Paul Dougherty, Raj Mehta, Aarsh Patel, Siddhant Shirguppe, Yiming Wang

QR code pose estimation for overlaying AR Objects

CS415 - Computer Vision-1

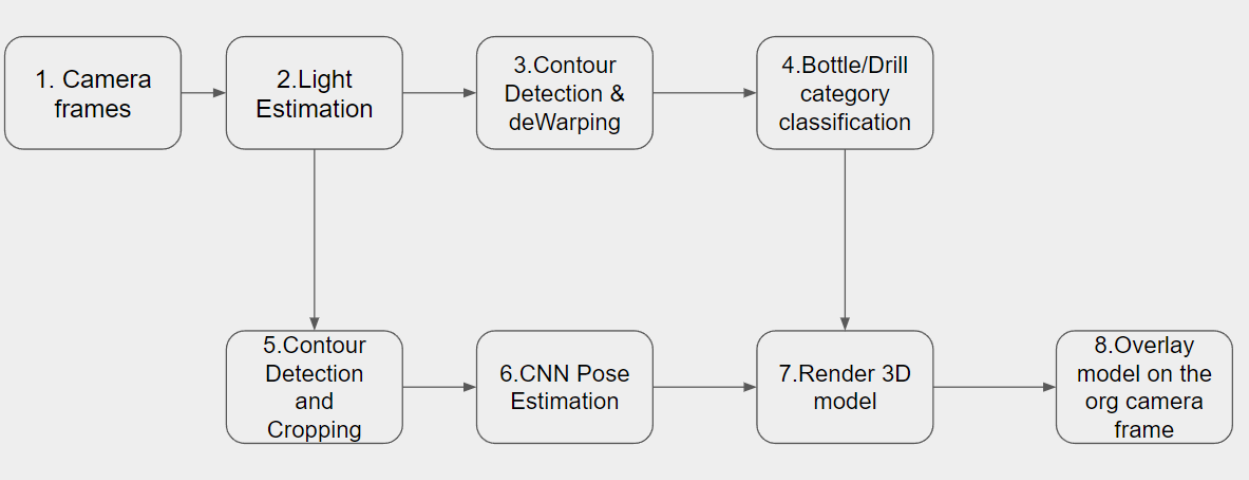
Problem Statement

With the introduction of the Apple Vision Pro, we are headed towards an era of spatial computing. QR codes are now ubiquitous everywhere. We aim to use these existing QR codes to display Augmented Reality Content on top of it. AR applications have exciting use cases in the fields of advertising, health care, education, etc. However traditional methods are not efficient when the QR codes are occluded and the feature points are lost. we aim to address this issue using a CNN-based approach.

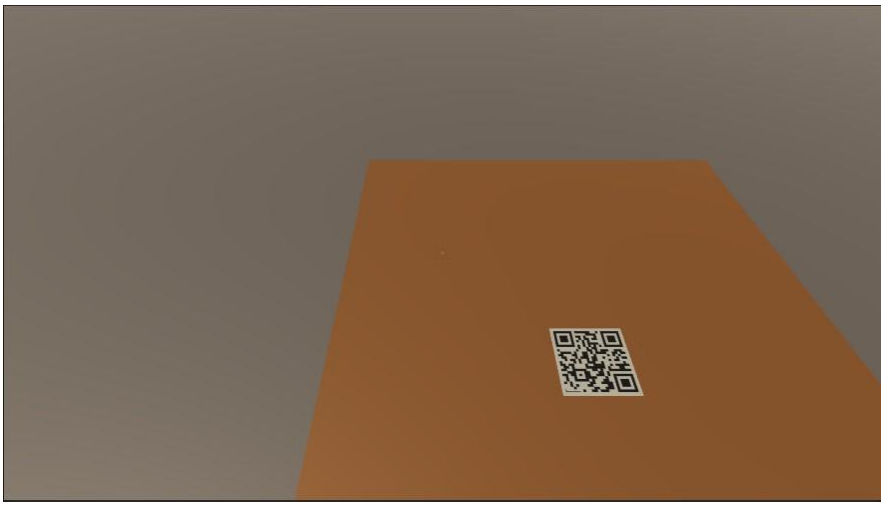
Objective

Build an Augmented Reality application to detect a QR code in the user's surroundings using the camera frame, read the data embedded in the QR code, and finally display a pose-accurate 3D model on the QR Code.

