Assembling LEGO set with augmented reality instructions

Othman Sbai
othman.sbai@eleves.enpc.fr

Pierre-Alain Langlois
pierre-alain.langlois@eleves.enpc.fr

Abstract

For our project for Object recognition and computer vision course, we tackled the problem of assembling Lego set using computer vision techniques. The aim is to make use of new holographic tools in order to suggest the right instructions to the user who is performing assembly task. We will first present our approach of the problem, show how we used computer vision algorithms to track the evolution of the assembly phase and suggesting the right instructions. For that task we will present the new and fast object detection framework called YOLO (You Only Live Once), which we used as a base image recognition tool by adapting it the LEGO pieces. We will present our training results accompanied by what difficulties we faced when using this young framework, and finally we will expose the work we did on the interface part of the project which means, building a hololens app that annotates the holographic world of the user by adding 3D LEGO objets on the detected pieces to move suggesting, this way, the next step to perform.

1. Introduction

The advent of augmented reality devices such as Microsoft Hololens, Sony SmartEyeglass or Google Glass and others have made possible many interesting applications that augment the visual experience of the user with 3D holograms that are blended on his reality. Applications range from interior decoration and design, gaming but also increasing productivity in businesses by enhancing the real world and giving birth to broader imagination.

Among the applications of this new promising technology, still in development, one can be interested providing instructions for people to help them accomplish tasks either with human supervision or by annotating reality through an artificial agent that sees the world and suggests instructions through holograms as it is shown in figure 1. In fact, as presented in [1], we can extract from the tutorial videos available online instructions for performing many tasks such as changing car tires, assembling furniture and also performing Cardiopulmonary rescucitation. These instructions can be efficiently provided to user with a 3D augmented reality device in the form of holograms and world annotations that are much comprehensible than paper instructions.

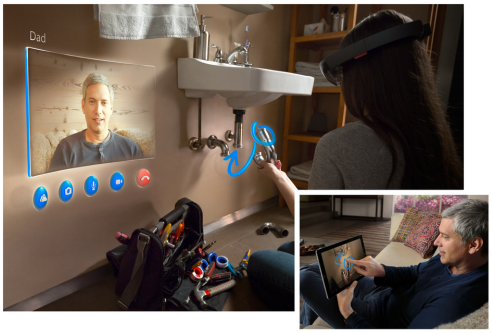

Figure 1. Annotating the world with Microsoft Hololens.

2. Project organization

We organized the work on this project in two consecutice sections, first detecting and localizing LEGO assemblies from a 2D video feed, for example the 2D locatable camera in the front of the Hololens and secondly present the relevant instructions for the corresponding step on the Hololens by blending LEGO Holograms on the user's reality. Note that it is possible to save the augmented reality video of both the camera feed and the annotating holograms from the Hololens thanks to a functionality called mixed capture.

3. Learning to recognize the assemblies and their location on an image

3.1. Suggested approach

In order to suggest the right instruction our algorithm should be able to detect the state or advancement of building the set and detect the assemblies. The problem can get complex if we think about the multitude of possibilities the user can do to fool the algorithm, but we focus on the use case where the user starts building one assembly gradually and not multiple assemblies. For each step, there are only 3 instances to recognize: the current assembly state, the part

that has to be added, and the global assembly when the instruction has been fulfilled. As a consequence, we want to build a system which is able at each state to recognize the 3 current instances and to locate them in the image.

In order to do so, we use the method described in (YOLO) [5], which we will presented in subsection 3.3 which allows to perform local classification efficiently thanks to a convolution neural network. This approach requires to first generate relevant data in order to perform the training.

3.2. Generating the data

Since we are not able to directly generate a dataset of annotated pictures for the lego set, we generate it artificially. We first create the 3D model the lego parts. In order to do so, we did use the software SolidWorks. Then, we built a script that generates the annotated pictures with Blender [2] and OpenCV [3]. Another possibility would be to use a software fro LEGO company called LEGO Digital Designer which provides all the LEGO parts 3D models and facilitates the assembly virtually.

In this section, we detail this script created for generating random rendering

3.2.1 Build

On the machine, we need to build Python 3 from source with './configure –enable-shared'. Then we need to compile Blender as a module for this Python distribution as well as OpenCV. Then, we have all the tools needed to perform the generation.

3.2.2 Input

The important point here is that the classes of our classification task are the assemblies, not the parts themselves. As a consequence, we have created a standard csv file that enables to defines an assembly by specifying each file that constitutes the assembly, the diffuse color for each part, and the world matrix that enables to place each part in space. Note that it is easy to place an assembly in blender gui and to obtain the World Matrix for each object thanks to the python command 'bpy.context.active_object.matrix_world'. This framework allows to easily generate classes for other system than the lego games.

