Project 2: Augmented Reality

David Aparco Cardenas
Rosa Yuliana Gabriela Paccotacya Yanque
Rolan Alexander Valle Rey Sánchez

***Abstract*—*Augmented Reality* (AR) applications are gradually becoming part of our lives allowing us to interact with virtual computer-generated information on a real world environment. Some applications in our smartphones as *Pokemon Go* are having a great success due to the incorporation of this novel technology. In the present work, we use augmented reality to project an image over a rectangular surface following the same transformation as it moves throughout the frames of a video. In order to accomplish this task, we use ORB (Oriented FAST and rotated BRIEF) as a feature detector. Next, we determine the matches between the feature descriptors of the *target frame* and each frame in the video. Subsequently, we determine an affine transformation among the object points in both images using RANSAC. Afterward, we transform our *overlying image* using the aforementioned affine transformation and paste it over each frame in the video. At last, a video comprised of all the processed frames is created.**

I. Introduction

Augmented reality (AR) technologies provide an enhanced perception of our environment to help us see, hear, and feel in such new and enriched ways that will benefit us in fields such as education, maintenance, videogames, etc.

Following the definitions in [1], an AR system:

- combines real and virtual objects in a real environment;
- aligns real and virtual objects with each other; and
- runs interactively, in three dimensions, and in real time.

In the present work, we perform AR by pasting an image over each frame of a video. For this purpose, we use ORB [2] as keypoint detector and feature descriptor to extract features from the images. For each frame in the video we compare the detected features with the features detected in the target frame to determine matches between feature descriptors in each image. Figure 1 shows the target image and an example frame of video.

Next, from the set of matched points we perform the RANSAC method to determine a subset of matches consistent with an affine transformation. This subset will become our inlier set and will be subsequently used to find a better estimate of the transformation. Next, we find out which of the interest points in each frame of video corresponds to the corners of the target frame. At last, we create a mask with the transformed overlying image and paste it over each frame. At the end of the process, we will produce a video contained each processed frame. The complexity of this project relies on the fact that the target frame on each video frame will not be static. In other words, the camera will perform several kind of movements and our pipeline must track each movement and determine an affine transformation that maps our target frame to each frame of the video.

(a) Target image (b) Frame of video

Fig. 1: Target image and an example frame of video.

The basic flowchart of this procedure is show in figure 2.

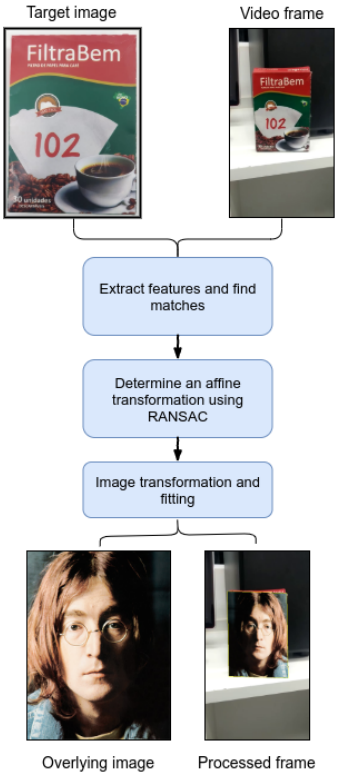


Fig. 2: Basic flowchart of the Augmented Reality (AR) pipeline.

RA: 230293 - d230293@dac.unicamp.br
RA: 263068 - r263068@dac.unicamp.br
RA: 230254 - r230254@dac.unicamp.br

The code used in this project was implemented using Python 3.7.3. Table I summarizes the libraries utilized in this work and its respective versions.

Library	Version
Python	3.7.3
NumPy	1.17.0
OpenCV-Python	4.1.1-dev

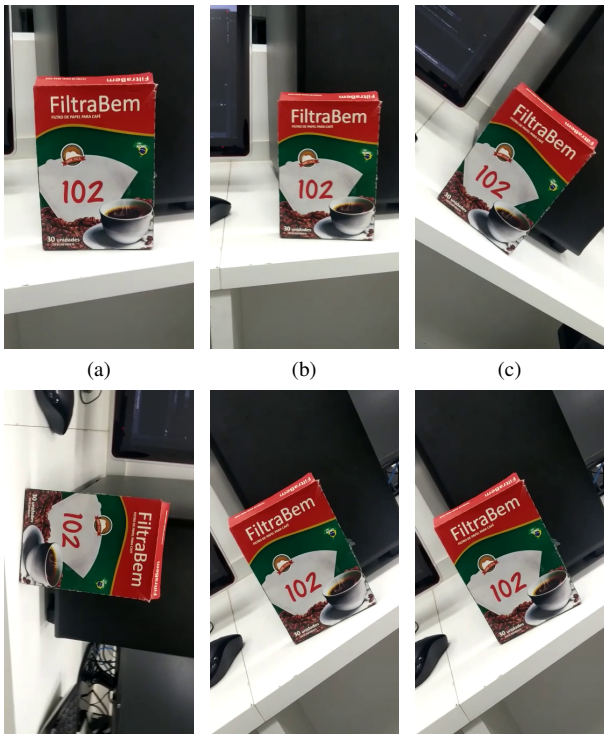
TABLE I: Python libraries used in this project.

