Computer Vision Class Paper

Haruya Ishikawa

July 20, 2018

Abstract

With technologies such as ARKit and ARCore released recently, augmented reality (AR) has become easily accessible to developers [1, 2]. Recently, more and more applications are released on application stores that incorporates AR such as the infamous "Pokemon GO" along with social media such as Instagram, Snapchat, and Facebook Messenger [3]. AR has been around for a while since the mid 2000s and with better hardware such as GPUs, AR on mobile devices is now accessible [4]. In this paper, I created a simple 2D AR application that swaps business card images using a video camera. This application uses Python 3.5 and OpenCV 3.1 [5, 6].

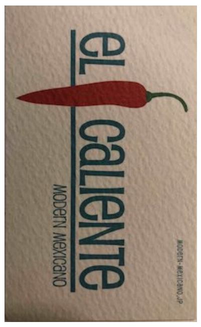
1 Introduction

2 Objectives

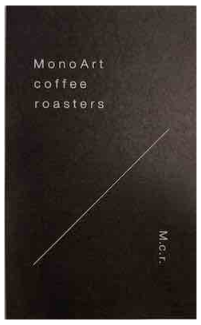
The objective of this application is to replace a business card in a video frame with another business card. In more details:

1. With a webcam, take a picture of business card.
2. Outline the image on the page with a rectangle, i.e., draw over the four sides as they appear in the image.
3. Match features in this area with each new image frame.
4. Replace the original business card a new business card with the appropriate homography.

For convenience sake, step 1 and 2 (outline/cropping of the initial business card) is removed with a more simple algorithm in this application at this time. The two business cards used throughout this paper is shown Fig. 1



(a) Reference business card



(b) Replacing business card

Figure 1: Two business cards used throughout the paper: where (A) is the reference business card (image that is detected and then replaced), and (B) is the replacing business card (image that replaces the reference business card).

where Fig. 1b is the replacing business card that replaces Fig. 1a that is the reference business card. Figure 1a will be called the "reference image" and Fig. 1b will be called the "replacing image" throughout the paper. The word "target image" refers to the video frame.

3 Acknowledgments

I would like to acknowledge this paper to Prof. Saito for teaching the basics of computer vision in his class.

4 Theory

The process of developing this application can be divided into several steps:

1. Identify flat surface in the image frame (from rendered video) that is the same as the reference image.
2. Estimate homography using the transformation from the reference image frame to the target image.
3. Derive coordinate transformation (projection) from reference to target coordinate system
4. Render the new image onto the target coordinate system

4.1 Recognizing Reference Image inside the Target Image

The first step is recognizing the target surface. There are many techniques that could be used to solve this problem, but feature based recognition is simple and the method that I had learned from this computer vision class. The steps for this method is:

- feature detection (detector)
- feature description (descriptor)
- feature matching

4.1.1 Feature Detection

In computer vision, the concept of feature detection refers to methods that aim to compute abstractions of image information which are given by points or groups of points in images. Feature detection in general means finding interesting points (features) in the image such as corners, templates, edges, and so on. Therefore by looking in both the reference and target images for features that are similar and stands out, those features can be used to find the reference image inside of the target image which is similar to how AR markers work. For this application, it is assumed that the same object is found when there are enough features that are matched.

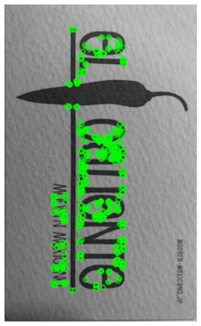
Conditions:

- Reference image should show only the business card the is used for tracking
- Dimensions of the reference image should be given
- Feature in the images should be unique (such as corners or edges), and the object in the reference should be invariant

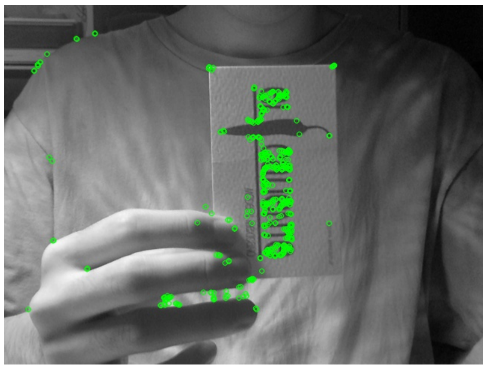
Figure 2 shows features extracted from the surface of the reference image in Fig. 2a and features extracted from the video frame in Fig. 2b. More about the feature used in the detection later on.

