Synthetic Dataset Generation Toolbox and Enhanced 6D Pose Estimation and Object Detection on YCB Objects

Hsin-Hua Lu [*] Gabriel SantaCruz [†] Pin-Hua Huang [‡]

Abstract

In recent years, pose estimation has gain its popularity in enabling robots to efficiently manipulate objects and interact with their surroundings. Xiang et al. introduced PoseCNN, a VGG16 network, as a solution for estimating both 6D translational and rotational poses, alongside semantic labels. Building upon this foundation, our research generates a comprehensive synthetic dataset comprising 2G of meticulously crafted scenarios featuring 25 YCB object setups. This synthetic dataset encompasses a diverse range of camera angles, curated to simulate the real-world settings and complex occlusion scenarios, thereby facilitating robust object pose estimation. Leveraging the capabilities of the pretrained PoseCNN backbone, our objective is to further enhance the accuracy and efficacy of 6D pose estimation on our curated dataset. Through this work, we aim to contribute to the ongoing evolution of pose estimation methodologies for more sophisticated robotic applications in real-world environments.

1. Introduction

Estimating 6D pose, including rotation and position, of known objects has always been an important task in robot manipulation. Robots need to understand the spatial relationship among the objects in order to interact with the environment and execute further control strategies. Moreover, the ability to perform classification tasks on the identified objects expands the cognitive capabilities, enabling the execution of intellectual jobs such as organizing pick-and-place operations or object localization.

Xiang *et al.* [11] introduced PoseCNN, a Convolutional Neural Network (CNN) designed for 6D pose estimation. PoseCNN estimates the 3D translation from the object center and camera distance, along with 3D rotation by regressing quaternion representations. They trained on the 92 YCB-video dataset and achieved high scores in classification on OccludedLINEMOD.

Building on this foundation, we extend PoseCNN to utilize additional data sources such as video inputs, point clouds, and camera trajectory information for enhancing the 6D pose estimation of YCB objects. Moreover, we adjust the output of the model to be probabilistic, based on bivariate Gaussian distributions, and add an additional output of a 2-dimensional direction vector for the model to predict an optimal direction for camera movement to enhance object visibility. This latter approach is more focused on robotics applications, given the noise and complex environments robots might face. We believe it will significantly boost the performance of the model compared to arbitrary outputs of the current time frame, ultimately providing a suitable angle for robots to accurately perceive the environment and plan subsequent movements. However, to achieve this, a dataset with multiple angles (spherical around the object setup) is necessary to train the model.

To support this, we propose a synthetic dataset generation toolbox for YCB objects based on Gazebo and ROS. The data is fully labeled with objects' poses, mask information, RGB images, and depth images. This synthetic dataset may serve as a valuable resource for this PoseCNN improvement as well as future research in point tracking models.

NOTE: YCB Objects and Model set stands for the ”Yale-CMU-Berkeley Object and Model Set.” It's a collection of common household objects used for benchmarking object recognition and robotic manipulation tasks [4][2][3].

Proposed Enhancements on PoseCNN:

- **Camera Pose Input:** Integrate the current camera rotation into PoseCNN to better account for the object's relative orientation, mimicking human visual perception for improved 6D pose estimation.
- **Depth Data Integration:** Use depth data as an additional input to enhance the robustness and accuracy of the pose estimation.
- **Direction Vector Output:** Output a direction vector for camera movement to improve object visibility and pose estimation accuracy, particularly in occluded scenarios.

[*]Email:alex49@stanford.edu
[†]Email: gsantac@stanford.edu
[‡]Email:pin9465@stanford.edu

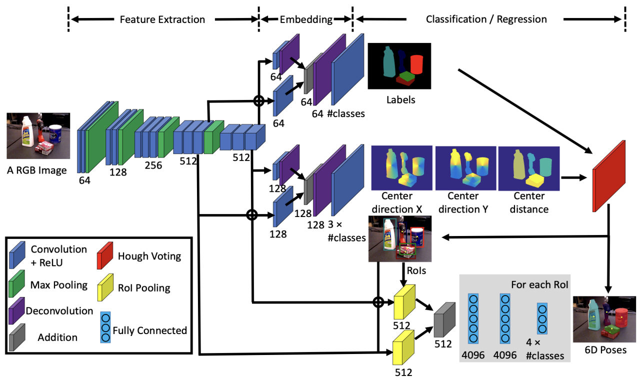


Figure 1: PoseCNN architecture from paper[11]

- **Probabilistic Pose Estimation:** Transition to a probabilistic model using Gaussian distributions to express uncertainty, allowing for confidence evaluation regarding camera angles and object visibility.