The present report is organized as follows. First, a brief introduction was done in this section. The following sections will explain in detail, the procedures followed during the experiments for each section in the pipeline.

II. Input

As input we take the following items:

- A video containing our target frame. During the shooting of the video we moved the camera according to the requirements as shown in Figure 3:
  - one rigid displacement of the camera;
  - one zoom-in and one zoom-out;
  - one complete rotation clockwise and one conter-clockwise.
- A target image;
- An overlying image.



(a) (b) (c)

(d) (e) (f)

Fig. 3: Several frames from video.

III. Interest Points and Descriptors

Object detection relies on recognizing feature vectors that represents robustly the shape of an object.

Object detection is commonly used in applications such as robotics, self-driving cars and Augmented Reality(AR), as in this project. AR-applications rely on the live view of the real world, to know where to place a virtual object in a given scene, an this is done in real-time. So, it is needed to identify object in a really fast and efficient manner. For the goals of this project, the virtual object is an image and the algorithm we used to get key points and descriptors was ORB (Oriented FAST and Rotated BRIEF) due to its fast speed, unsusceptibility to noise illumination and image transformations such as rotations, scaling, and limited affine changes [3].

A. FAST

ORB uses FAST (Features from Accelerated and Segments Test) to detect key-points quickly by comparing the brightness levels in a given pixel area. That is to say, given a pixel, p in an image, FAST compares the brightness of p to a set of 16 surrounding pixels that are in a small circle around p (See Figure 4). Each pixel in this circle is then sorted into three classes, brighter than p ($> (p+h)$), darker than p($< (p-h)$), or similar to p, where h is a threshold. After classifying the pixels, pixel p is selected as a key point if more than eight connected pixels on the circle are either darker or brighter than p. This can be done also by comparing p to only four equidistant pixels in the circle,i.e pixels 1, 5, 9, and 13, instead of all the 16 surrounding pixels, in this case, if there are at least a pair of consecutive pixels that are either brighter or darker than p, p is a key point. This optimization reduces the time required to search in an entire image for key points by a factor of four, that is why FAST is efficient, however, the points returned by FAST just contain the location and not the direction of the change of intensity.

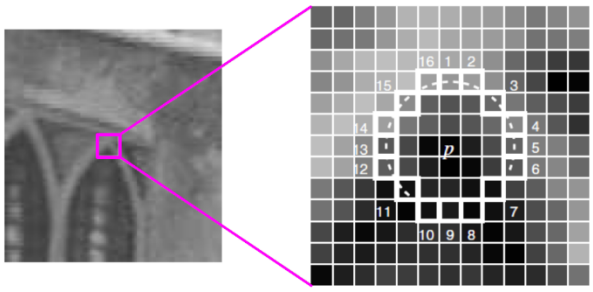


Fig. 4: Zoomed patch around a building's arch showing the pixels considered for SURF Detection of Key Points. Taken from OpenCV Documentation

B. BRIEF

Binary robust independent elementary features BRIEF creates binary feature vectors from a set of key points. Each key point is described by a binary feature vector that has a 128-512 bits string that only contains ones and zeros. Using binary vectors allows us to store and compute efficiently and quickly.

Basically BRIEF for each point, select two random points from a patch, the first pixel in the random pair, is obtained using a Gaussian distribution centered around the key point and with a standard deviation σ.Similarly, we got the second pixel but,centered in the first pixel and with a standard deviation of σ^2. Then, to construct the binary descriptor for the key point, we compare the brightness of the two pixels, if the first pixel is brighter than the second, we append to the vector the value of one, otherwise the value of zero. Subsequently, for the same key point we repeat this process the length we would like to have in our feature vector, e.g. for a 256 bit vector, BRIEF repeats this process for the same key point 256 times before moving onto the next key point.