4.1.2 Feature Description

In pattern recognition, feature extraction is a special form of dimensionality reduction. When the input image to an algorithm is too large to be processed and it is suspected to be notoriously redundant, then the input image will



(a) ORB features in reference image



(b) ORB features in video frame

Figure 2: On the left, features extracted from the target image of the surface. On the right, features extracted from a sample video frame. Note how corners have been detected as interest points in the rightmost image.

be transformed into a reduced representation set of features (feature vector). This process of turning the input image into a set of features is called feature extraction. In short, feature extraction represents the interesting points found by feature detector and compare them with other feature points in the image to reduce the representation.

There are many algorithms that extract image features and compute its descriptors such as SIFT, SURF, or Harris. The one that we will use in this project is OpenCV's native algorithm, the ORB (Oriented FAST and Rotated BRIEF) [7]. This will produce binary strings as its descriptor.

The OpenCV code for this is shown below:

```
img = cv2.imread('model_image.png', 0)

orb = cv2.ORB_create() # initialize detector
kp = orb.detect(img, None) # find keypoints
kp, des = orb.compute(img, kp) # compute descriptors

img2 = cv2.drawKeypoints(img, kp, img, color=(0, 255, 0), flags=0)
cv2.imshow('keypoints', img2)
cv2.waitKey(0)
```

4.1.3 Feature Matching

After computing both descriptors for the reference and target image, matches are detected using hamming distance since ORB descriptors output binary strings. This is a brute force method and is used in this application since better methods exist for future improvements and is not the point of this application. A threshold of minimum number of matches should be set to decide whether an object was found or not.

The code for feature matching is shown below. Note that the threshold for matches is 20.

```
MIN_MATCHES = 20
cap = cv2.imread('target.jpg', 0) # target image
model = cv2.imread('ref.jpg', 0) # reference image

orb = cv2.ORB_create()
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True0)
kp_model, des_model = orb.detectAndCompute(model, None)
kp_frame, des_frame = orb.detectAndCompute(cap, None)
matches = bf.match(des_model, des_frame)
matches = sorted(matches, key=lambda x: x.distance)

if len(matches) > MIN_MATCHES:
    cap = cv2.drawMatches(model, kp_model, cap, kp_frame,
        matches[:MIN_MATCHES], 0, flags=2)
    cv2.imshow('frame', cap)
    cv2.waitKey(0)
else:
    print("Not enough points")
```

Figure 3 shows the closest 20 brute force matches found between the reference image and the target image. From the code and the figure, it can be said that the criteria to decide if the image was detected or not is met since for most of the cases, the ORB features are corresponding between reference and target images.

4.2 Homography Estimation

Once we have identified the reference surface in the current frame and have a set of valid matches we can proceed to estimate the homography between both images. This application needs transformation that maps points from the surface plane to the image plane. This transformation will have to be updated each new frame.

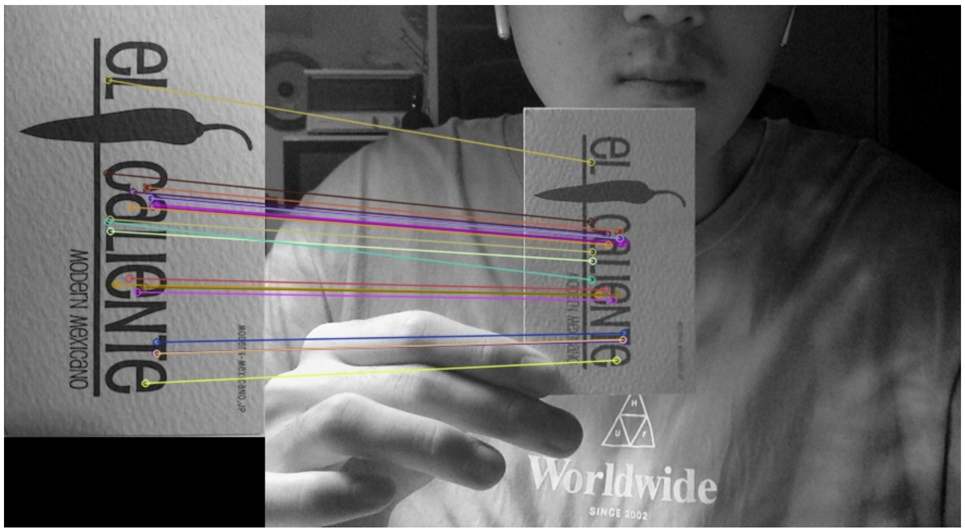


Figure 3: Closest 20 brute force matches found between the reference surface and the target image.

To find such transformation, since a set of matches between reference image and target image is known, an homogeneous transformation will satisfy this criteria. To find the homography between the reference surface and the image plane, homography matrix has to be calculated.