These enhancements aim to integrate into a Kalman filter-based robotic system, assessing the system's performance in dynamically adjusting the camera position based on the direction vector outputs, across consecutive frames. This advanced model will be further discussed and detailed in the Technical Approach section of this paper.

2. Related Work

2.1. PoseCNN Review

Our model use PoseCNN-PyTorch at its base - which is a Convolutional Neural Network developed by Nvidia[8]. It is designed for 6D object pose estimation using a direct regression method described in the paper [11]. PoseCNN predicts the 6D pose of objects with semantic segmentation and translational/rotational information [11]. PoseCNN utilizes a VGG16 backbone, trained on coco datasets and collected YCB video data, followed by three branches: semantic labeling (1), object translation (2), and object rotation (3). Specifically, for each pixel $p = (x, y)^T$ and each object in image it give following output:

$$(x, y) \rightarrow \text{cls} \tag{1}$$

$$(x, y) \rightarrow \left(n_x = \frac{c_x - x}{\|c - p\|}, n_y = \frac{c_y - y}{\|c - p\|}, T_z \right) \tag{2}$$

$$(object) \rightarrow \text{Quaternion}(q_w, q_x, q_y, q_z) \tag{3}$$

The VGG16 outputs CNN features that classify YCB objects for each pixel. The first branch then performs semantic labeling where deconvolutional layers are used to increase the spatial resolution of the feature maps. This branch is trained using softmax cross-entropy loss, with inference based on softmax probabilities.

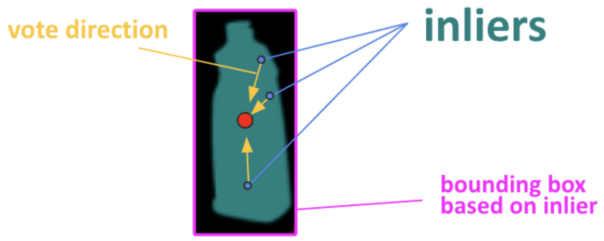


Figure 2: Hough voting for object center localization: Each pixel casts votes for image locations along the ray predicted from the network[11]

Next, PoseCNN leverages camera intrinsics to deduce translation rather than directly regressing it for object translation. Direct regression is not generalizable, as objects can appear in various locations within the image, making it challenging to learn translation solely based on object size. Additionally, direct regression cannot handle multiple instances of the same object category. Therefore, PoseCNN predicts the object center (c) and depth (T). Using the camera's focal length and principal point, object translation is calculated (4), where (p_x, p_y) is the principle point and (f_x, f_y) is the focal length of the camera.

$$\mathbf{T} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} \frac{(c_x - p_x)T_z}{f_x} \\ \frac{(c_y - p_y)T_z}{f_y} \\ T_z \end{bmatrix} \tag{4}$$

The object center is found using Hough voting influenced by 1 Implicit Shape Model (ISM) [7], where each pixel regresses a direction towards the object center. Non-maximum suppression is then applied to the voting scores followed by a threshold.

Lastly, 3D rotation regression uses Region of Interest (RoI) Max Pooling layers with fully connected (FC) layers to generate 3D rotation quaternions. The traditional pose loss (Equation 5) measures the average squared distance between points on the correct model pose and their corresponding points on the estimated model pose. This approach penalizes alternative rotations for symmetric objects unnecessarily. PoseCNN introduces ShapeMatchLoss (Equation 6) to handle symmetric objects more effectively by measuring the offset between each point on the estimated model and the closest point on the ground truth model. This complements the traditional pose loss by encouraging equivalent rotations for symmetric shapes.

$$\text{PLOSS}(\tilde{q}, q) = \frac{1}{2m} \sum_{x \in M} \|R(\tilde{q})x - R(q)x\|^2 \tag{5}$$

$$\text{SLOSS}(\widetilde{q}, q) = \frac{1}{2m} \sum_{x_1 \in M} \min_{x_2 \in M} \|R(\widetilde{q})x_1 - R(q)x_2\|^2 \quad (6)$$

