Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

Interactive Data Augmenting Tool

Software Development Project

January 23, 2020

Venkata Santosh Muthireddy

Introduction

Problem

- Existing benchmark datasets are not suitable for custom objects.
- Semantic segmentation dataset is needed for RoboCup @Work objects.
- High labeling cost for semantic segmentation data.
- Time-consuming to annotate.

Motivation

- Semantic segmentation provides rich information from image space.
- Potential applications:
  - Semantic segmentation of living areas.
  - Object recognition tasks.
  - Ease of dataset generation for custom objects.

Solution Approach

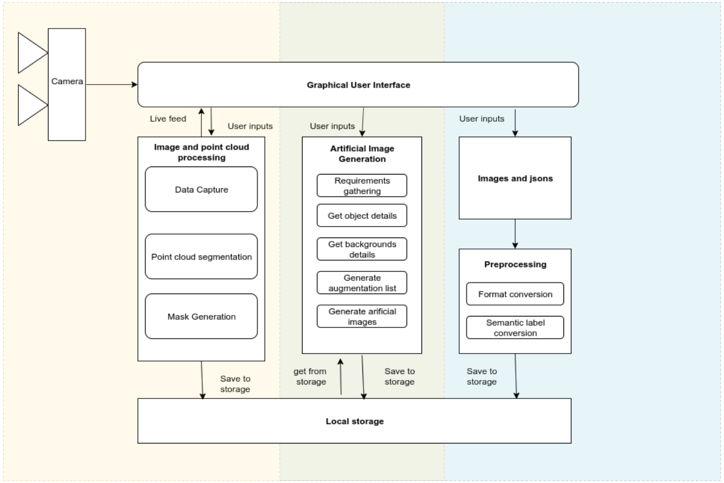
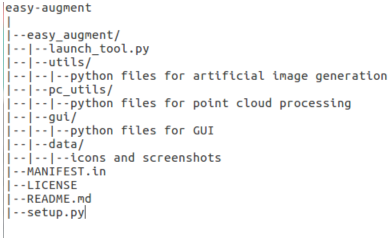


Figure 1: Architecture of the developed project, left: handles capturing part and right: handles labeleme annotations

Solution Approach

Design decisions

- Software development approach: Agile
- Coding convention: PEP 8
- Programming paradigm: Object Oriented Programming
- Design pattern: The Composite Pattern
- Input: Raw data from camera or Annotated images.
- Output: Images and annotations.



```
easy-augment
|
|--easy_augment/
|--|--launch_tool.py
|--|--utils/
|--|--|--python files for artificial image generation
|--|--pc_utils/
|--|--|--python files for point cloud processing
|--|--gui/
|--|--|--python files for GUI
|--|--data/
|--|--|--icons and screenshots
|--MANIFEST.in
|--LICENSE
|--README.md
|--setup.py
```

Figure 2: Folder structure of package that follows Composite pattern

Implementation

System setup:

- Ubuntu 16.04 (18.04 testing)
- Intel RealSense Camera
- Processor Intel i5 or higher
- Python 3.5
- Installed pcl 1.7 library

Implementation

Dependencies:

- GUI: PyQt5
- Data Processing: OpenCV, pcl-, Scipy, Numpy, pascal-voc-tools, pascal-voc-writer
- Hardware: PyRealSense2
- Testing: pytest-qt
- Others: tqdm, matplotlib, imutils, joblib

