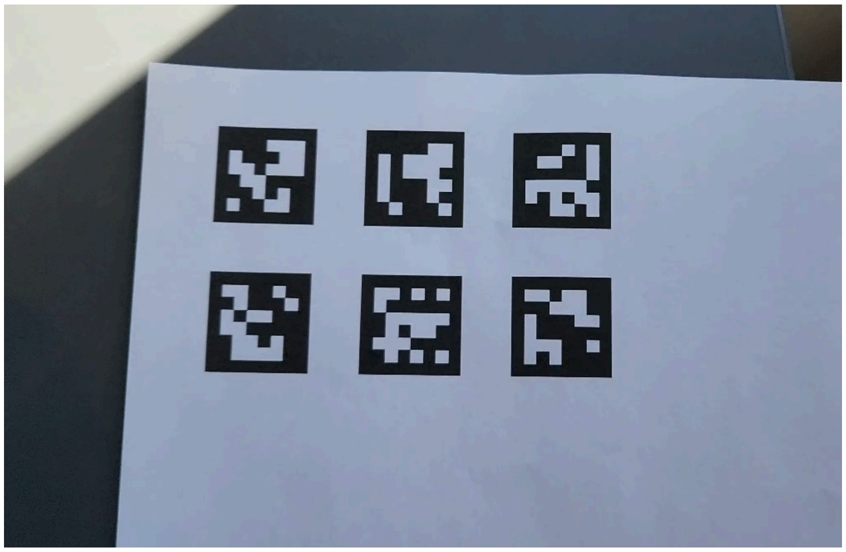


Fig4: ARuco Markers

Task 5: Project Outside Corners on 3D Axes

We have multiple targets and we projected the corners for each one as follows.

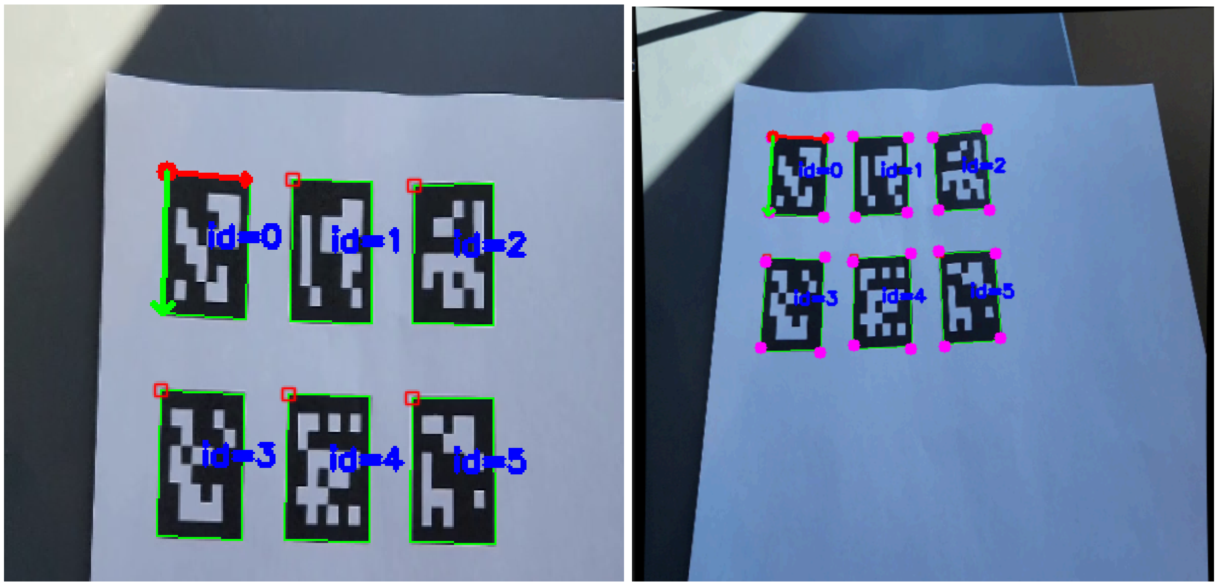


Fig5: Corners projected

The first image shows the aruco markers being detected by the getTarget function. Once we get the corners, we run solvePnp for each set of corners and find camera location with respect to each marker. The origin for each marker is considered to be the top left corner of the marker. To find the projected corners for each target, we run the projectPoints function with the respective rvec and tvec values for the marker and get the projected points. The points seem to align well with the corners indicating that the process is correct.