C. ORB

ORB (Figure 5) makes a scale pyramid(blur and subsample), then detect keypoints at each scale using FAST, doing this makes ORB scale-invariant,next applies a Harris corner measure to find top N strongest points, later uses non-max suppression to discard interest points in adjacent locations and finally apply oriented BRIEF or Rotation aware BRIEF which makes it rotation-invariant.

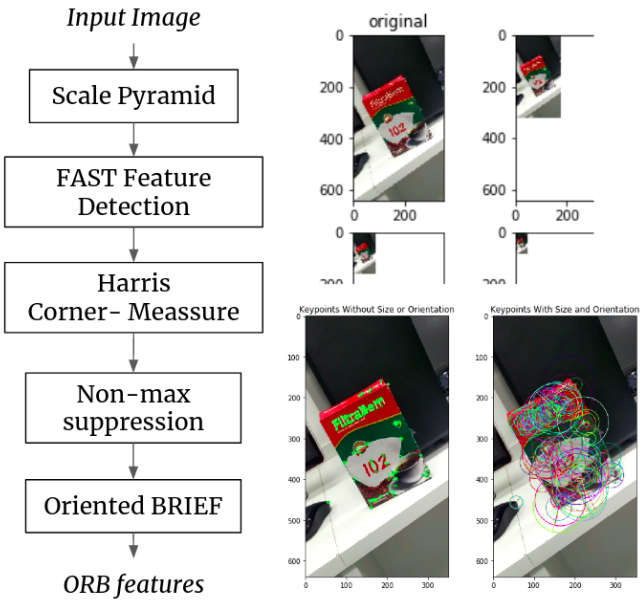


Fig. 5: ORB Pipeline

For this project, we used the default values given by OpenCV for the parameters: The number of levels is 8, the patch-size for BRIEF is 31 and the threshold when using FAST is 20.

D. Experiments

In this section, we used ORB to extract the features for the target image and each frame of the video. For each image, the ORB algorithm returned an array of 500 feature descriptors. Each feature descriptor consisted of 32 8-bits elements.

(a) Target image (b) Frame of video

Fig. 6: Interest points from target image and video frame.

IV. First Target Frame Detection

For this problem we shooted the object of interest with its corners coinciding with the corners of the picture, as it can be seen in Fig. 12. This is done to have as reference the corner pixels of the target image when trying to get the perspective transformation from the target images to the target image in the frame.

Fig. 7: Target frame

V. Match Hypothesis

For each pair of images (target image and frame of video) we found the candidate matches using the Hamming distance, since we are using ORB as our feature detector and OpenCV recommends to use this distance for binary string based descriptors like ORB, BRIEF, BRISK, etc.

The Hamming distance is defined as the number of positions at which the corresponding symbols are different. Since each

feature descriptor is comprised of 32 symbols, our maximum Hamming distance will be 32 where all symbols are different between two feature descriptors.

Fig. 8: Feature matching with threshold 40

Fig. 9: Feature matching with threshold 50

Also we multiply this distance by 2, in order to penalize higher distances. This procedure allows us define a threshold which we use to select the possible matches among the space of feature descriptors obtained. Therefore, now our Hamming distance will have as maximum value 64.

In order to perform the matching we performed a brute force matching comparison with each feature descriptor from the target image and each feature descriptor from the frames of the video. With a threshold of 40, that is, we allow only 10 symbols to be different between two feature descriptors we obtain as average 4 to 5 matches.

Meanwhile, if we use a threshold of 50, the number of matches increase but the quality of results decrease, that is, some matches are not correct. These experiments can be seen in Figures 8 and 9.

Hence, we decided to pick as threshold 40, because it provides the correct matches all the times.

VI. Affine Transformation Fitting

In this section our problem was to learn the affine transformation between our two images. As recommended, we assumed an affine transformation due to its simplicity. Therefore from our original coordinates (x, y) we want to find our transformed coordinates (x', y')

$$\begin{aligned} x' &= ax + by + c, \\ y' &= dx + ey + f, \end{aligned} \tag{1}$$

Equation 1 can be rewritten as is shown in Equation VI.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c \\ f \end{bmatrix} \tag{2}$$

Therefore, since our objective is to determine the parameters $a, b, c, d, e,$ and f

We can generalize Equation for three points as shown in Equation 3:

$$\begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ x_3 & y_3 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_3 & y_3 & 0 & 1 \end{bmatrix} * \begin{bmatrix} a \\ b \\ d \\ e \\ c \\ f \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \end{bmatrix} \tag{3}$$

Here we can name the matrix as follows:

$$XA = Y, \tag{4}$$

where X, A, and Y represent the corresponding matrix. Hence, the affine transformation can be computed by

$$A = (X^T X)^{-1}(X^T Y) \tag{5}$$

Therefore, this operations is implemented instead of using a solver as is required.