Project Pipeline



1. CAMERA FRAMES AND POSES:

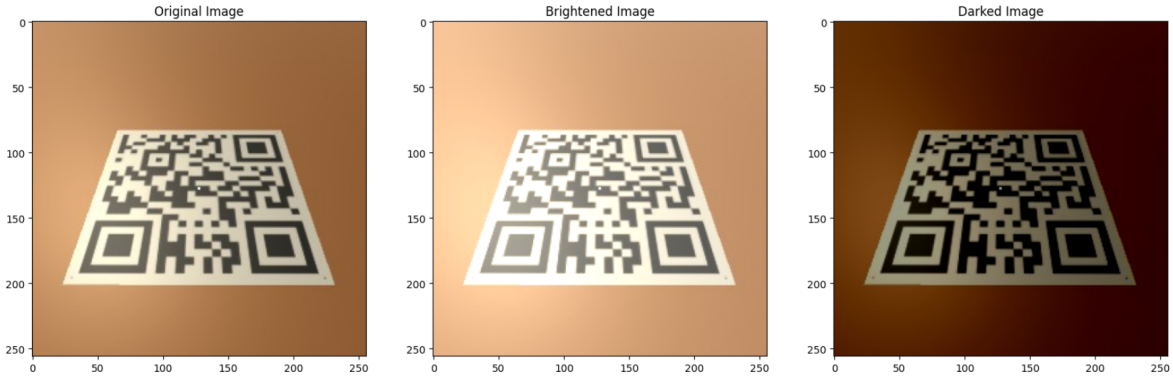


Original camera frame

Cropped frame to send to CNN

The first step is generating synthetic camera frames with associated camera poses using a scripted camera path in Unity. The original image is cropped so that the QR code will be isolated and the camera frames can be converted to the domain space of the CNN model.

2. Light Estimation



The Light Estimation part is used to preprocess the QR code pictures. When QR codes are scanned in our reality, there might be some different conditions affecting the pictures we take. One of them will be the lightness. Too dark or too bright pictures will increase the difficulties of scanning the QR codes.

To solve this problem, openCV library is used to recognize and adjust the brightness of images. The pictures are converted into grayscale and the average brightness is calculated. If the mean brightness is too low or too high, then based on the predefined thresholds, there will be a warning message “Bad Lighting Detected in the image!”. And then the images are converted into HSV (Hue, Saturation, Value) color space, which helps brightness manipulation. The current brightness is calculated with the mean value and if it is much lower or higher than the target brightness, then it will be adjusted automatically.

```
Warning: Bad Lighting Detected in a frame!
Adjusted brightness for a frame!
115.9736574074074
Warning: Bad Lighting Detected in a frame!
Adjusted brightness for a frame!
114.95913065843621
Warning: Bad Lighting Detected in a frame!
Adjusted brightness for a frame!
```

Also, the brightness adjusted is tested in the video. We could adjust the brightness for each frame extracted from the video.

3. Contour Detection & Dewarping

Image Preprocessing and Edge Detection

The initial steps involve preparing the input image for further analysis. Grayscale conversion simplifies the image, and GaussianBlur is applied to reduce noise, ensuring a cleaner input for subsequent operations. The Canny edge detector is then employed to identify significant edges in the image. This series of preprocess

Original Image

Grayscale Blurred Image

Detected Edges

Contour Detection and Polygon Approximation

We then proceed to identify contours in the edge-detected image using the findContours function. These contours, representing the outer boundaries of distinct shapes in the image, are essential for subsequent analysis. To streamline the complexity of these shapes, the Ramer-Douglas-Peucker algorithm is applied, approximating contours to polygons. To enhance the accuracy of the polygon representation, a filtering mechanism is implemented, specifically targeting the removal of close vertices. This meticulous process is crucial for refining the contours and ensuring a more precise representation of the underlying shapes. Notably, the code focuses on selecting only the outermost contour during this stage, simplifying subsequent validation steps. The resulting polygons undergo validation based on both vertex count and an area threshold, with a keen emphasis on those likely to represent QR codes.

