Nhóm 3:

Thành viên:

18521591 Đặng Nguyễn Anh Tuấn

18521503 Đặng Hữu Toàn

18521632 Nguyễn Văn

Đề tài: Augmented Reality with local feature

Phần 1: What is augment reality?

Augmented Reality (AR) is known as a technology allow human to observe, interact with virtual information (3D objects, video, image…) in real life through a electronic device

Ưu, Nhược, thách thức:…

Phần 2: các bước tiến hành

1. OverView

Using OPENCV, project in a screen a 3D model of a figure whose position and orientation matches the position and orientation of some predefined flat surface.

Furthermore, we want to do it in real time, so that if the surface changes its position or orientation the projected model does so accordingly.

To achieve this we first have to be able to identify the flat surface of reference in an image or video frame.

Once identified, we can easily determine the transformation from the reference surface image (2D) to the target image (2D). This transformation is called [homography](https://en.wikipedia.org/wiki/Homography_(computer_vision)).

Finally, project the 2d image or 3d model to the flat surface

The project can be divined into:

          1.  Identify the flat surface of reference in an image or video frame

          2.  Use homography method to transform space.

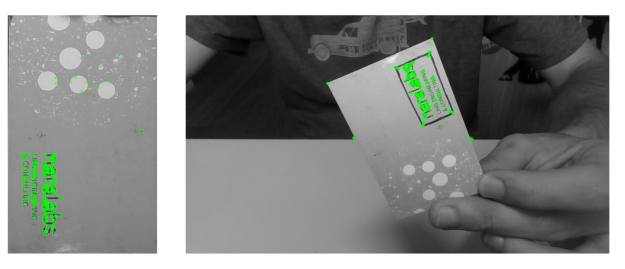
          3.  Finally, project the 2d image or 3d model to the flat surface.

1. **Material**

The material we wwill

1. Recognizing the target surface
   1. Feature extraction

Roughly speaking, this step consists in first looking in both the reference and target images for features that stand out and, in some way, describe part the object to be recognized. This features can be later used to find the reference object in the target image.



We will assume we have found the object when a certain number of positive feature matches are found between the target and reference images

For a region or point of an image to be labeled as feature it should fulfill two important properties:

first of all, it should present some uniqueness at least locally. Good examples of this could be corners or edges.

Secondly, since we don’t know beforehand which will be, for example, the orientation, scale or brightness conditions of this same object in the image where we want to recognize it a feature should, ideally, be invariant to transformations; i.e, invariant against scale, rotation or brightness changes

* 1. Feature description

Once features have been found we should find a suitable representation of the information they provide. This will allow us to look for them in other images and also to obtain a measure of how similar two detected features are when being compared.

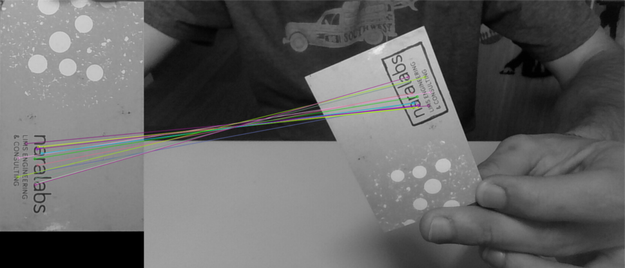
This is were descriptors roll in.  A descriptor provides a representation of the information given by a feature and its surroundings. Once the descriptors have been computed the object to be recognized can then be abstracted to a [feature vector](https://en.wikipedia.org/wiki/Feature_vector),  which is a vector that contains the descriptors of the keypoints found in the image with the reference object.