Our implementation of RANSAC begins by selecting 3 random points from the matches obtained in the aforementioned section. Once this points are selected, then we proceed to estimate an affine transformation matrix from these points from Equation 5. Subsequently, with this matrix we estimate the correspondence of estimated points using the root squared error of the differences. Then, we select a threshold of 1 to determine the inliers and outliers. We repeat this procedure using 2000 iterations in order to improve the quality of our estimation.

VII. Transform

After all iterations on RANSAC, we select an affine transformations A which contains the largest amount of matches. This final transformation is computed by generalizing Equation 3 as following:

$$\begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ x_3 & y_3 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_3 & y_3 & 0 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x_n & y_n & 0 & 0 & 1 & 0 \\ 0 & 0 & x_n & y_n & 0 & 1 \end{bmatrix} * \begin{bmatrix} a \\ b \\ d \\ e \\ c \\ f \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ \dots \\ x'_n \\ y'_n \end{bmatrix} \quad (6)$$

And we solve this equation in the same way as in Equation 5.

Then we use this equation to obtain the corresponding corners in each frame of video, that is, we transform each corner in the target image to the video frame. Therefore, we would end up with the corners of our target image in each video frame. The corners in our target image are $(0, 0), (0, w), (h, 0)$, and (h, w)

VIII. Augmentation

Once we got the corners of our target image in each frame of video, we proceed to determine the perspective transformation matrix using the corner of the target in both images. Afterward, using this perspective transformation matrix we apply the warp perspective in our overlying image and then we create a mask with it. Then, we add the mask to the frame of the video resulting the following image.

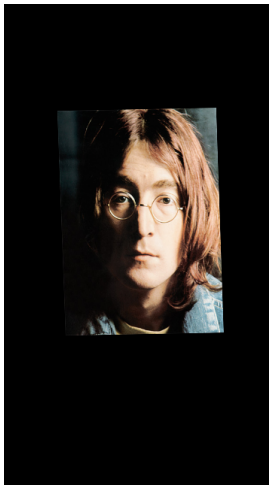


Fig. 10: Mask.

IX. Conclusions

We performed successfully augmented reality using an target image over each frame of a video.

RANSAC with RSME for 2D transformations proved to be a practical to obtain the matrix transformation of rotation and translation of the frame that we use as canvas,

When we project an image using an affine transformation over a canvas, the sense of virtual reality look great most of the time.

Fig. 11: Processed frame with pasted overlying image.

However, we saw that in some frames a distorted projection of the imagine occurs. This is due to the false positives.

Something really important that using RANSAC we can use likely any target frame as canvas, without the need of mark points or fiducial markers as many applications use this to project a GIF in order to simulate augmented reality.

References

[1] R. T. Azuma, "A Survey of Augmented Reality," *In Presence: Teleoperators and Virtual Environments 6*, vol 4, pp. 355-385, 1997

[2] E. Rublee,V. Rabaud, K. Konolige, and Gary Bradski,"ORB: An efficient alternative to SIFT or SURF," in IEEE International Conference on Computer Vision, Barcelona (ICCV), pp. 2564–2571 , 2011

[3] S. Tareen, and Z. Saleem, "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK," in Computing, International Conference on Mathematics and Engineering Technologies (iCoMET), pp. 1-10, 2018.

Contribution Matrix

Item\RA	**230293**	**263068**	**230254**
Key-Points /Descriptor	33,33%	33,33%	33,33%
Affine Transformation Solver	33,33%	33,33%	33,33%
RANSAC	33,33%	33,33%	33,33%
Image Transformation	33,33%	33,33%	33,33%
Video Generation	33,33%	33,33%	33,33%
Report	33,33%	33,33%	33,33%

APPENDIX A
SIFT

A. Feature Point Detector with SIFT

In order to get good key-points we first we have get the feature with the highest gradients of the image for that we the Differential of Gaussian of an image, so the more relevant features will remain

SIFT is very good approximation of this Laplacian detector called the Difference of Gaussians (DoG) detector.

1) DoG:

$$DoG_\sigma(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right). \tag{7}$$

```
sigma=1.7
def DIFFgaussian_kernel(size, sigma):
    size = int(size//2)
    x, y = np.mgrid[-size:size+1, -size:size+1]
    g = np.exp(-(x*2/float(size)+y2/float(size_y)))/
    np.sqrt(2*np.pi*sigma*2)
    return g / g.sum()

DG = DIFFgaussian_kernel(13,1)
Kernel=np.around(255*DG/np.amax(DG), decimals=0)
```

Algorithm 1: Function to find the Differential of Gaussian Kernel

2) finding the most relevant descriptors: alex para corrige los errores de toda tu seccion sino no va a correr .

