Object 6D Pose Recognition and Grasping Based on Machine Learning

Dongrui Li 12112208 Zhaokai Chen 12212220 Songran Wang 12010114
Haoyang Xiang 12010611 Zuo Xu 12111625 Yifei Luo 12012039

Abstract—**The study of 6D pose estimation methods is crucial for enhancing robotic perception and manipulation capabilities, particularly in complex environments. Initially, we employed the Deep Object Pose Estimation (DOPE) project within the Gazebo simulation environment to recognize and grasp objects. However, DOPE demonstrated poor performance in highly cluttered or occluded scenarios. To address these challenges, we turned to DenseFusion, a more robust approach, and thoroughly examined its underlying paper and code.Due to limitations in computational resources and time, we focused on a single object and adjusted the parameters to expedite the training and evaluation processes. We successfully trained the DenseFusion model, conducted evaluations, and visualized the results. We compared the performance of our modified model with the official DenseFusion model, observing that while our adjustments improved speed and feasibility, the official model outperformed ours in terms of accuracy and robustness in evaluation and visualization tasks. This comparison highlights the trade-offs between model optimization and performance in practical applications.**

I. Introduction

With the continuous deepening of the research in the field of artificial intelligence, robot technology is gradually expanding from the traditional industrial field to a wider range of application scenarios. Robots have become an important tool in intelligent manufacturing, automated production and service industries, and their applications range from industrial assembly lines to home services, medical care and other fields. In the research of intelligent robot, intelligent autonomous grasping is a key task, which requires the robot to accurately identify and grasp the target object in complex and changeable environment. The core of intelligent autonomous grasping is accurate target monitoring and grasping pose estimation, which is very important to realize the intelligent operation of robot. In unstructured scenes, objects with different shapes, random positions and poses and mutual occlusion put forward higher requirements for multi-target monitoring and grasping of robots. The traditional robot grasping system usually depends on the preset model and path, which is unable to meet the needs in the unstructured environment. Therefore, the development of intelligent capture system with adaptive ability and ability to deal with changing environment has become a hot and difficult point in the current research. In order to meet these challenges, this project aims to develop a 6-DOF robot arm vision system based on DOPE robot design framework, which is committed to high-precision visual servo control. The project uses RGB-D camera to recognize and locate multi-target objects, and introduces the Densefusion[3] deep learning attitude estimation algorithm. Densefusion attitude estimation algorithm can predict the attitude of unseen objects in any environment, which improves the flexibility and adaptability of the system. In addition, the project will also establish and analyze the spatial description and motion model of the six degree of freedom manipulator, and use the improved path planning algorithm to ensure the accurate recognition, positioning and grasping of the target object in the simulation environment. Specifically, this project studies the design and Simulation of the six degree of freedom manipulator vision system, including the selection of components, camera imaging model and internal parameter calibration, target recognition and positioning, manipulator kinematics modeling and improved path planning algorithm. The RGB-D camera and Densefusion deep learning attitude estimation algorithm are used to identify and locate the target object. The DOPE framework is used to establish the model of the manipulator and optimize the path planning algorithm to achieve accurate target monitoring and capture.