There are many algorithms that extract image features and compute its , take a look at [SIFT](http://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/), [SURF](http://www.vision.ee.ethz.ch/~surf/eccv06.pdf), or [Harris](http://aishack.in/tutorials/harris-corner-detector/). The one we will be using was developed at the OpenCV Lab and it is called [ORB](http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html) (Oriented FAST and Rotated BRIEF). The shape and values of the descriptor depend on the algorithm used and, in our case,  the descriptors obtained will be binary strings

* 1. Feature matching

 The simplest way of doing this is to take the descriptor of each feature in the first set, compute the distance to all the descriptors in the second set and return the closest one as the best match (I should state here that it is important to choose a way of measuring distances suitable with the descriptors being used. Since our descriptors will be binary strings we will use [Hamming distance](https://en.wikipedia.org/wiki/Hamming_distance)). This is a brute force approach, and more sophisticated methods exist.

For example, and this is what we will be also using, we could check that the match found as explained before is also the best match when computing matches the other way around, from features in the second set to features in the first set. This means that both features match each other. Once the matching has finished in both directions we will take as valid matches only the ones that fulfilled the previous condition. Figure 4 presents the best 15 matches found using this method.

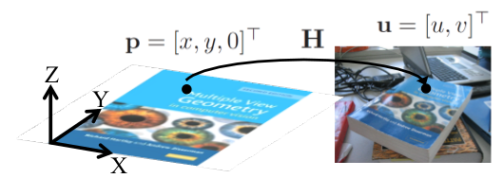


Finally, after matches have been found, we should define some criteria to decide if the object has been found or not. For this I defined a threshold on the minimum number of matches that should be found. If the number of matches is above the threshold, then we assume the object has been found. Otherwise we consider that there isn’t enough evidence to say that the recognition was successful.

On a final note and before stepping into the next step of the process I must point out that, since we want a real time application, it would have been better to implement a tracking technique and not just plain recognition. This is due to the fact that object recognition will be performed in each frame independently without taking into account previous frames that could add valuable information about the location of the reference object. Another thing to take into account is that, the easier to found the reference surface the more robust detection will be.

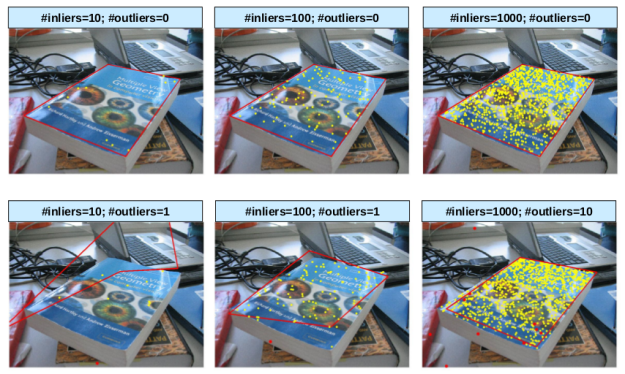
1. Homography estimation

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. As explained before, we want to find the transformation that maps points from the surface plane to the image plane (see Figure 5). This transformation will have to be updated each new frame we process.



Homography is …

There are several methods that allow us to estimate the values of the homography matrix, and you maight be familiar with some of them. The one we will be using is **RAN**dom **SA**mple **C**onsensus ([RANSAC](https://en.wikipedia.org/wiki/Random_sample_consensus)).  RANSAC is an iterative algorithm used for model fitting in the presence of a large number of outliers, and Figure 12 ilustrates the main outline of the process. Since we cannot guarantee that all the matches we have found are actually valid matches we have to consider that there might be some false matches (which will be our outliers) and, hence, we have to use an estimation method that is robust against outliers.



RANSAC::

Result

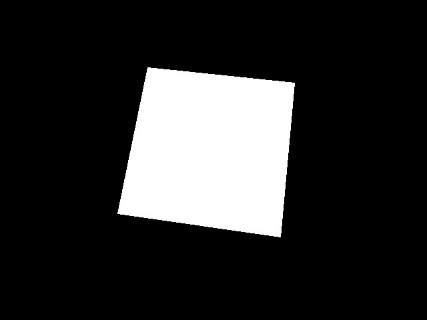
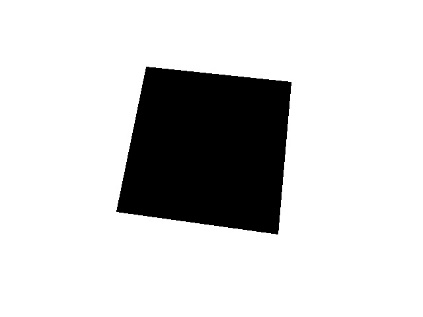
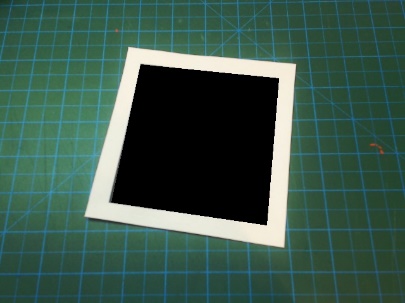