$L(x,y,k\sigma)$ is the convolution of the original image $I(x,y)$ with the Gaussian blur $G(x,y,k\sigma)$

$L(x,y,k\sigma) = G(x,y,k\sigma) * I(x,y)$

the DoG image $D(x,y,\sigma)$ is obtain by

$D(x,y,\sigma) = L(x,y,k_i\sigma) - L(x,y,k_j\sigma)$

$L(x,y,k_i\sigma)$

$L(x,y,k_j\sigma)$

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix} \tag{8}$$

The trace of H, i.e., $D_{xx} + D_{yy}$, gives us the sum of the two eigenvalues, while its determinant, i.e., $D_{xx}D_{yy} - D_{xy}^2$, yields the product. The ratio $R = \mathrm{Tr}(\mathbf{H})^2/\mathrm{Det}(\mathbf{H})$ can be shown to be equal to $(r+1)^2/r$

from the Gaussian-smoothed image $L(x,y,\sigma)$ we have to get the magnitude and the orientation

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{9}$$

$$\theta(x,y) = \arctan2\left(L(x+1,y) - L(x-1,y), L(x,y+1) - L(x,y-1)\right) \tag{10}$$

d_q is the query descriptor d_1 is the first nearest descriptor in the search image d_2 is the second nearest descriptor in the search image

reject unless $\frac{\arccos(d_q * d_1)}{\arccos(d_q * d_2)} < r$ r where r is the threshold ratio.

B. (

Estimating Fundamental matrix F by RANSAC)
we Need 7 (8) correspondences X1, X2
Sample set = set of matches between 2 images

1) Select a random sample of minimum required size 8 inerlines (X1, X2)
2) Compute a putative model from these
3) Compute the set of inliers to this model from whole sample space Repeat 1-3 until model with the most inliers over all samples is found

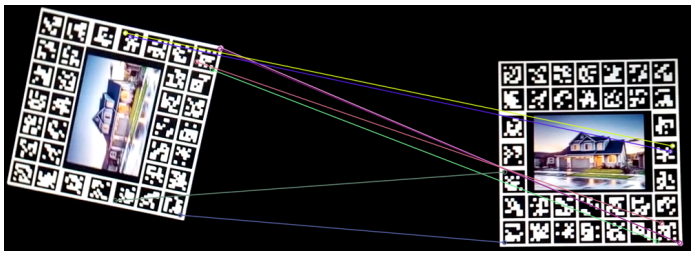


Fig. 12: 8 points to construct inerlines

```
def GetFundamentalMatrix(x1,x2):
  A=[]
  for i in range(len(x1[:,0])):
    A+=[[x1[i,0]*x2[i,0],x1[i,0]*x2[i,1],x1[i,0],x1[
    i,1]*x2[i,0],x1[i,1]*x2[i,1],x1[i,1],x2[i,0],x2[
    i,1],1]]
  A=np.array(A)
  U, S, VT = svd(A)
  # U, S, VT = np.linalg.svd(A, full_matrices=False,
     compute_uv=True)
  f=VT[:,8]
  u,s,vt=svd(F)
  d[2]=0
  FM=np.matmul(np.matmul(u,np.diag(d)),vt.T)
  return FM

def Ransac8points(Soursepoints,DestinationPoints):
  #lista=list(range(len(vx)))
  MaxInliner=0
  S=[]
  for k in range(50):
    ListaPoints=list(range(NumPoints))
    Points8=random.sample(ListaPoints,k=8)
    X1,X2=Soursepoints[Points8,:], DestinationPoints
    [Points8,:]
    FM=GetFundamentalMatrix(X1,X2)
    u,v=Soursepoints[ListaPoints,:],
    DestinationPoints[ListaPoints,:]
    NumInliner=0
    for r in range(NumPoints):
      vr1=np.append(v[r,:],1)
      ur1=np.append(v[r,:], 1)
      #chekear
      err=abs(np.matmul(np.matmul(vr1,FM),ur1.T))/np
    .linalg.norm(np.matmul(FM[0:1,:],ur1.T))
      if err<1:
        NumInliner=NumInliner+1
    if MaxInliner<NumInliner:
      MaxInliner=NumInliner
  return FM
```

Algorithm 2: Function of RANSAC 8points