Implementation

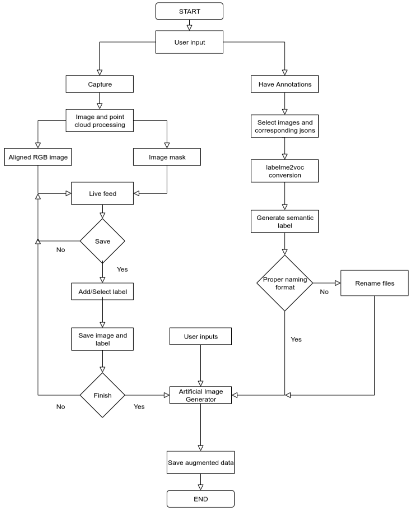


Figure 3: Overall algorithm for the project

Implementation

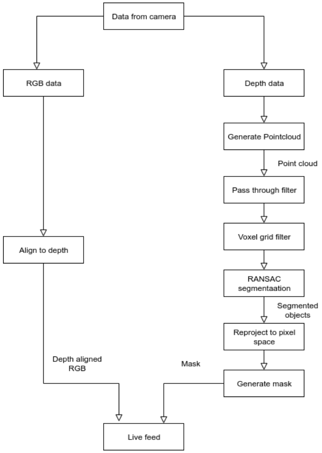


Figure 4: Image and point cloud processing algorithm for the project

Implementation

Refactoring

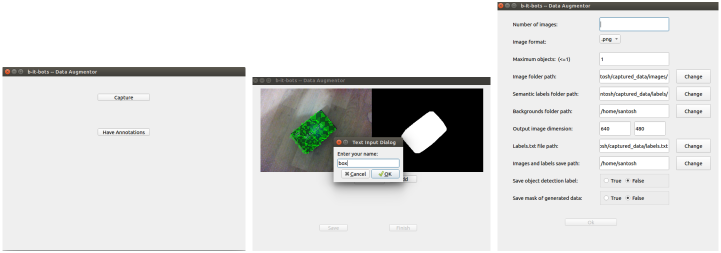
- Source code from artificial image generator.[1]
- Debugging:
  - Image size (< 640X480 pixels)
  - Input arguments refactored.
  - Position of augmented data.
  - Python cyclic dependency.
- Additional functionalities:
  - Input and output dimensions of images can be controlled.
  - Any naming format can be given for images and labels (e.g, classname_00x.png/jpg).

[1] "Naresh Kumar Gurulingan, Semantic Segmentation using Resource Efficient Deep Learning, Research and developement project" submitted to HBRS.

Implementation

GUI development

- Choice of technology: PyQt5, Python3.5
- Graphical User Interface for ease of using the tool.



b-it-bots -- Data Augmentor
Capture
Have Annotations
b-it-bots -- Data Augmentor
Text Input Dialog
Cancel
OK
Save
Finish
b-it-bots -- Data Augmentor
Change
Ok

Enter your name: box

Number of images:	
Image format:	.png
Maximum objects: (<=1)	1
Image folder path:	tosh/captured_data/images/
Semantic labels folder path:	ntosh/captured_data/labels/
Backgrounds folder path:	/home/santosh
Output image dimension:	640 480
Labels.txt file path:	osh/captured_data/labels.txt
Images and labels save path:	/home/santosh
Save object detection label:	True [✓ False]
Save mask of generated data:	True [✓ False]

Figure 5: Sample GUI windows from left to right: Main window, Capture window and Artificial image generator window

Implementation

Point cloud segmentation

- Choice of technology: pcl 1.7, python-pcl, Python3.5, OpenCV
- Generation of pointcloud from depth data.
- Object segmentation from pointcloud.
- Point space to pixel space projection.
- Mask generation for objects.

Implementation

Point cloud segmentation

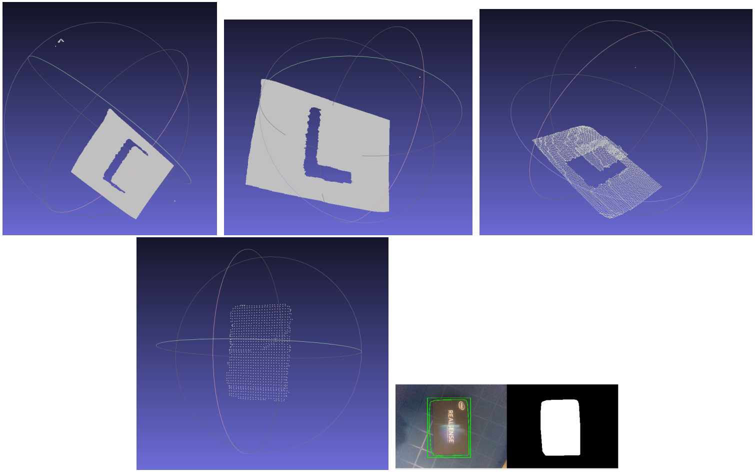


Figure 6: Pointcloud segmentation various stages for object retrieval.