Task 6: Create Virtual Object

We first undistort the image and use solvePnp function to calculate the current position of the camera. We noticed that the estimate of Z direction kept changing direction. To solve this issue, we find rvec and tvec in the first iteration and then use that value for future estimates by setting useExtrinsicGuess flag true in the function. This change can be seen in task6 as we use non zero z values for that task. This method helped stabilize the Z direction and the projected points.
We made a campsite with 6 humans dancing around a tent as our projected object.

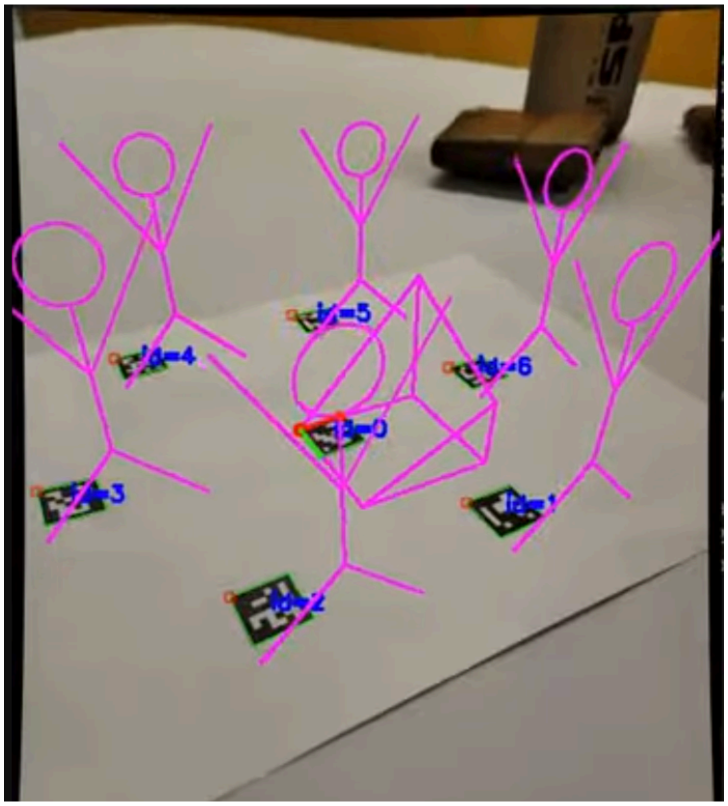


Fig6: Virtual Object

Video showing the AR demo - https://www.youtube.com/shorts/YXEigihUHyo

The video is tagged as a short on youtube because of its short duration and aspect ratio.

Task 7: Detect Robust Features

For this task, we decided to detect SURF features using the OpenCV surf function. We chose the wallet as our pattern as there was variation in color, fraying of the fabric and logo. It also had a decent amount of edge for edge detection. We found that SURF features captured variation in color, pattern. The edges were also captured. As the camera moved around, the features did not move or change much. This also helped us verify how SURF features are rotation, translation and scale invariant.

We experimented with different values for the thresholds and found that increasing the threshold reduced the number of features being detected in the frame.