II. Method

A. Related works

6D object pose can be accurately estimated from rgb-d data, which is the most commonly used method in the capture system. If there is a 6DOF (six degrees of freedom) grasp posture in the database, the current 6DOF grasp posture can be retrieved from the knowledge base according to the complete shape, or obtained by sampling and sorting compared with the existing grasp. If the 6DOF capture attitude does not exist in the database, the analysis method is used to calculate the capture attitude. These methods consider kinematics and dynamics formulas to determine the grasping posture [sahbani et al., 2012]. Both traditional and deep learning based 6D object pose estimation algorithms are used to assist robots in grasping tasks. Most of the methods proposed in Amazon picking challenge [zeng et al., 2017b] first estimate 6D attitude through partial registration. Zeng et al. [zeng et al., 2017b] proposed a method, which uses full convolutional neural network to segment and mark multiple views of the scene, and then fit the pre scanned 3D object model into the segmentation results to obtain the 6D object pose. In addition, billings and Johnson Roberson [billings and Johnson Roberson, 2018] proposed a method that uses convolutional neural network (CNN) pipes to jointly realize object pose estimation and grab point selection. Wong et al. [wong et al., 2017] proposed a method that integrates RGB based object segmentation and depth image based partial registration to obtain the pose of the target object. They proposed a novel index to evaluate the quality of model registration, and carried out multi hypothesis registration to achieve accurate attitude estimation with 1cm position error and¡5 ° angle error. Using this accurate 6D object pose, you can grab with a high success rate. Some 6D object pose estimation methods based on deep learning, such as densefusion[wang et al., 2019b], also show high success rate in the implementation of actual robot capture tasks.

B. *Theory of 6D pose estimation model*

We employed a dense fusion model for 6D pose estimation, which includes two stages. In the first stage, an encoder-decoder structured semantic segmentation network is used to perform semantic segmentation of the input color image based on color and depth, followed by mask cropping. This network contains N+1 channels, where N represents the number of features to be segmented. After segmentation, the image undergoes mask cropping, and information is extracted from the color and depth channels. The data from the depth channel is converted into a three-dimensional point cloud for further processing.

In the second stage, dense feature extraction is performed on the segmentation results from the first stage. For color information, a convolutional neural network-based architecture is used to map the color image into an embedding space, where each pixel in the embedding space is a vector representing the appearance information at that position. For embedding depth information, the PointNet[1] architecture is employed to extract geometric features. Unlike the standard PointNet, average pooling is used instead of max pooling.

To address potential occlusions and segmentation errors in the image, which may result in local errors in the dense features from the previous step, a pixel-wise fusion method is adopted. [2]This method associates the image and geometric features of the corresponding pixels to obtain dense pixel features, which are then input into another neural network to obtain globally fused features for pose estimation.

During pose estimation, a pose loss minimization is defined, and an iterative algorithm is used to continuously refine the prediction results. A residual estimation network is used in the prediction process, which is typically trained after the main network has converged. This approach effectively improves the accuracy of 6D pose estimation in complex scenes.

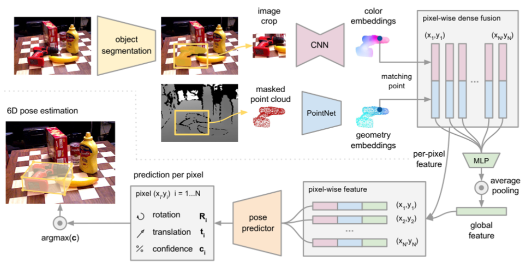


Fig. 1. Overview of the 6D pose estimation model

III. Experiment

A. *DOPE Experiments*

DOPE is a two-step solution to address the problem of detecting and estimating the 6-DoF pose of a set of known household objects from a single RGB image. First, a deep neural network estimates belief maps of 2D keypoints of all the objects in the image coordinate system. Secondly, peaks from these belief maps are fed to a standard perspective-n-point (PnP) algorithm to estimate the 6-DoF pose of each object instance. In this section we describe these steps, along with the simulation of this novel method.

