[[1]](#footnote-1)

ASUforia Augmented Reality Framework

Sarvesh Patil, Shubham Nandanwankar, Achal Shah  
Arizona State University (2018)

**Abstract—Augmented Reality domain, since it first surfaced on the technology scene, has captured the attention of the application development world and is undoubtedly the hottest trend in while proving itself as the next big transformative technology, enabled by advances in computer vision. Augmented reality combines the virtual world with physical environments by looking for correspondences and placing objects in view. The advances in Computer Vision have also introduced way in for open source library framework such as OpenCV which enabled developers to build or incorporate AR features in their applications. In this project we have built “ASUforia” AR library providing this functionality. The primary purpose of this work is to create a comprehensive framework to enable the use of augmented reality within mobile applications by building upon the components of vision library such OpenCV.**

*Index Terms*—Augmented Reality library, Camera2, FLANNBASED, OpenCV, ORB, PnP Ransac, pose estimation

# INTRODUCTION

A

UGMENTED reality or AR is one of the prime research domain for today’s mobile computing technology world. In this project we have attempted to implement one such Augmented Reality framework “ASUforia” using OpenCV. The key concept that is acting under the hood enabling this is called “pose estimation”. Use of open source library such as OpenCV is another key component in building process of this library. Although just having OpenCV is not enough for AR vision applications since OpenCV simply performs vision tasks such as feature detection & matching in image.

This is to be completed through authoring of a library that can be called upon by an Android application activity to complete the image feature detection and pose estimation, and then use this information to allow the developer to construct a virtual object in the camera preview screen. Furthermore, ASUforia library will be used for development of an application that will initiate a camera preview, and then use the AR library construct to draw a cube on a reference image that will be placed in the view of the camera.

We have made use of Android’s Camera2 API instead of traditional Camera API to make use of its rich classes exposed for variety of application development scenarios.

Finally we have also defined & implemented our callback method onPose() that is called when doing the pose estimation on camera stream root data.

# Design & Implementation

The library & subsequent application design is comprised of multiple components that work together to produce the desired augmented Reality effect. The key components to design are Camera2 API, OpenCV for pose estimation, callback method implementation exposed to developer.

The User interface here just consists of an image preview pane for the user. On opening application, it first asks user for the permission to the user device’s camera. After getting the permission, the layout xml file generates two surface view to hold the image stream captured using the device camera & reference image. In design, surfaceTexture class object is used with camera for handling same.

The To implement all the image capturing from camera, we are using Android’s camera2 API. We use system service manager called Camera Manager in this to detect, characterize, and connect to the camera Device. Camera2 API captures frames in the YUV-420-8888 format which we convert to different formats as needed at different stages of processing. YUV-420-8888 image format is used for its advantages such as lower bandwidth that allows for faster processing through the image pipeline and so a consistent frame rate for real-time image processing applications. Thus making it an ideal choice here for us.

The CameraCaptureSession has StateCallback class which is a callback object which can be used to get update about Camera Capture Session state. Another important class we make use of is ImageReader which is at heart of accessing image from the surfaceview. Methods such as acquireNextImage() allows us to access images queued up in buffer to get seemless image capture.

Once we capture the image, we work with the frame using number of OpenCV methods exposed for image processing for respective tasks. The workflow for this is to detect & match keypoints & descriptors computations between reference & scene imagery. We are then implementing PnP Ransac algorithm to get the pose estimation once we get the 2D and 3D correspondences.

## User Interface

All the UI components design is done in *activity\_main.xml* file. We have used RelativeLayout approach for designing our UI. Two surfaceView components are used within this to hold scene imagery & reference imagery.

## Main Activity

This is the main launcher activity that is presented to user on opening the application. MainActivity.java file handles the all main activity initializations & operations.

In this application, MainActivity serves as the developer point of view implementation of using ASUforia library. After finishing initializations at start, the onCreate() callback itself calls onResume() callback which is pivotal in behavior of how the application works with camera. Through startEstimation() method, that takes care of image processing tasks through callback methods defined in ASUforia library.

Similarly we use onPause() callback that invokes endEstimation() method of library to close & yield camera to any other application that is invoked on device. This also ensures to release all threads that are running the background tasks for making sure camera feed works properly.

## ASUforia library File

This library file contains the complete implementation of ASUforia AR library, & subsequent implementations for OpenCV processing & callback methods.

The parametric constructor of ASUforia takes in two Bitmap images as arguments & sets first one as reference image while other as scene image for all further processing. The pose estimation process is initiated every time by calling startEstimation() method from onResume() callback with associated view object to provide camera preview. ImageReader object is used in the frame\_refresh() library method through the onImageAvailableListener callback for achieving same. Everytime surfaceCreated callback is called, we invoke a background thread which runs this frame\_refresh() method to continuously capture frames. For every captured frame, onImageAvailable callback calls startImageProcesing() method which then performs feature detection & matching compute part using OpenCV. onPause() callback calls endEstimation() method of library to close & yield camera to other application & release all threads running the background tasks for camera. Here as shown in fig 1, we are using one surfaceview to preview incoming camera stream & on that surfaceview drawing an rectangle on canvas.