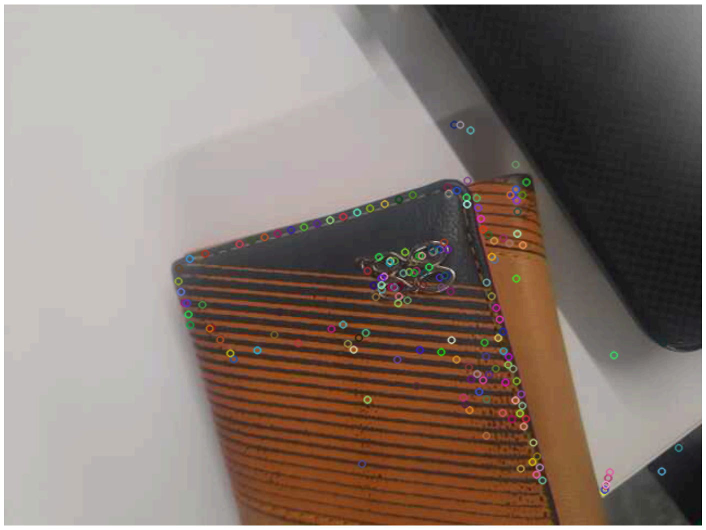


Fig7: Detecting SURF features in image.

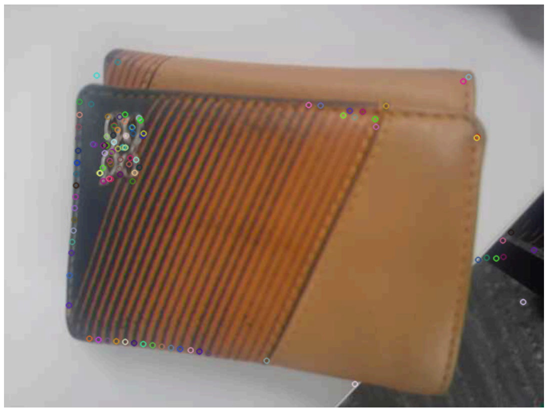


Fig8: Detecting SURF features with higher threshold

After extracting features, we can use K-nearest neighbors (KNN) for feature matching. By comparing these features between frames, we can identify corresponding points, assuming they lie on the same surface or target object.
Once we have matched features, we can utilize algorithms like solvePNP to estimate the camera pose relative to the scene. By considering the matched feature points and their corresponding 3D positions, solvePNP can calculate the camera's position and orientation in the real world.
Throughout this process, it's essential to maintain the origin and transformation information obtained from successive frames. This allows us to continually update the camera pose and track any movements or changes in the scene accurately. We can use this information and create virtual objects

Extensions

Extension 1: Testing several cameras and compare quality of calibration

As an extension, we tested out cameras of various laptops and phones. Here we are using the quality of calibration to compare the different cameras.

Device	**Reprojection Error**
Dell G3 357 Laptop Webcam	0.19853
Lenovo Legion 5i Pro	0.09542
Samsung Galaxy S22	0.138797
Samsung Galaxy M31s	0.211508

On an average, we found that the reprojection error was slightly higher when we used a phone camera as an input as opposed to laptop's inbuilt webcam

Extension 2: Implementing and detecting multiple features

As an extension for task 7, we are also implementing SIFT and Harris Corner Detections. As observed from the outputs, the Harris Corner detector performs really well to detect edges and corners.
SIFT is used to detect overall features in the image which are invariant to rotation, translation, scale and illumination. SURF gave us the best results out of the three and effectively captured the most relevant features.

