3/24/2020 - Eduardo Davalos

**THESIS WORK REPORT**

**1. Background Information**

The goal of this thesis (still seeing if this is possible) is to create an RGB category-level 6D pose estimation neural network (NN). The field of computer vision is neatly cut into different sections by task, such as object detection, image segmentation, face recognition, and pose estimation. 6D pose estimation is the estimation of two main attributes of an object: translation and rotation. Note that 6D pose estimation is different from human body estimation, for 6D pose estimation only addresses rigid bodies. Each attribute requires incorporate distinct 3D information, therefore resulting in 3D + 3D = 6D, hence the name. In 6D pose estimation, rotation and translation are determined with respect to the camera’s position and angle.

A picture containing table

Description automatically generated

Figure 1: Rotation and Translation Visualized

*1.1 Category-Level vs. Instance-Level*

One of the important distinctions in 6D pose estimation is Category-Level and Instance-Level pose estimations. Instance-level pose estimation focus on a single instance of an object’s category. Category-level pose estimation focus on an entire object’s category, not just one instance of it. For example, an instance-level pose estimation NN would only be able to estimate the pose of the single camera on the left side of Figure 2, while a category-level pose estimation NN would be able to estimate the pose of all the cameras on the right side of Figure 2.

A close up of a camera

Description automatically generated VS. A close up of a camera

Description automatically generated

Instance Category

Figure 2: Instance-Level vs. Category-Level

Currently for testing purposes, I am using [NOCS\_CVPR2019](https://github.com/hughw19/NOCS_CVPR2019), an RGBD Category-Level Pose Estimation NN, to achieve the task of Category-Level classification. One of the main ways that NOCSnet achieves Category-Level classification is by its use of NOCS, Normalized Object Coordinate Space, to represent an object’s category with a single 3D model. This 3D model is constructed by averaging multiple instances of an object to achieve a form, like Plato’s theory of form, that represent the object’s general properties and attributes. Figure 3 shows an example of a category’s 3D model, in this case a camera.

A picture containing light

Description automatically generated

Figure 3: Category-Level 3D of ‘Camera’ Category

The main difference between this thesis’s NN and NOCSnet is that NOCSnet requires depth as an input. My goal is a way to remove this requirement. Since NOCSnet’s representation of objects within the NOC Space is normalized, obtaining the true scale of the object is difficult. This is why depth is used by NOCSnet. Therefore, this thesis’s NN intents to incorporate depth prediction to into the network’s design to remove the depth input requirement.

Additionally, instance-level applies multiple restrictions to the neural network. Instance-level requires that the objects detected have been scanned and

*1.2 Depth Prediction*

The neural network I am using to create depth images is DenseDepth. Its input is an RGB image of size (480, 640) and it outputs a depth image have the size (240, 320). Simply the image is scaled up to match the size of the input.

**2. Current Situation**

The main challenge that I am trying to do is format the depth generated by DenseDepth to use it for the NOCS network. By formatting, I am referring to changing the size, encoding, and scaling of the depth image to match the excepted original depth images.

Main Differences (that I know this far)

DenseDepth Depth

* Size: (240, 320)
* Encoding: float32
* Scaling: ?

NOCS RGBD Camera Depth

* Size: (480, 640)
* Encoding: uint16
* Scaling: ?

In the following page, I will show multiple images of various experiments to explain my investigation for how to make the depth generated by DenseDepth functional with NOCS network.

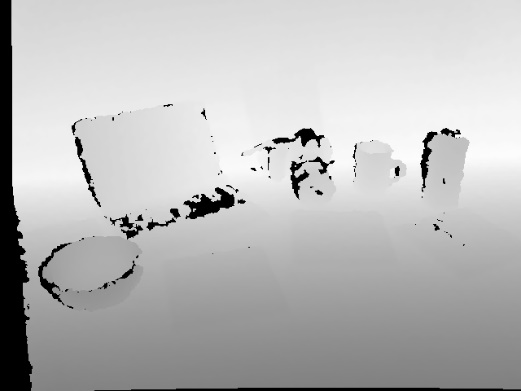


Figure 1: NOCS RGB and D images

**3. Future Work & Conclusion**