PoseCNN uses the Average Distance (ADD) metric to evaluate the accuracy of 6D pose estimation. This metric calculates the mean pairwise distance between 3D model points transformed by the ground truth pose and the estimated pose. Specifically, the ADD metric is defined as:

$$ADD = \frac{1}{m} \sum_{x \in M} \left| (\mathbf{R}x + \mathbf{T}) - (\widetilde{\mathbf{R}}x + \widetilde{\mathbf{T}}) \right|, \quad (7)$$

where M denotes the set of 3D model points, and m is the number of points. For symmetric objects, where point matching can be ambiguous, PoseCNN employs the ADD-S metric, which uses the closest point distance:

$$ADD - S = \frac{1}{m} \sum_{x_1 \in M} \min_{x_2 \in M} \left| (\mathbf{R}x_1 + \mathbf{T}) - (\widetilde{\mathbf{R}}x_2 + \widetilde{\mathbf{T}}) \right|. \quad (8)$$

Given the robustness of these metrics in evaluating pose accuracy, we adopt both ADD and ADD-S as our evaluation metrics to ensure consistent and reliable performance assessment of our system.

Overall, PoseCNN features a novel CNN structure with three outputs, a robust physics-encoded model for translation estimation, and ShapeMatch-Loss for handling symmetric objects. PoseCNN demonstrates strong performance in cluttered scenes using only RGB images. However, PoseCNN has limitations in effectively estimating symmetric object poses and often gets trapped in local minima with ShapeMatchLoss. Higher resolution and FoV cameras may help to improve performance, however, the model struggles with heavily occluded objects which requires a direction vector output for complex, noisy environments.

3. Data

3.1. Dataset Preperation

In order to enhance the accuracy and robustness of our model, few types of datasets are selected. Including

- Training dataset: Simulated YCB object point clouds in Gazebo environment with different arrangements of objects and camera angles
- Training and validation dataset: YCB video datasets provided in Xiang's paper [11]
- Test dataset: OccludedLINEMOD [1]

3.1.1 YCB video dataset

The YCB video dataset comprises 113,198 images captured across 91 distinct environmental setups. Each frame showcases various YCB objects, such as scissors, screwdrivers, and more, positioned against diverse background images, often with certain parts occluded. In every environmental configuration, approximately 760 images were captured from similar angles and distances. Furthermore, the validation sets consist of around 2000 images per environmental settings from setting 48 to 59 obtained from the video frame. This compilation of video datasets enriches the expansive repository, offering potential advantages for future training in robotic computer vision. Nevertheless, a drawback of the video frame dataset is the absence of consistent angles and camera parameters, which could be accessed through the system environment settings to provide additional metadata.

3.1.2 COCO dataset

CoCo dataset stands for Common Objects in Context. It is a widely used benchmark dataset for object detection, segmentation and captioning tasks. The categories in the dataset consists of everyday scenes, including people, vehicles, and YCB objects. The 2014CoCo training dataset consists of approximately 80,000 images with corresponding bonding boxes, segmentations, keypoints and captions. In the PoseCNN-Pytorch structure, we utilized 2014 CoCo train image (13 GB) for background as the backdrop against which objects or subjects are placed or detected in computer vision tasks.

3.1.3 OccludedLINEMOD

OccludedLINEMOD is an extension of the LINEMOD (LINEns of Model) dataset, specifically designed for evaluating the performance of object detection and pose estimation algorithms in the presence of occlusions. LINEMOD is a widely used benchmark dataset in the field of object detection and pose estimation, containing images of objects from various categories captured under controlled conditions.

3.1.4 YCB objects model

The 3D object models used in the synthetic dataset generation toolbox is mainly from two sources including [6] and [5] covering over 40 YCB objects. Config, sdf, textures, materials and meshes of the object are prepared (Some generated by us) for later usage in the toolbox.

4. Synthetic Dataset Generation Toolbox

4.1. System Overview

Built upon [9] which runs on old API and ros envionment, our system leverages latest ROS 2 Humble and Gazebo Fortress to generate the synthetic dataset. The system configures object setups and samples camera poses using calculated angles and distances (`phi`, `theta`, `dist`). This approach ensures varied and full coverage of the environment, which is essential for training the direction vectors of the improved PoseCNN.