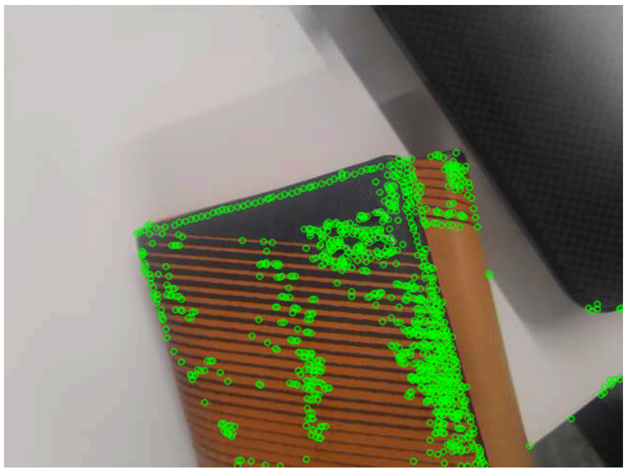


Fig9: Harris Corner Detector

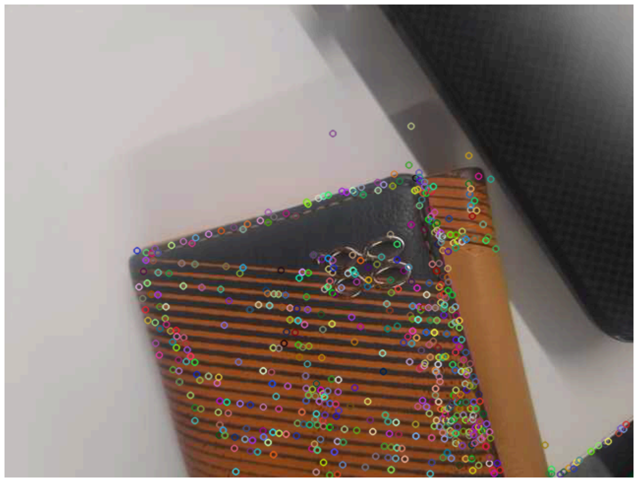


Fig10: SIFT Features

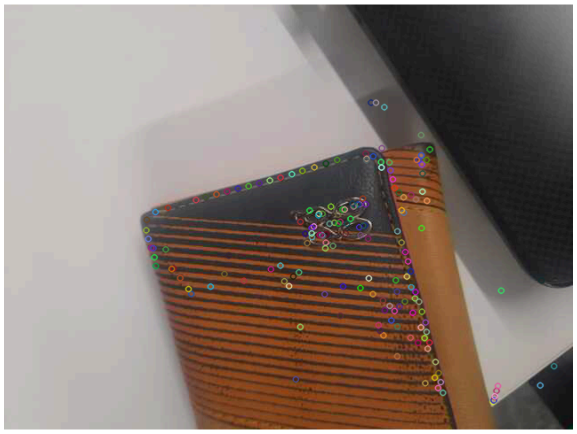


Fig11: SURF Features

Extension 3 - Multiple Targets

We used multiple aruco markers in our setup and made the code so that multiple targets could be used regardless of the number of targets. The code detects each target and places an object on each one. A unique object is placed on aruco marker with id 0. All the ids have the same object. The screenshot from task6 showcases this functionality