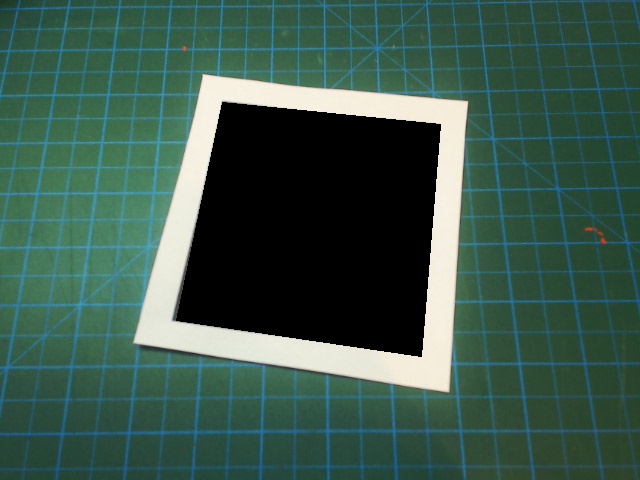
1. 2d./

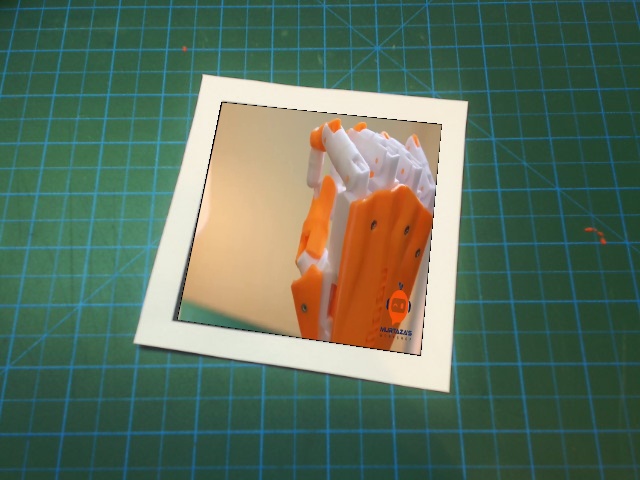
### **Creating the Mask**

 remove the area in the webcam image where we want to overlay our target image and then add them together. This can be done using masking.

So first we are creating a mask based on the location of the target found. Now we can use the inverse method to find its negative. If we add the mask inverse and the webcam image we would get the anew image where all the webcam image information is shown except where the image is suppose to be augmented. So the black area can be thought of an empty space where we can add our image.



So once we have the imgAug which is our new masked image and the imgWarp, we can simply add them up using the bitwise Funtion. (add final image with warped image)



1. 3d
   1. Pose estimation from a plane

…

* 1. Model projection

 the final stages of the project

only using simple models in Wavefront .obj format. Why OBJ format? Because I found them easy to process and render directly with bare Python without having to make use of other libraries such as OpenGL. The problem with complex models is that the amount of processing they require is way more than what my computer can handle. Since I want my application to be real-time, this limits the complexity of the models I am able to render.

several (low poly) 3D models format from [clara.io](https://clara.io/)

loads the geometry of the model. Once the model is loaded we just have to implement a function that reads this data and projects it on top of the video frame with the projection matrix we obtained in the previous section. To do so we take every point used to define the model and multiply it by the projection matrix. One this has been done, we only have to fill with color the faces of the model. .

final demo