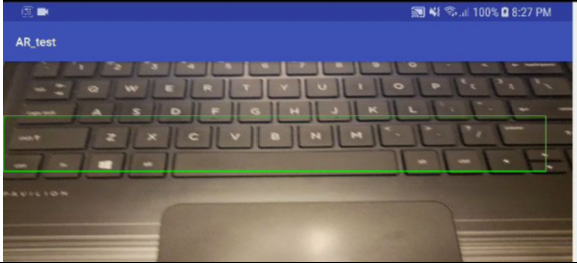


Fig 1. Drawing on canvas with other surfaceview

As shown in Fig 2, we have implemented interface Poselistener to register the onPose() callback method. This callback is exposed to be used in main\_activity for application implementing this interface.

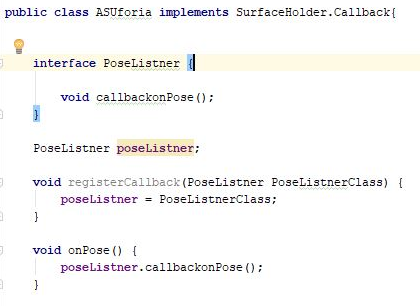


Fig 2. PoseListener Interface implementation

In OpenCV implementation part of library, first class Method startImageProcesing() takes in two imageviews set from layout. We are first converting the bitmap image to OpenCV mat format for all further image processing tasks that are carried out in initializeOpenCVInstances() method. We are using ORB detector on the received image mat in this method. Using it’s detect() & compute() method, we find out the keypoints & descriptors in reference image. After that using Imgproc.cvtColor() method of Imgproc library, we first convert the RGBA format image to grayscale image to get the improvement in processing speed. After this step, we are using Features2d.drawKeypoints() method so we can draw the keypoints that were computed in previous steps onto our image mat before converting it back to bitmap again.

In next step, we make call to match\_feature() method with scene image as argument for this part of computing. In match\_feature() we again follow the same steps of first converting bitmap scene image to mat format & then from RGBA to grayscale. Then we detect & compute keypoints and descriptors respectively for the scene image mat as well. Descriptors are converted to CV\_32F format. We are using knnMatches arraylist to hold the list of matched descriptor points in computed descriptors of both images with k=2. Another list listOfGoodMatches is used to hold all the extracted “good” point matches from our matches with 0.7 set as threshold distance. Finally Features2d.drawMatches() is used to draw the selected matched features in reference & scene image mats.

Once with the 2D and 3D correspondences we have to apply a PnP Ransac algorithm in order to estimate the camera pose. In this we make use of one of four PnP methods: ITERATIVE. *Ransac* is a non-deterministic iterative method which estimate parameters of a mathematical model from observed data producing an approximate result as the number of the iterations increase. After applying the *Ransac*, all the *outliers* will be eliminated to then estimate the camera pose with a certain probability to obtain a good solution. We are computing 2D point from the scene & 3D point from ref image first storing them in list\_points2d\_scene\_match & list\_points3d\_model\_match respectively. Then we are using estimatePoseRANSAC() function which gives estimates of the rotation and translation matrix given a set of 2D/3D correspondences, the desired PnP method to use, the output inliers container and the Ransac parameters.

# Evaluation & benchmarking

All evaluations & benchmarking tests have been performed on NVIDIA shield tablet & Samsung Galaxy S9 as base hardware. All the subsequent observations have been listed in this section.

ASUforia library was implemented as AR framework with callbacks exposed to developer for usage in respective application development. We were able to successfully capture the camera stream & preview it on surfaceview in UI. PoseListener interface was also implemented to define onPose() callback method.