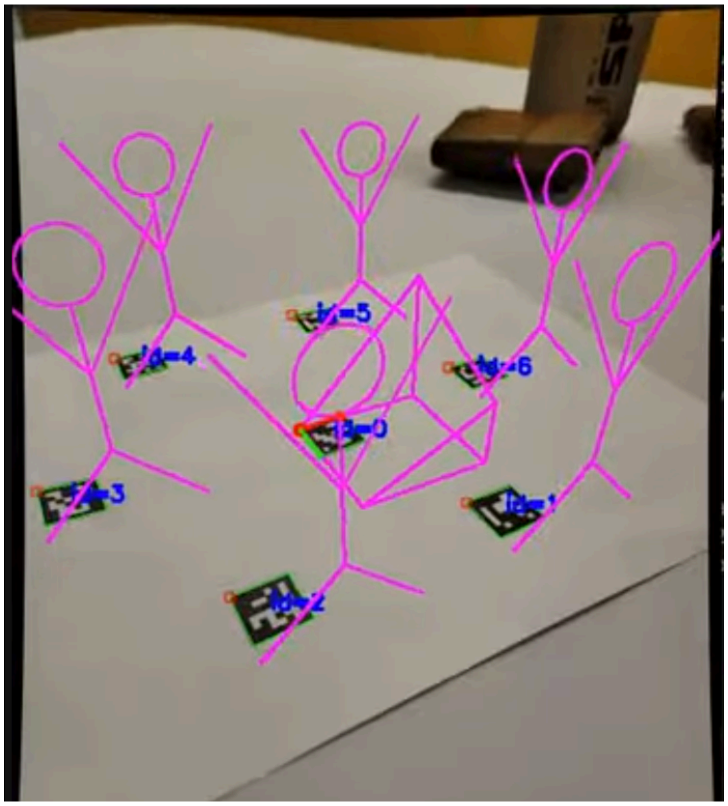


Fig12: Multiple Targets

<u>Extension 4 - Creative Virtual Environment</u>

We spent a good amount of time making our virtual scene starting from generating a specific array and pattern of aruco markers to designing the objects. The campsite with dancing people ￼was the idea of a creative environment for us

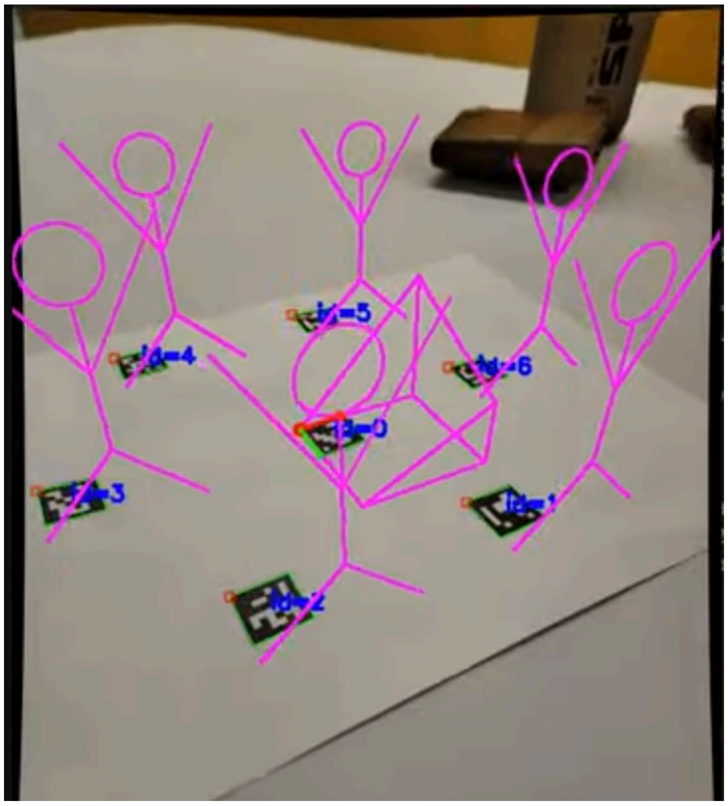


Fig13: Virtual Environment

Reflection

This assignment was really fun and helped us understand camera projection and the transformations better. We learnt how different flags in the built-in OpenCV functions can affect the stability of the virtual object. We also understood how different features such as SURF and SIFT work. We had to rebuild OpenCV a bunch of times to use the functions. Along the way we also learned how functions like solvePnp and projectPoints worked. We also got to familiarize ourselves with the aruco library in opencv and its functions. Overall this assignment taught us a lot of fundamental things like cameraPose estimation that have a variety of applications in robotics and other fields too.

Acknowledgment

We would like to thank Prof Bruce Maxwell for helping us grasp these concepts quickly. His in-class notes were very useful for solving this project. We would also like to thank TA Erica Shephard for helping us debug small bugs in our implementation

References

1. CS5330 Class Recordings and Lecture Notes
2. StackOverflow for debugging
3. OpenCV Documentation - https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html
4. https://docs.opencv.org/2.4/modules/nonfree/doc/feature_detection.html