Outermost contour

Approximate corners

Transformed Image

Perspective Transformation and Output

Upon identifying valid polygons, the code calculates a perspective transformation matrix based on the polygon's vertices. This matrix is used to apply a perspective transformation using the cv2.warpPerspective function. The transformed image is then returned, providing a clear and undistorted representation of the originally distorted region. The use of the “transformed_image_size” parameter ensures a consistent output size, facilitating further

processing. Additionally, there are commented-out sections in the code that, if uncommented, would provide visualizations of contours and corners, aiding in the debugging process.

4. Rotate and Redundant Classification

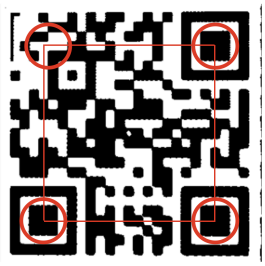
The purpose of this module is to send the correct QR code classification of bottle/drill to the 3d rendering engine, to display the called-for image. The strategy behind this module is to have a redundant classification/identification of the QR code, in order to deal with the diminishing quality of synthetic VR images being taken at increasingly distant or remote angles.

Once the rotate/classify module receives the transformed/de-warped image as a flat QR code, the first step is to classify/identify it using the QRCodeDetector function of OpenCV. Then a binary threshold is taken, and rotation to final position is achieved by taking the L1 norm of the four quadrant centers, three of which are black. The highest norm gets rotated to lower-right.

Finally the redundant classification is achieved by taking the L1 norm of an area of difference between the two QR codes (of which the module must have knowledge, in this two-class problem scope). The higher norm is the class for which that area is white instead of black. Because the L1 norm is more consistent, we use this.

From Example 1 (PoC):

Receive as flat square



Take L1 norms, rotate highest to lower-right

Take L1 norm of diff area

Classification/Identification error rate

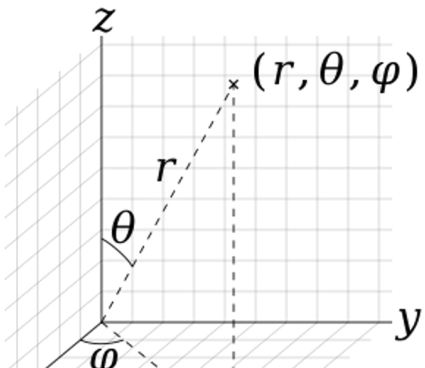
QR code pair	QRCodeDetector (bottle)	QRCodeDetector (drill)	L1 norm (bottle)	L1 norm (drill)
Example 1	3%	Same 3% (arbitrary example)	3%	3%
Example 2	16%	11%	11%	1%

Note 1: Example 1 pair is a PoC/trial generated from a single QR code by "manually" removing black areas, therefore the decoded string is the same for both. Example 2 pair have different strings encoded.
Note 2: Example 2 pair come from a more distantly captured sample, which is the reason for the increased error.

5. CNN Pose Estimation

In order to render the 3D object on top of the QR code, we must know where the camera is with respect to the QR code in the real world coordinates. We can first assume that the QR code is at the origin of our coordinate system and that the camera always points directly at the QR code. Then, we only need to estimate the following three quantities:

1. The Azimuthal angle (phi)



2. The Polar angle (theta)
3. And the radial distance (*r*)

Then, we can train a Convolutional Neural Network that takes in the normalized, grayscale form of the images seen by the camera and outputs the values for each of them.

So far, we have trained the CNN on data generated such that the radial distance and the polar angle are kept constant and the azimuthal angle is changed, i.e., the camera is moved around the QR code in a circle. So, the only output of the CNN is the azimuthal angle.