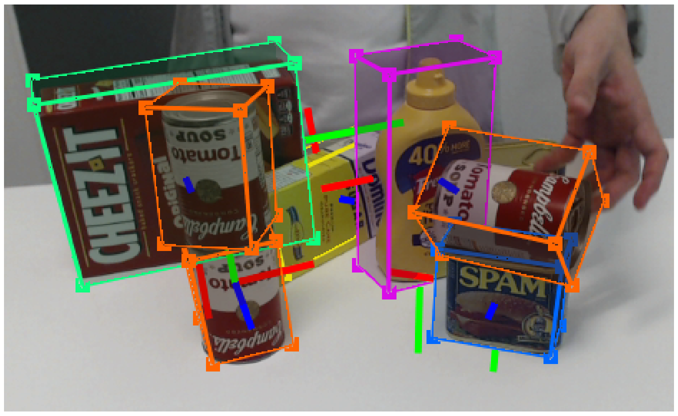


Fig. 2. 6D pose estimation by DOPE

*1) **Experimental Setup**:* The simulation is performed on Ubuntu 20.04 with python 3.8.19, ROS Noetic 1.16. Pakages including Pytorch 2.30, CUDA 12.0 are used in this project. We add the ur5 robotic arm, robotiq 2F-85 gripper and realsense d435 depth camera to the gazebo environment to complete the simulation.

In DOPE experiments, we use the pre-trained DOPE model for simulation in gazebo simulation environment.DOPE is trained with two publicaly available datasets: YCB, and HOPE. We chose the one trained with HOPE dataset, which is included in the attachment.

The HOPE dataset is a collection of rgbd images and video sequences with labeled 6-DoF poses for 28 toy grocery objects. To complete the simulation, we downloaded the 3d model from the Internet and these models are in .obj format. From these objects,we chose milk model to add to the simulation environment.

*2) **Simulation Environment**:* The simulation environment is gazebo-11 on ROS Noetic 1.16. The robot arm is control by MoveIt plugin.Firstly, we obtain urdf description files of ur5, robotiq gripper and d435 camera from github, then we put these models into a world file named ’pick_and_place.world ’,which is used to create a simulation world including a table and the milk model in gazebo. The whole scene is showed in the following picture.

After preparing for the simulation environment, I use MoveIt for robot path planning and motion control to achieve

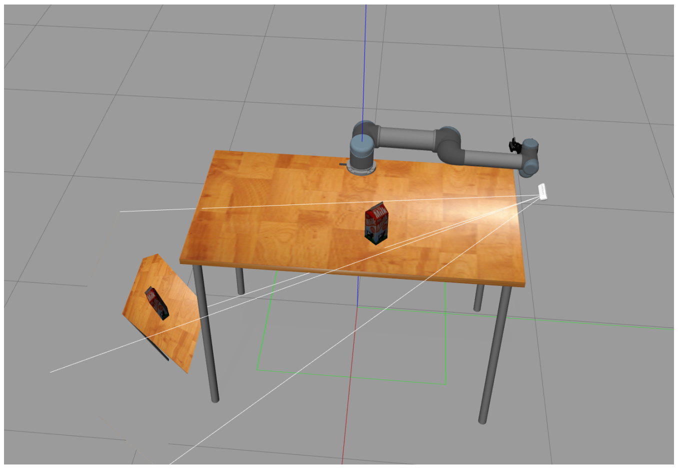
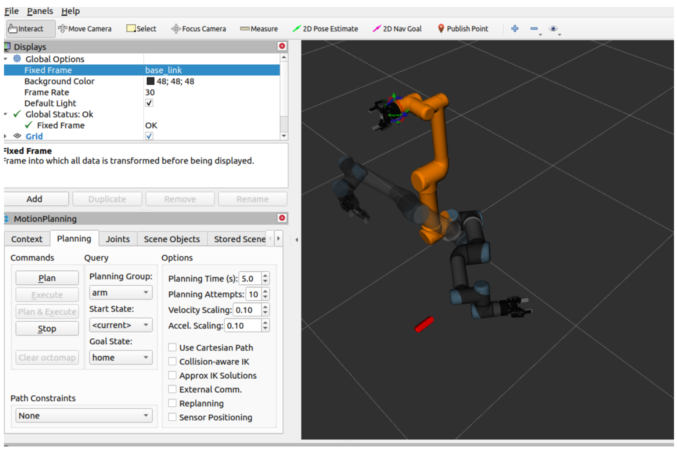


Fig. 3. Simulation environment in Gazebo

object grasping. MoveIt is a powerful robot motion planning framework that supports complex arm motion planning, collision detection, and real-time kinematics calculations. By applying MoveIt, we are able to interact with gazebo and receive object position and orientation data from the DOPE system, plan the motion trajectory of the robotic arm, and ensure that the arm accurately reaches the specified 3D space position and grasps the target object with the predetermined pose.