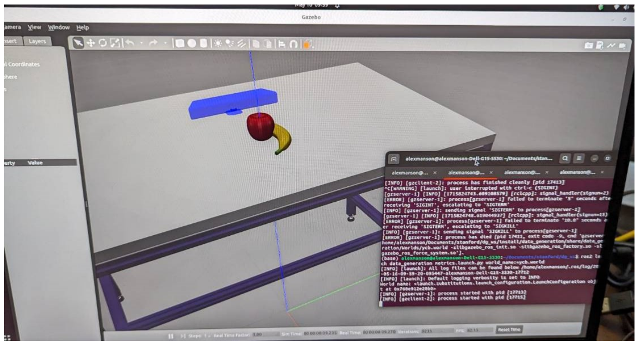
Camera and object states are managed through Gazebo's `GetEntityState` and `SetEntityState` services (from plugin libgazebo_ros_state.so), with `CvBridge` facilitating image data conversion and handling between ROS 2 and OpenCV. The workflow includes capturing RGB, depth, and binary mask images, which are then transformed into formats suitable for algorithm training.

The final outputs include RGB images, depth images, binary masks, and metadata capturing object poses and camera poses, all stored in PNG and MAT formats.

This system is particularly valuable for research as it effectively utilizes the latest ROS 2 API and accommodates the changes in the ros_gazebo_interface, which has been challenging to find in recent implementations. The dataset generation also considers relative spatial information, making it ideal for training models that require spatial context, such as finding direction vectors for optimal camera angles or reinforcement learning models that need to predict the scene after robot movement. The toolbox covers the entire environment setup and provides 100 different configurations with varying occlusion scenarios, along with a simple API to retrieve images and data for different camera movements (e.g., moving left a step).

4.2. System Setup

- **ROS 2 Humble:** Offers real-time communication capabilities within robotic systems and Gazebo integration.
- **Gazebo Fortress:** A robust robotics simulator that offers detailed environmental dynamics and physics.
- **Open3D:** Employed for its efficient 3D data processing capabilities, which is crucial for point cloud manipulation and analysis.
- **OpenCV:** Used for image acquisition, conversion, and processing tasks essential in vision systems.
- **SciPy:** Applied for its extensive scientific computing tools, aiding in mathematical transformations and algorithm development.



```
[INFO] [gzclient-2]: process has finished cleanly [pid 17413]
^C[WARNING] [launch]: user interrupted with ctrl-c (SIGINT)
[gzserver-1] [INFO] [1715824743.609108579] [rclcpp]: signal_handler(signum=2)
[ERROR] [gzserver-1]: process[gzserver-1] failed to terminate '5' seconds afte
receiving 'SIGINT', escalating to 'SIGTERM'
[INFO] [gzserver-1]: sending signal 'SIGTERM' to process[gzserver-1]
[gzserver-1] [INFO] [1715824748.619044937] [rclcpp]: signal_handler(signum=15)
[ERROR] [gzserver-1]: process[gzserver-1] failed to terminate '10.0' seconds a
er receiving 'SIGTERM', escalating to 'SIGKILL'
[INFO] [gzserver-1]: sending signal 'SIGKILL' to process[gzserver-1]
[ERROR] [gzserver-1]: process has died [pid 17411, exit code -9, cmd 'gzserver
home/alexmanson/Documents/stanford/dg_ws/install/data_generation/share/data_ge
ration/Worlds/ycb.world -slibgazebo_ros_init.so -slibgazebo_ros_factory.so -sl
gazebo_ros_force_system.so'].
(base) alexmanson@alexmanson-Dell-G15-5530:~/Documents/stanford/dg_ws$ ros2 la
ch data_generation metrics.launch.py world_name:=ycb.world
[INFO] [launch]: All log files can be found below /home/alexmanson/.ros/log/20
-05-16-09-59-20-095447-alexmanson-Dell-G15-5530-17712
[INFO] [launch]: Default logging verbosity is set to INFO
World name: <launch.substitutions.launch_configuration.LaunchConfiguration obj
t at 0x760e912e20b0>
[INFO] [gzserver-1]: process started with pid [17713]
[INFO] [gzclient-2]: process started with pid [17715]
```