Implementation

Augmenting with labelme annotations

- Using labelme annotations to augment data.
- convert to Pascal_VOC format to generate semantic masks.

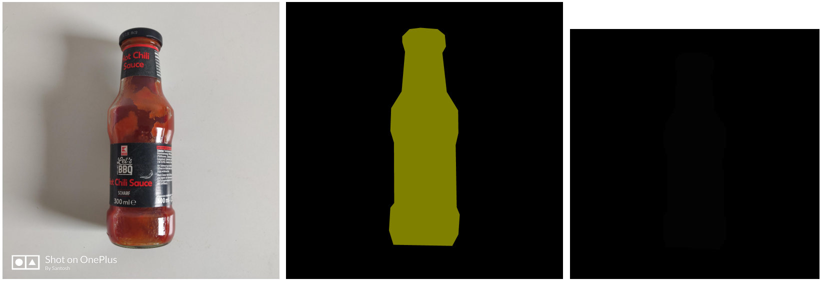


Figure 7: RGB Image, Generated PASCAL VOC mask, semantic mask with class label

Implementation

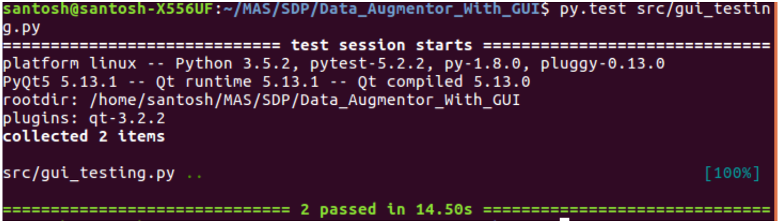
Problems occurred and solved

- Name format - classname_00x.png/jpg
- Image dimension
- Immutable arguments
- Travis ci - continuous integration
- PCL version dependencies
- Integrating qt-bot for testing.
- Python cyclic dependency for adapted source code.

Implementation

Testing and Continuous Integration

- Choice of technology: pytest-qt(Testing), Travis-ci(Continous Integration)
- Test cases for GUI simlation are done using pytest-qt
- Test cases are used to check build status in travis-ci



```
santosh@santosh-X556UF:~/MAS/SDP/Data_Augmentor_With_GUI$ py.test src/gui_testin
g.py
============================ test session starts =============================
platform linux -- Python 3.5.2, pytest-5.2.2, py-1.8.0, pluggy-0.13.0
PyQt5 5.13.1 -- Qt runtime 5.13.1 -- Qt compiled 5.13.0
rootdir: /home/santosh/MAS/SDP/Data_Augmentor_With_GUI
plugins: qt-3.2.2
collected 2 items

src/gui_testing.py ..                                                   [100%]

============================== 2 passed in 14.50s ============================
```

Figure 8: Testing using pytest-qt

Development Status

Capabilities:

- Live feed of RGB images and corresponding mask.
- Save Image and corresponding semantic label.
- Generate artificial data from captured images.
- Generate artificial data from labelme annotated images.
- Control shape of artificial generated data.
- Types of labels generated:
  - Semantic label with class information.
  - Pascal VOC semantic label.
  - Bounding box in xml file.
- Package delivered as pip installable.

Development Status

Installation and Usage:

- pip3 install easy-augment
- “easy-augment“ command launches the tool.

Limitations:

- Only RealSense camera can be used.
- Number of classes captured >=2.
- Only one object per scene is allowed.

Future Work:

- Testing for Ubuntu 18.04
- Implementing generic python pcl wrapper to avoid dependencies.
- Enable for multiple RGBD cameras like Microsoft Kinect.

Lessons Learnt

- Development life cycle of a software project.
- Selecting testing framework.
- Writing test cases.
- Enabling continuous development and continuous testing.
- Using GUI developing libraries.
- Version control and git commands.
- Building and deploying pip package.

Contributors

- Venkata Santosh Sai Ramireddy Muthireddy
- Naresh Kumar Gurulingan
- M. Sc. Deebul Nair

Thank You