Global Options
Fixed Frame base_link
Background Color 48; 48; 48
Frame Rate 30
Default Light ✓
Global Status: Ok
Fixed Frame OK
Grid ✓

Fixed Frame
Frame into which all data is transformed before being displayed.

Planning Group: arm
Start State: <current>
Goal State: home
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.10
Accel. Scaling: 0.10
☐ Use Cartesian Path
☐ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning
Path Constraints: None

Fig. 4. Moveit for path planning and motion control

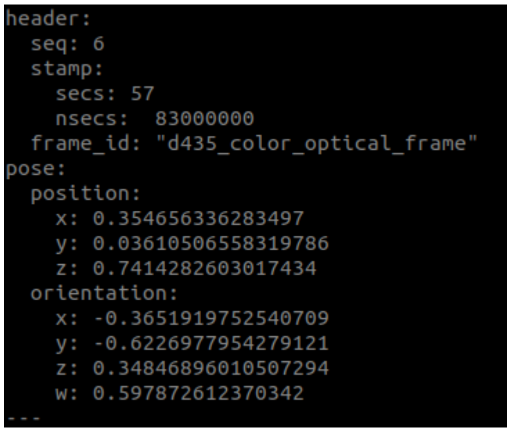
The next steps we focus on refining the integration between the DOPE system and the MoveIt controlled robotic arm to enhance the efficiency and accuracy of the object grasping process, such as adjust the pose of the milk box to ensure that its orientation can be correctly predicted.

3) System Architecture: The DOPE GitHub open-source project provides a ROS interface for our simulation. Through publisher and subscriber nodes, we use the DOPE model to receive image data published by the camera, perform 6D pose prediction, and then provide it to the robotic arm for grasping.

Our project's engineering files mainly consist of model files, MoveIt configuration files, DOPE model files, and code files. The model files describe the pose, joints, collision properties, and rotational inertia of the robotic arm, gripper, and camera within the simulation environment. MoveIt configuration files set up joint controllers and kinematics solvers and include MoveIt's setup files. The DOPE model files contain the model's training files, pre-trained .pth weight files, prameter files used in codes and the 3D model files used for pose estimation. In this section, we focus on the code implementation of the project.

The project's code section includes five code files, among which three scripts function as libraries called by the core ROS topic publisher, 'dope.py'. These are used to calculate the 6D pose of items and to draw cuboid meshes, marking the object's position. The remaining script subscribes to the object 6D pose information published by 'dope.py' and import the moveit_commander package to control the robotic arm for grasping tasks.

dope.py This code serves as the core, subscribing to the RGB information transmitted by the camera. After processing, it determines the 6D pose of objects within the camera's coordinate frame, 'd435_color_optical_frame', and publishes this data on the topic 'dope_MILK_pose'. Additionally, the script publishes other information useful for pose estimation visualization, which will be displayed in rviz in the following section.



```
header:
  seq: 6
  stamp:
    secs: 57
    nsecs:  83000000
  frame_id: "d435_color_optical_frame"
pose:
  position:
    x: 0.354656336283497
    y: 0.03610506558319786
    z: 0.7414282603017434
  orientation:
    x: -0.3651919752540709
    y: -0.6226977954279121
    z: 0.34846896010507294
    w: 0.597872612370342
---
```

Fig. 5. 6D pose of objects within the camera's coordinate frame

In 'dope.py', the following three codes are invoked to implement the estimation and visualization of an object's 6D pose

1. detector.py This code is responsible for loading and managing the model data of the object detection neural network. It maintains the weights and configuration state of the network model, providing an interface for loading and accessing the neural network model. Also, it contains the main methods for detecting and recognizing objects in images. Using the trained neural network model, it processes the input images to generate confidence maps and affinity fields, ultimately identifying the target objects in the images and estimating their poses.