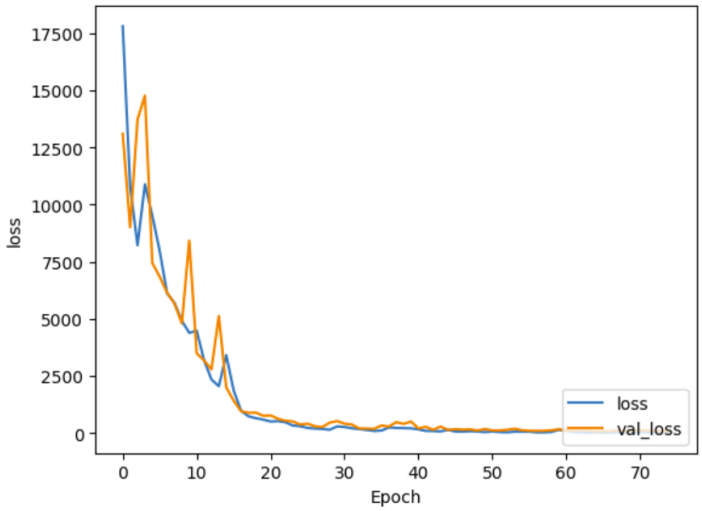
Architecture of the CNN:

Our TensorFlow-Keras CNN has two convolutional layers and two max pooling layers. The kernel size for the convolutions is (3,3) and for the Max pooling layer is (2,2). We used 32 filters and 'same' padding in the CNN.

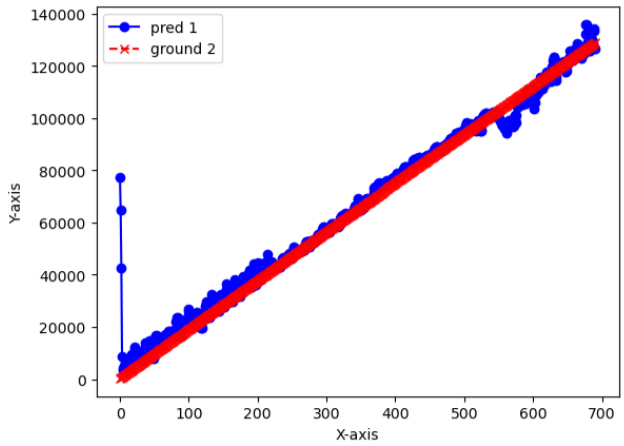
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 86, 86, 32)        320

 max_pooling2d_12 (MaxPoolin  (None, 43, 43, 32)       0
 g2D)

 conv2d_13 (Conv2D)          (None, 15, 15, 32)        9248

 max_pooling2d_13 (MaxPoolin  (None, 7, 7, 32)         0
 g2D)