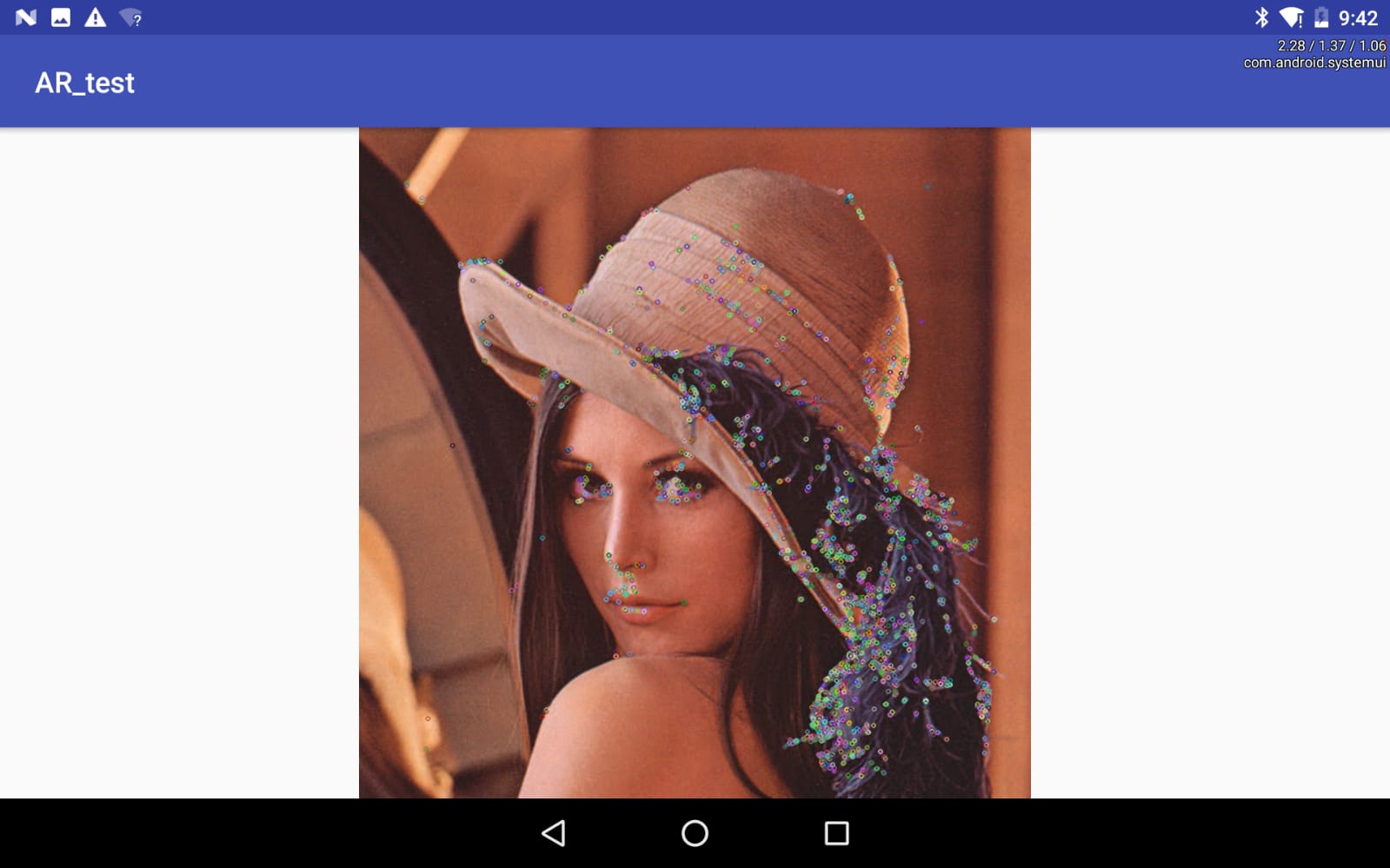


Fig 3. Feature detection keypoints on scene imagery

The feature detection implementation results in detecting feature points on sample scene image as shown in Fig 3.

Also we managed to get the feature matching for two images by detecting & matching descriptors. One example of such feature matching is shown in fig 4