Given an object in a world coordinate system, a camera takes a picture at a certain position and orientation with respects to this world coordinate system. For ease of calculation, the camera coordinate system is assumed to be a pinhole camera, which roughly means that the rays passing through a 3D point $\boldsymbol{p}$ (on the object surface) and the corresponding 2D point $\boldsymbol{u}$ intersect at $\boldsymbol{c}$, the camera center. The coordinates in the image plane $\boldsymbol{u}(u, v))$ of a point $\boldsymbol{p}$ expressed in the camera coordinate system is computed as the following equation:

$$\begin{bmatrix} u \cdot k \\ v \cdot k \\ k \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^{cam} \\ y^{cam} \\ z^{cam} \end{bmatrix} \tag{1}$$

where f_u and f_v is the focal lengths and (u_0, v_0) is the projection of the optical center. The focal length is the distance from the pinhole to the image plane. Also, k is a scaling factor. This equation shows how the image is formed. However we do not know the point p in the camera coordinate system. Therefore another matrix transformation is needed to map points from the world coordinate system to the camera coordinate system. The transformation is given as:

$$\begin{bmatrix} u \cdot k \\ v \cdot k \\ k \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2}$$

This equation is simplified to become a one large projection matrix:

$$\begin{bmatrix} u \cdot k \\ v \cdot k \\ k \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3}$$

Since the points in the reference image always have its z coordinate equal to 0, the equation is then simplified even more than the equations above. It is clear that the product of z coordinate and the third column of the projection matrix will always be 0. By renaming the calibration matrix as $\boldsymbol{A}$ and taking into account the external calibration matrix is an homogeneous transformation, the equation is simplified as:

$$\begin{bmatrix} u \cdot k \\ v \cdot k \\ k \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} \boldsymbol{R}_1 & \boldsymbol{R}_2 & \boldsymbol{t} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4}$$

From this equation, it can be concluded that the objective homography matrix is

$$\boldsymbol{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}. \tag{5}$$

There are several methods to estimate the values of the homography matrix. The one used in this application is a method called random sample consensus or better known as RANSAC [8]. RANSAC is an iterative algorithm used for model fitting in the presence of a large number of outliers. Since there is no guarantee that all the matches that are found are actually valid matches, some false matches should be compensated (which will be the outliers). RANSAC is an estimation method that is robust against outliers. The main outline for this algorithm is:

- Choose a small subset of points uniformly at random

- Fit a model to that subset
- Find all remaining points that are ”close” to the model and reject the rest as outliers
- Do this many times and choose the best model

For homography estimation, the algorithm flow is:

- Randomly sample 4 matches
- Estimate Homography $\boldsymbol{H}$
- Verify homography (search for other matches consistent with $\boldsymbol{H}$)
- Iterate until convergence

In OpenCV, this whole process is:

```
# assuming matches stores the matches found and
# returned by bf.match(des_model, des_frame)
# differenciate between source points and destination points
src_pts = np.float32([kp_model[m.queryIdx].pt for m in
    matches]).reshape(-1, 1, 2)
dst_pts = np.float32([kp_frame[m.trainIdx].pt for m in
    matches]).reshape(-1, 1, 2)
# compute Homography
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
```

where kp_{model} and kp_{frame} are key points in reference and target images respectively. The threshold distance is 5.0.

The result of feature detection and homography estimation is shown in Fig. 4. The reference image is matched with the target image and the four corner points are extracted then transformed, which is shown by the blue outline.

4.3 Replacing the Reference Image

Replacing the reference image with another image is simple given the replacing images corner points. This can be done by computing the affine transform from the detected corner points in the target image and the replacing image’s corner points. This transform is then used to warp the replacing image to the target image.

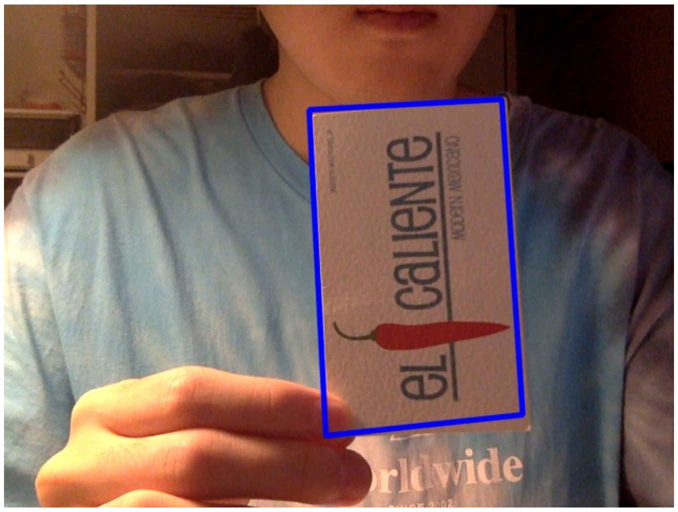


Figure 4: Render of the surface detection algorithm.