Figure 3: Simulation Setup

4.3. System Pipeline

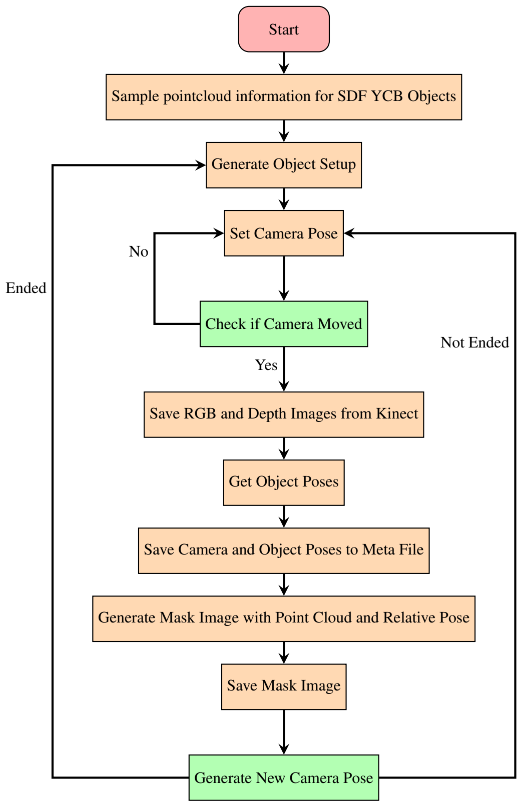

Start
Sample pointcloud information for SDF YCB Objects
Generate Object Setup
Set Camera Pose
No
Ended
Check if Camera Moved
Not Ended
Yes
Save RGB and Depth Images from Kinect
Get Object Poses
Save Camera and Object Poses to Meta File
Generate Mask Image with Point Cloud and Relative Pose
Save Mask Image
Generate New Camera Pose

Figure 4: Synthetic Dataset Generation Pipeline

The flowchart illustrates the pipeline of the data generation system. More details on each step will be explained in the following sections.

- Object selection and placement is outlined in 4.4
- Semantic label segmentation via mask generation is described in 4.5
- Computing the camera pose and covering entire environment is introduced in 4.6

4.4. Object Setup Auto-Generation and Collision handling

When setting the object enviornment, the system uses a fully automatic pipeline that is designed to generate occluded scenarios with different YCB objects. It begins by randomly selecting objects from the pool along with their poses. To avoid collisions, the system employs a checker function that resamples an object's pose if a collision is detected. The objects are then spawned using the `ros_gazebo_interface` functions `SpawnEntity` and `DeleteEntity` from the plugin `libgazebo_ros_factory.so`. Additionally, a small fall is designed to create potential top-down occlusion (where objects lay on top of other objects).

This toolbox supports up to 15 object types and includes multiple parameters to adjust and create different types of occluded scenarios, as listed below:

Object Name	
Banana	banana
Apple	apple
Bleach Cleanser	bleach_cleanser
Bowl	bowl
Cracker Box	cracker_box
Gelatin Box	gelatin_box
Master Chef Can	master_chef_can
Mustard Bottle	mustard_bottle
Pitcher Base	pitcher_base
Potted Meat Can	potted_meat_can
Pudding Box	pudding_box
Sugar Box	sugar_box
Tomato Soup Can	tomato_soup_can
Tuna Fish Can	tuna_fish_can

Table 1: Supported YCB Objects

Parameter	**Description**
$d_{\text{collision}}$	Collision check threshold (distance to allow between objects, influencing occlusion level)
$p_{\min}, p_{\max}$	Min and max thresholds for object position(influencing object density)
$n_{\min}, n_{\max}$	Min and max number of objects in the scene(influencing object density)
h_{fall}	Fall height (influencing top-down occlusion)

Table 2: Adjustable Parameters for Occluded Scenarios

4.5. Binary Mask Generation

Point clouds, derived from 3D object meshes (dae) using Open3D, are projected onto the image plane given the relative camera intrinsic matrix, camera pose and object pose. The detailed steps are as follows:

1. *Projection and Transformation:* Point clouds are first projected into the camera's coordinate system using transformation matrices that consider the camera and object poses. This alignment ensures that the point cloud data correspond accurately to the camera's view, matching the same viewpoint in color information (RGB) and depth image.
2. *Optical Adjustment:* The points are then adjusted for the camera's optics. This involves transforming the coordinates to align with the camera's intrinsic properties, such as focal length and distortion, facilitated by the camera's projection matrix. This step converts the 3D point coordinates into 2D pixel coordinates on the image sensor.
3. *Clipping and Mask Creation:* After projection, points that fall outside the camera's view are clipped, and the remaining points are used to create a binary mask. Each point corresponds to a pixel in the image, which is set to a high value to indicate the presence of the object, thus forming the binary mask.
4. *Flood filling[10]:* The flood filling algorithm is called to fill empty space in point cloud projection. This increases the processing speed as less points can be sampled in the first stage of sampling pointcloud information. In addition, it improves the quality of output mask image by preventing holes and noise.
5. *Layering:* Lastly, the masks for different objects are layered on top of each other with varying intensities to differentiate multiple objects within the same scene. This layering helps in identifying individual objects in occluded or densely populated scenes.

4.6. Camera Angle Sampling

The data generation toolbox provides two sampling methods for camera position: the Grid-Based CamPos sampling (Space-filling/ Full Factorial) and the Random CamPos sampling. The Grid-Based system strategically positions the camera at fixed intervals across a predefined grid, which ensures comprehensive coverage of the viewing field and results in adequate coverage over the design space. This method adjusts the camera's position and orientation to cover all viable angles, and is ideal for training direction vector outputs due to its predictable camera transitions. In contrast, the Random system employs a stochastic approach to camera positioning by generating camera angles and distances randomly within specified bounds. This approach introduces diverse viewing angles and distances, which aids in model generalization and robustness testing by including challenging and unconventional views.

The camera position is primarily determined by three variables: distance ($dist$), view direction (θ), and pan angle (ϕ). The distance represents how far the camera is from the center of the object setup, the view direction indicates the angle at which the camera is filming the setup in world coordinates, and the pan angle is the angle relative to the horizontal plane. The final camera pose is calculated using Equation 9, 10, where h_{table} is the height of the table YCB objects stand on.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} d\cos(\phi)\cos(\theta) \\ d\cos(\phi)\sin(\theta) \\ |d\sin(\phi)| + h_{table} \end{bmatrix} \tag{9}$$

$$\begin{bmatrix} q_x \\ q_y \\ q_z \\ q_w \end{bmatrix} = \text{Quaternion from Euler angles: } \begin{bmatrix} 0 \\ \phi \\ \theta + 180^\circ \end{bmatrix} \tag{10}$$

5. Experiments

5.1. Synthetic Dataset Generation Tuning

To generate best dataset for enhancement on PoseCNN, tuning and experiments are done to ensure the quality of final output.

At first, using random sampling will cause a lot of problems including interpenetration, object distortion due to collision, and some thinner objects could be easily interpenetrated when falling (see Figure 5).

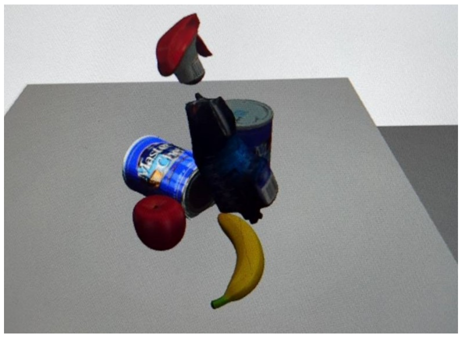


Figure 5: Interpenetration and object distortion

Interpenetration and distortion are fixed by tuning suitable threshold for object position, and maximum number of object to prevent to many objects spawn together. A collision check is introduced to prevent spawning objects within other object. Also thinner object penetration problem is solved by lowering falling and setting time lag in between each spawn. In addition, some object with bad quality 3d models and collision boundary set are removed from the list. The final list of object accepted is listed in 1. And the final parameters is as following Table 3:

Parameter	**Tuned Value**
$d_{\text{collision}}$	0.12
$p_{\text{min}}, p_{\text{max}}$	-0.15, 0.15
$n_{\text{min}}, n_{\text{max}}$	2, 6
h_{fall}	1.3

Table 3: Tuned Parameters for PoseCNN Enhancements

For the final binary mask generation, the initial images are very noisy and each segment has holes, especially when the camera is too close to the objects, as shown in Figure 6. This problem is resolved by using flood filling, increasing the sampling number from the point cloud to generate the mask, and adjusting the grid-based parameters in camera pose sampling as shown in Table 4, where init indicates the initial values, max represents the maximum values, and increment specifies the step size for adjusting the camera pose parameters in grid sampling. The final result shown in Figure 7.