3.2.3 Algorithm

The algorithm is designed in two parts :

- Rendering the parts
- Making insertions

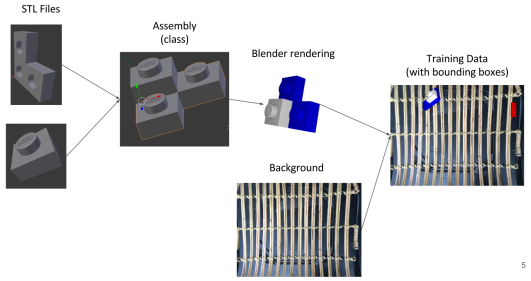

Figure 2. Rendering process

The full process is detailed on Figure 2. Each assembly is being set up thanks to the world matrix and the corresponding materials are setup thanks to the defined color. The number of rendering for each part is taken as a parameter. We define the center of the object as the center of its bounding box. For each rendering, we take a random point on the sphere around the object. We set a camera that points towards the object center and we include a random lightning that is located in a region which is close to the camera. Then, we are able to make the rendering.
The second is step is to randomly insert the rendering on random backgrounds. Note that we add a random number of renderings on a single background. We obtained the backgrounds from the Imagenet [4] dataset. We set the size of the rendering to be between 10% and 90% of the background size with a threshold in order to avoid oversizing.
In YOLO, we must set one text file per image in order to define the bounding boxes and the class of each object located in the window. We checked the validity of our implementation by drawing the bounding boxes with the same function on the data that were used in [5] and our data.

3.3. About Yolo framework

The Yolo framework was entirely built by the author of [5]. It is made in C language with a CUDA support in order to be able to perform efficient training. The framework is still under development and needs a few modifications in order to be able to perform detection on arbitrary data. The YOLO neural divides the picture on a regular grid and it generates two types of data out of an image. It first generates one class detection per cell on the picutre ; and each cell is also responsible for generating a given number of bounding boxes that are centered inside of the cell itself, with a confidence indice for each box. From the two information and thanks to an additional threshold on the confidence value, we are able to deduce the bounding boxes.
The cost function for the training (see [5]) allows to give a customized weight on each aspect of the prediction : the center of the bounding box, its size, and the class that is being predicted. When we train the network on customized data, we only have to train the 4 final layers, because the first layers are trained on imagenet. This makes the data

Figure 3. Typical learning curve when training the model

customization easier because we need less time to compute the customized network.

3.4. Results

We tried the training on 2 classes. We chose 700 rendering per class and 10000 training data (renderings added to the backgrounds) for the training part. 0.1% of the data is dedicated to the evaluation. In order to monitor precisely the learning process, we modified the code in order to output the learning curves as represented on Figure 3. The training took about 2 hours on an AWS EC2 web server. Unfortunately, we could not achieve the last integration part in which we use the trained network on the Hololens. However, we could plan the integration process and we could experiment on this last integration brick.

4. Porting to Hololens

Microsoft Hololens is not a cheap platform to work on as it cost at least 3000$. We would like to thank our professor for providing this device for development. Ans thankfully there is the possibility of working on an emulator for hololens on Visual Studio. This requires a certain setup: (Visual Studio 2015 working on Windows 10 Pro or Education edition which support virtualization needed by the emulator). There are two possible ways of creating an app for the Hololens, either by creating a scene on the game engine Unity, with a specified setup and then building the VS solution, or directly creating a DirectX C++ app which is more low-level than the first one.

In order to draw holograms in the correct place in the 3D scene in Unity and thus rendered in the right 3D location in user's vision, we need to map the pixel position on the RGB 2D frame obtained from the hololens locatable camera to the 3D coordinate in the Holographic Scene. This mapping provides only the ray toward which the object should

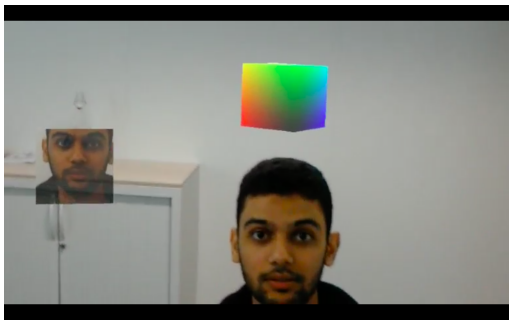

Figure 4. Annotation based on specified location using frontal camera info.

be place, however we still need to determine the distance of the holorgram from the user. This can be achieved through two different methods. One can for example place the holograms relative to the surrounding space of the user by intersecting the ray found previously with the spatial mapping information provided by the Hololens. Or one can use the size of the detected bounding box and the known size of the tracked object to infer the depth of the hologram.

We demonstrated the second approach based on a code from Microsoft Universal Samples where we detect a face an superpose a rotating colored cube using a DirectX app as we can see in the figure 4

References

[1] J.-B. Alayrac, P. Bojanowski, N. Agrawal, J. Sivic, I. Laptev, and S. Lacoste-Julien. Unsupervised Learning from Narrated Instruction Videos. *arXiv:1506.09215 [cs]*, June 2015. arXiv: 1506.09215.

[2] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2016.

[3] G. Bradski. Opencv. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *arXiv:1506.02640 [cs]*, June 2015. arXiv: 1506.02640.