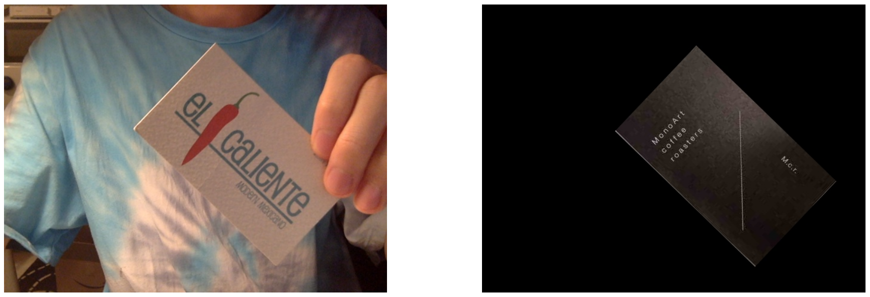

(a) Video frame (b) Warped replacing image

Figure 5: Result of affine transformation and warping the replacing image.

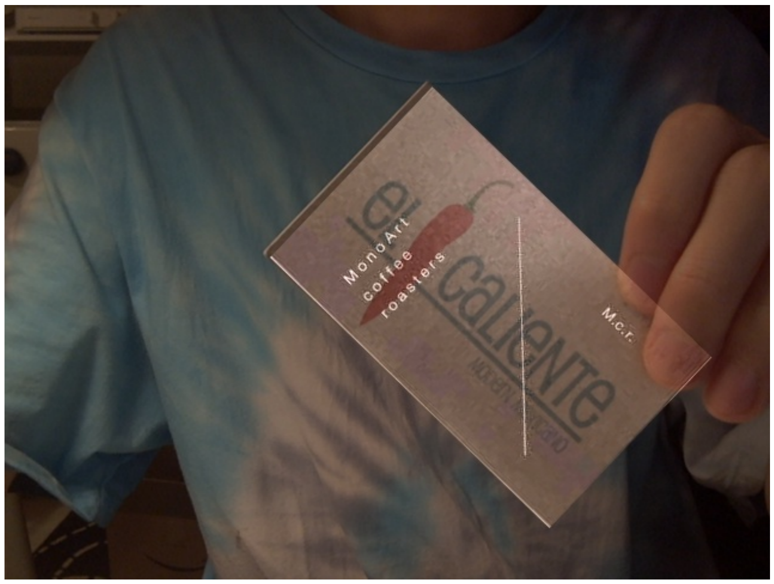


Figure 6: Result of the application.

5 Results

The result of the application is shown in Fig. 6.

6 Discussion

The difficulty of implementing this was calculating the homography matrix and implementing the homography estimation process. From Fig 6, one can argue that this is not ”replacing” the reference image. While this is true, I was not able to successfully create a mask of the warped image and replace the masked section in the target image with the replacing image. As a future improvement, I will implement a better algorithm for replacing the image. For now, blending the warped image with the target image is used as a replacing feature. Other than this, the algorithm works perfectly without failing. The hardware used in this experiment is a MacBook Pro (late 2015) with Intel core i5 CPU with integrated GPU.

7 Conclusion

In this paper, the application was able to detect the reference image using ORB feature detector and homography estimation was used to calculate the transformation to map the replacing image to the target video frame. This application presents the core functionality of "replacing" the reference image with the replacing image. Further improvements will be tested later such as a better detection scheme and better presentation of the "replacing" feature. Also a GUI based "cropping" feature to get the image for reference image should be implemented.

References

[1] Apple Inc., *ARKit - Apple Developer*, "Apple Developers", https://developer.apple.com/arkit/, July 20, 2018.

[2] Google Inc., *ARCore Overview*, "Google Developers, https://developers.google.com/ar/discover/, July 20, 2018.

[3] adweek.com, *Snapchat Introduces 3 New Capabilities of Its Augmented Reality Lenses With 'Shoppable AR'*, https://www.adweek.com/digital/snapchat-introduces-3-new-capabilities-of-its-augmented-reality-lenses-with-shoppable-ar/, July 20, 2018.

[4] Eric A. Taub, *Webcam Brings 3-D to Topps Sports Cards*, The New York Times, https://www.nytimes.com/2009/03/09/technology/09topps.html?mcubz=1, March 8, 2009.

[5] *python*, "Python.org" ,https://www.python.org/, July 20, 2018.

[6] *OpenCV*, "OpenCV.org", https://opencv.org/, July 20, 2018.

[7] *ORB (Oriented FAST and Rotated BRIEF)*, "OpenCV.org", https://opencv.org/, July 20, 2018.

[8] *Random sample consensus*, "wikipedia.org", https://en.wikipedia.org/, July 20, 2018.