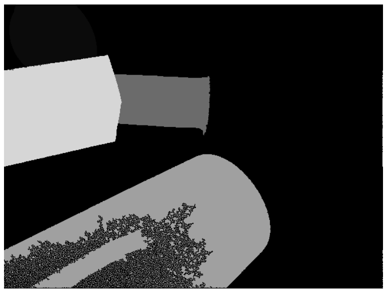


Figure 6: Binary Mask Image Generation with holes and noise

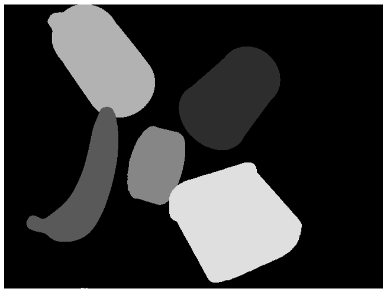


Figure 7: Fixed Binary Mask Image Generation

5.2. Synthetic Dataset Generation Final Result

As the result, we generated a comprehensive synthetic dataset comprising 2G of meticulously crafted scenarios

Parameter	Tuned Value
ϕ_{init}	35
θ_{init}	0
d_{init}	0.4
ϕ_{max}	75
θ_{max}	360
d_{max}	1.2
$\phi_{\text{increment}}$	5
$\theta_{\text{increment}}$	30
$d_{\text{increment}}$	0.2

Table 4: Tuned Parameters for PoseCNN Enhancements

featuring 25 YCB object setups in under 12 hours. This synthetic dataset encompasses a diverse range of camera angles, curated to simulate the real-world settings and complex occlusion scenarios, thereby facilitating robust object pose estimation. Given more time and processing power, more can be generated for further usage if needed. The user also could generate more dataset and adjust the parameter to adapt the toolbox to the need of the application, generating setup for different occlusion scenario.

In addition, the toolbox also provided wide range of utils functions to facilitate the training for different robotic application. This includes functions to find the image in the grid based on movement, to find the distance between camera and the specific target, to generate customized pointcloud sampling for mask image generation from different model type (stl/dae), and etc.

The examples of the generated dataset are shown in Figure 8,9,10.

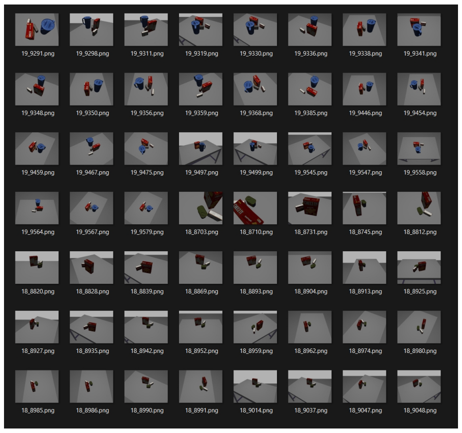


Figure 8: RGB images generated by Synthetic Dataset Generation Toolbox

5.3. Challenges and Constraints

Throughout the process, we encountered several constraints and identified areas for improvement in future.