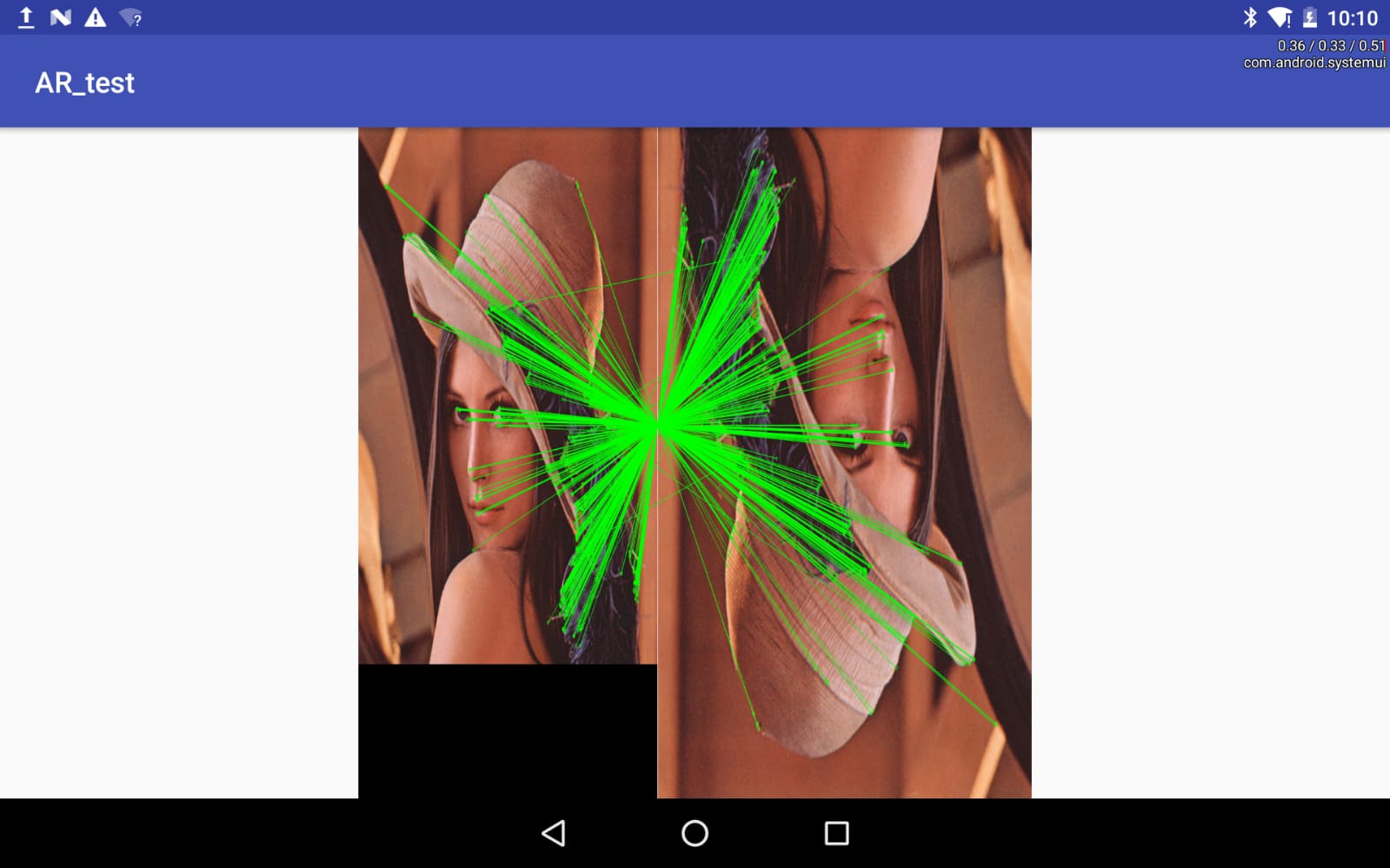


Fig 4. Feature matching for scene & reference imagery

# Some Common Challenges & Mistakes

Along the development of this project, we encountered few bugs/issues which we have listed in this section. Getting OpenCV to work flawlessly was a challenge so much so as to slow down the development of all components of project. For the first time users of OpenCV it was significantly harder to get it to work using online resources only.

. In initial stages integrating Camera2 API was found to be difficult task due to complexity of API 7 the way it is implemented in this project.

Also pose estimation computations & corresponding UI integration was a challenging portion of the project work. To tackle this we initial got it working with two static images acting as scene and reference imagery & then proceeded to computing the feature detection & matching for two images.

# Conclusion

## Observing the results, we conclude that we have successfully integrated device camera stream to preview on surfaceview using the Camera2 APIs & draw the same on canvas view.

## We implemented OpenCV libraries successfully to operate on the received camera frames to carry out feature detection & matching for reference & scene imagery.

## Finally we also successfully implanted our callback method for implementing pose estimation functionality with ASUforia serving as a single AR library to be used by developers in their application design by exposing these calls.

Acknowledgement

We owe a great thanks to everyone who helped and supported us during the completion of this project.

Our deepest thanks go to Prof. Robert LiKamWa, Professor for the subject Mobile Systems Architecture who acted as our mentor and guided us throughout this project. We thank him for giving ideas and briefs for the work that had to be done.

A special thanks to Jinhan Hu, Teaching Assistant, Mobile Systems Architecture, for constant help and guidance in the labs as well as discussion group.

.

References and Footnotes

## References

1] Camera2 API : https://developer.android.com/samples/Camera2Raw/src/com.example.android.camera2raw/Came

ra2RawFragment.html

2] OpenCV

<https://github.com/akshika47/OpencvAndroid>

https://medium.com/@siantlords/opencv-and-android-ndk-integration-in-android-studio-883a810

3] FLANN matcher

<https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html>

1. This project was submitted for evaluation on December 6th, 2018 at Arizona State University.

   All development done for this project is licensed under Arizona State University. Authors Sarvesh Patil, Shubham Nandanwankar & Achal Shah were with Arizona State University during the development of this project (2018). This project was developed under the guidance of Professor Robert LiKamWa of Arizona State University (2018). [↑](#footnote-ref-1)