2. cuboid_pnp_solver.py This code serves as a solver for the Perspective-n-Point (PnP) problem that provides accurate mapping from 2D images to the 3D world, can be used to calculate the position and rotation of the object in the camera coordinate system. By combining 2D image points with corresponding 3D model points, CuboidPNPSolver can accurately calculate the object's pose, which is key to inferring three-dimensional spatial poses from single view images.

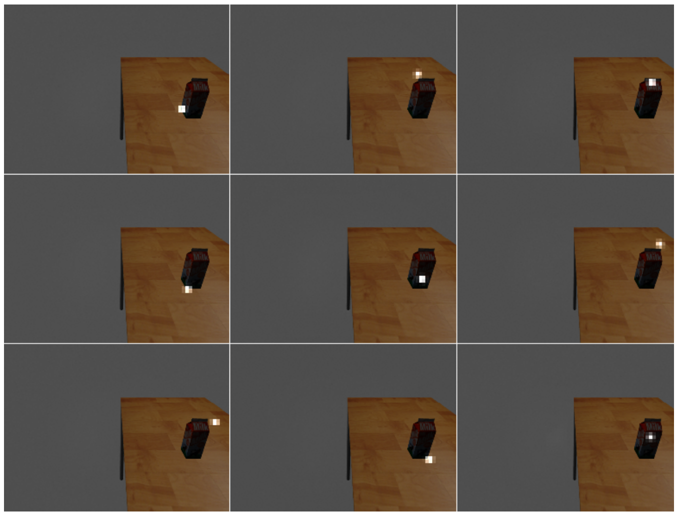


Fig. 6. Positions of eight vertices and geometric center of the cuboid in 2D

3. cuboid.py This code defines a 3D model of a cuboid or rectangular prism. It primarily provides the three dimensional structure of the object, including the coordinates of its vertices. These 3D coordinates are used to match with 2D image coordinates, which supports pose estimation and projection calculations.

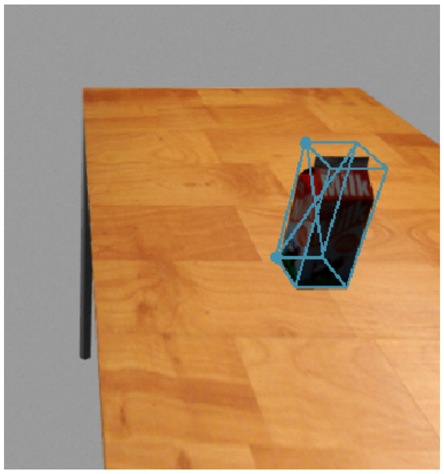


Fig. 7. 3D cuboid used to describe 6D pose of the object

pickup.py This code is a ROS node that subscribes to the 'dope_pose_MILK' topic for the object's 6D pose in the camera coordinate system and converts it to the robot arm's 'base_link' coordinate system. It then apply MoveIt to control and grasp with the robotic arm.

*4) **Experimental Result:*** To complete the experimental section, we first run the code, opening Gazebo, rviz, and running the 'dope.py' code to establish communication between nodes. At this point, the estimation of the 6D pose has been completed and published. Then we open rviz and can visualize the prediction results by subscribing to 'dope_markers' and 'dope_pose_milk', comparing the predicted results in rviz with the actual results graphically. It can be seen that the pose of the two are very close. See in Fig 8 and Fig 9.