 flatten_6 (Flatten)         (None, 1568)              0

 dense_12 (Dense)            (None, 64)                100416

 dense_13 (Dense)            (None, 1)                 65

=================================================================
Total params: 110,049
Trainable params: 110,049
Non-trainable params: 0
```

Training the CNN:

The CNN was trained for 75 epochs with batch size of 32 for an initial learning rate of 0.03 given to the Adam optimizer. MSE loss was used to optimize the parameters and the model was saved using a Callback whenever the validation loss was reduced. The following training curves were obtained for the loss:



The final model had a training loss of 21.56 and a validation loss of 60. The following graph shows the prediction error curve for all angles from 0 to 360 degrees:



As we can see from the graph, the model performs reasonably well at predicting the azimuthal angle from the given image. However, there's still room for improvement as there are some outliers and some jitteriness in the predictions. As the professor suggested during the presentation, we can maybe improve this by binning the angles from 0 to 360 degrees into 10 or 36 bins and convert this from a regression task to a classification task and use Categorical Cross Entropy loss instead of Mean Squared Error loss during the optimization process.

The next steps would be to generate data for different polar angles at different radial distances and then train a general CNN model that can predict all the three quantities (azimuthal and polar angle, radial distance) from the image seen/generated by the camera.

6. Render the 3D model

Firstly we get information about which model to render from the Bottle/Drill classification module. Then we get information about the pose of the model to render using the CNN pose estimation module. we extract the color map and the depth map. Then we binarize the depth map to be used as an Alpha mask.

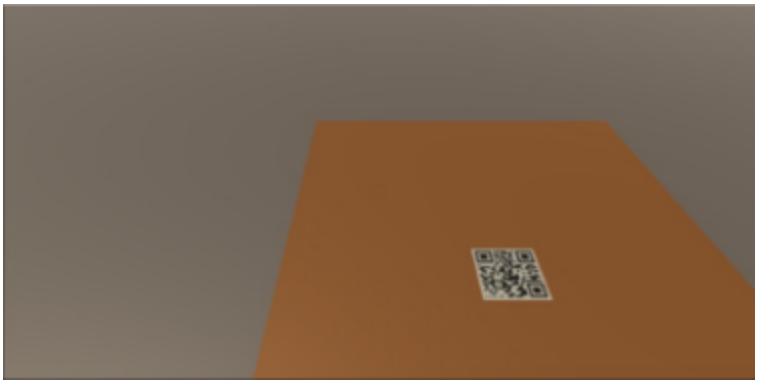
COLOR MAP

DEPTH MAP

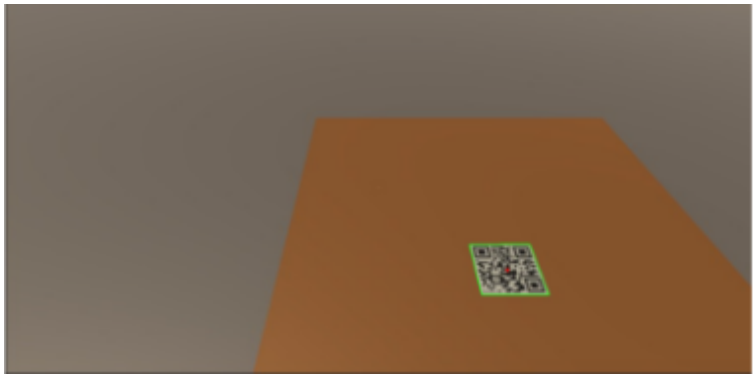
BINARIZED DEPTH MAP

7. Overlay the 3D model on the original frame

We get the centroid of the QR code and pass the cropped window to CNN model to get the pose. The pose correct 3D Model is overlaid on the cropped window.We replace the cropped window back on the original camera frame for the final output.



Original camera frame



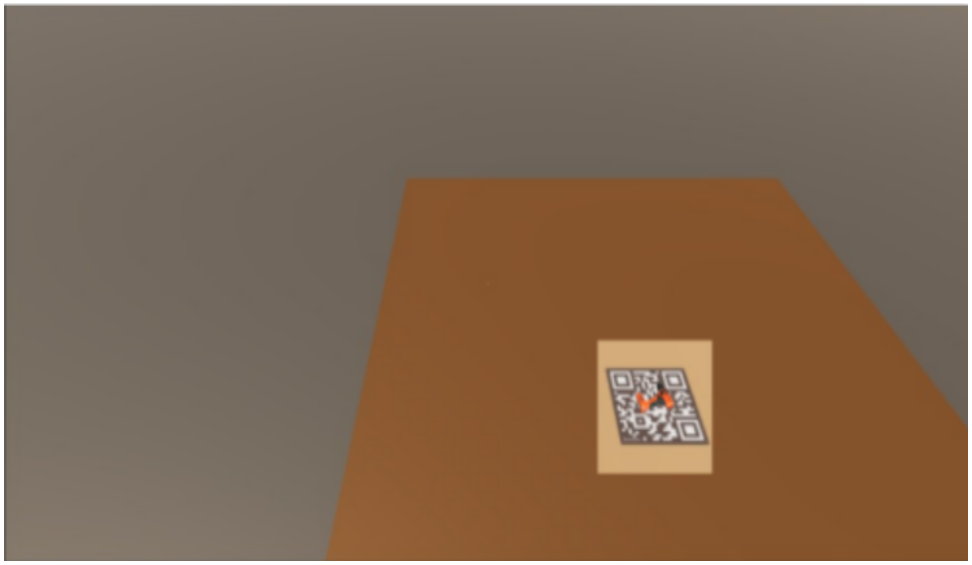
Detect the contours and centroid of the QR

Crop the image for input for CNN

Overlay the 3d model on the cropped image

Angle A

Angle B



Overlay the cropped image back onto the original camera frame

8. Advantages over Traditional Approach

1. Tracking is never fully lost. The model tries to always give the best possible pose.
2. Pose can be estimated even if the QR code is partially occluded.

9. Limitations

1. The poses can be jittery due to the Inference model.
2. It requires training CNN models for the individual QR codes.

10. Future Work

1. Add Polar Angle and Scale estimation using the CNN approach.
2. Adding ambient occlusion and shadowing effects.
3. Reducing the jittering effects during rendering the 3D model.