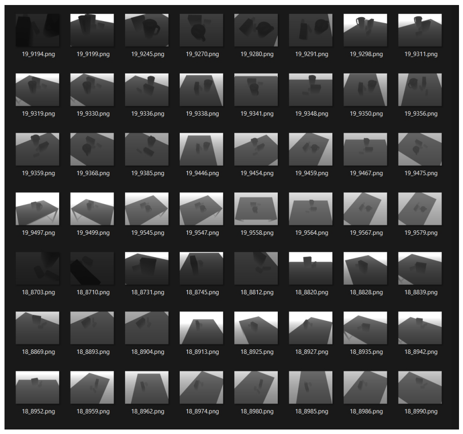


Figure 9: Depth images generated by Synthetic Dataset Generation Toolbox

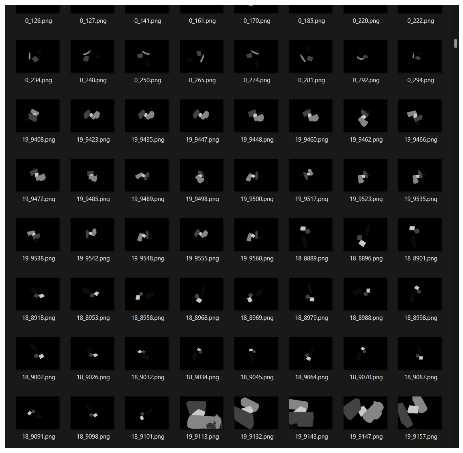


Figure 10: Semantic labeled (Binary Mask) images generated by Synthetic Dataset Generation Toolbox

These challenges primarily revolved around adapting existing network setup[11], including:

- **Expired Training Dataset and Pretrained Weights Link**: The expired links to the training dataset and pretrained weights had hinder our ability to access the necessary data and pre-trained models required for training. Furthermore, the environment and code setting would violate the current packages and more time was spent on debugging.

- **Incorrect Training Step**: It's possible that the training step we followed was not correct or optimal. Training deep learning models requires careful attention to details such as hyperparameters, data preprocessing, model architecture, loss functions, and optimization algorithms. In particular, the dataset cleaning and a better data selection should be performed.

- **Time Constraints**: Training deep learning models can be computationally intensive and time-consuming, especially for large datasets and complex models. It might be beneficial to train the model with a slimmer dataset or tuning epochs and hyperparameters to get a better iteration of the data.

- **Environment Setup Challenges**: Setting up the development environment for training deep learning models can be challenging, particularly when dealing with GPU drivers, CUDA, and Python libraries. Compatibility issues, installation errors, and configuration problems arose, leading to delays and difficulties in getting started with training.

6. Conclusion

In this work, we've utilized Gazebo for synthetic data generation, focusing on generating data with more meta data to improve model accuracy. A wide range of utils functions are provied in the toolbox for further training in robotic application. As a result, we generated a comprehensive synthetic dataset comprising 2G of meticulously crafted scenarios featuring 25 YCB object setups in under 12 hours.

For further improvement, the issue of holes in the mask image generation still occasionally occurs. In addition to parameter adjustments, implementing a more effective hole-filling algorithm or a checker to monitor the distance between the camera and objects should be considered. This checker should dynamically adjust its grid setup or alternatively skip the sample (or mark it as invalid).

Additionally, the YCB video dataset [11] uses a complex background environment like messy table or bedroom to train a more generalized model. Our system should incorporate similar complexity to enhance model generalization and create a more realistic occluded scene.

Lastly, due to the poor quality of the collision boundary setup and 3D models, only 14 YCB models are currently supported. Expanding this support to include more models in the future is necessary.

In terms of PoseCNN enhancement, due to the several constraints and time limits, some proposed enhancement including probablistic model and direction vector are not implemented. These can be done in the future as a promising direction.

References

[1] E. Brachmann. 6D Object Pose Estimation using 3D Object Coordinates [Data], 2020.

[2] A. Calli and e. a. Singh. Yale-cmu-berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research*, 36(3):261–268, 2017.

[3] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 510–517, 2015.

[4] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics Automation Magazine*, 22(3):36–52, 2015.

[5] CentralLabFacilities. gazebo ycb. `https://github.com/CentralLabFacilities/gazebo_ycb.git`, 2020.

[6] C. Ke. ycb gazebo sdf. `https://github.com/chengkecodes/ycb_gazebo_sdf.git`, 2020.

[7] B. Leibe, A. Leonardis, and B. Schiele. Combined object categorization and segmentation with an implicit shape model. In *ECCV Workshop on Statistical Learning in Computer Vision*, 2004.

[8] NVlabs. Posecnn-pytorch. `https://github.com/NVlabs/PoseCNN-PyTorch.git`, 2019.

[9] E. R. Saad Ahmad, Kulunu Samarawickrama and R. Pieters. Automatic dataset generation from cad for vision based grasping. In *20th International Conference on Advanced Robotics (ICAR)*, December 2021.

[10] A. S. Tanenbaum and D. J. Wetherall. *Computer Networks*. Pearson Education, 5th edition, March 23 2010.

[11] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. 2018.