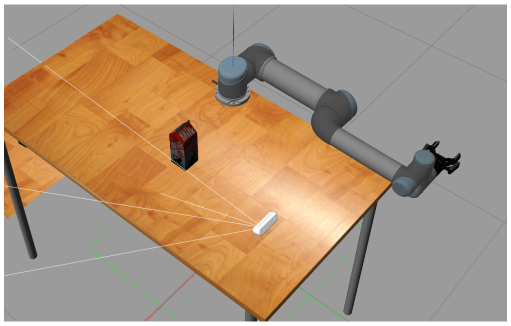


Fig. 8. Real world pose of milk box

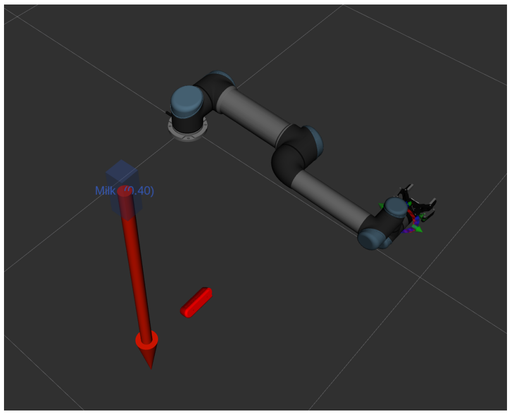


Fig. 9. 6D pose of the object predicted by DOPE

After completing the information transfer between nodes, we run the 'pickup.py' script to perform the grasping operation. See in Fig 10 and Fig 11

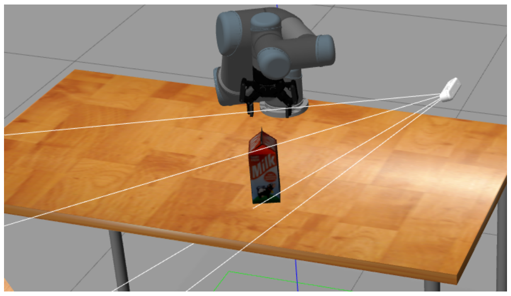


Fig. 10. Grasping task performed in gazebo

It can be observed that under control, the robotic arm arrives at a position 0.3 meters above the geometric center

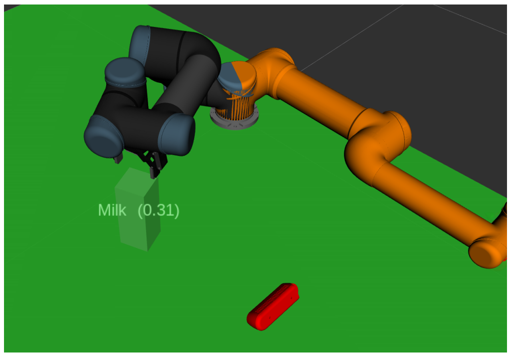


Fig. 11. Grasping task visualized in rviz

of the milk carton with the correct orientation, and then descends to perform the grasping operation. However, in our multiple attempts, we have not successfully grasped the milk carton, possibly because the downloaded model is relatively large compared to the gripper, preventing a successful grasp.Nevertheless, it is evident that the pose predicted by DOPE is relatively accurate.

In other attempts, pose prediction failures generally occurred due to poor placement of the model, the model being too far away, or being directly in front of the camera. Additionally, occlusions can also cause prediction failures. It is evident that DOPE's 6D pose estimation has high requirements for the RGB images captured by the camera. Therefore, we experimented with a second model, DenseFusion.

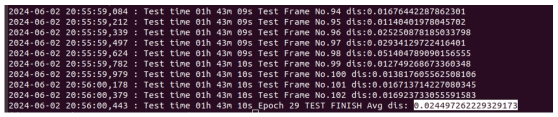
B. DenseFusion Experiments

1) Experimental Setup:

- The project required the installation of several libraries and tools, including Python, PyTorch, PIC, scipy, numpy, pyyaml, logging, and matplotlib. Additionally, NVIDIA drivers and CUDA 10.0 were essential for leveraging GPU acceleration. Due to the project's age, specific library versions were required for compatibility. The final versions we used were: Python 3.6.15, PyTorch 1.0.0, scipy 1.5.2, numpy 1.19.2, pyyaml 6.0.1, logging 0.5.1.2, matplotlib 3.3.2, NVIDIA driver 470.223.02, CUDA 10.0.
- The DenseFusion project was evaluated on both the YCB and Linemod datasets, performing exceptionally well. However, due to the large size of the YCB dataset (up to 265GB), we opted to use the preprocessed Linemod dataset provided by DenseFusion. This dataset was placed in the project's './datasets/linemod' directory.
- The dataset consists of three main folders: 'data', 'models', and 'segnet_results'. The 'data' folder contains 13 subfolders, each corresponding to an object. Each object's folder contains 'depth', 'mask', and 'rgb' subfolders, representing depth images, mask images, and RGB images of the object captured from over 1000 viewpoints. Additionally, there are four files in each object's folder containing the object's rotation matrix, translation matrix, standard box, object category, camera intrinsic parameters, scaling factor, and the pre-split test and training datasets. The 'model' folder contains '.ply' files representing the point cloud information of each object. The 'modelst_info.yml' file contains radius and dimensions information for each object's point cloud model. The 'segnet_results' folder contains images segmented by the semantic segmentation network.
- The original DenseFusion code was written for PyTorch 0.4. However, we used PyTorch 1.0, which required recompiling knn in the './lib/knn' directory, we executed the following commands:
  ```
  python setup.py build
  python setup.py install
  ```
  After execution, a dist folder appeared in the './lib/knn 'directory, containing a compiled '.egg' file. We extracted this file and moved it to the './lib/knn' directory.

2) Training:

- The initial training attempts encountered significant issues due to the large dataset size, which resulted in insufficient GPU memory and subsequent errors. To address this, reducing the batch size was considered. however, this approach led to a substantially slower training process, rendering it impractical for timely completion.
- To address these limitations, the training process was modified by focusing on a single object, specifically the second object in the dataset. Additionally, several parameters were adjusted to optimize the training efficiency. The number of points in the point cloud was reduced from 500 to 100, the number of epochs per training iteration was decreased from 20 to 5, and the maximum number of epochs was lowered from 500 to 30. These adjustments significantly reduced the computational load, allowing the training to proceed without memory issues. Executed the following commands:
  ```
  python3 ./tools/train.py
  --dataset linemod
  --dataset_root
  ./datasets/linemod/Linemod_preprocessed
  --batch_size 8
  ```
- As a result of these optimizations, the training duration was significantly reduced. Previously, each epoch required approximately 30 minutes to complete. With the new settings, the time per epoch was reduced to approximately 5 minutes. This improvement enabled the completion of 30 epochs in roughly 3 hours, resulting in a trained model 'pose_model_current.pth' saved in './trained_models/linemod/'.



```
2024-06-02 20:55:59,084 : Test time 01h 43m 09s Test Frame No.94 dis:0.01676442287862301
2024-06-02 20:55:59,212 : Test time 01h 43m 09s Test Frame No.95 dis:0.01140401978045702
2024-06-02 20:55:59,339 : Test time 01h 43m 09s Test Frame No.96 dis:0.025250878185033798
2024-06-02 20:55:59,497 : Test time 01h 43m 09s Test Frame No.97 dis:0.02934129722416401
2024-06-02 20:55:59,624 : Test time 01h 43m 09s Test Frame No.98 dis:0.051404789090156555
2024-06-02 20:55:59,782 : Test time 01h 43m 10s Test Frame No.99 dis:0.012749268673360348
2024-06-02 20:55:59,979 : Test time 01h 43m 10s Test Frame No.100 dis:0.013817605562508106
2024-06-02 20:56:00,178 : Test time 01h 43m 10s Test Frame No.101 dis:0.016713714227080345
2024-06-02 20:56:00,379 : Test time 01h 43m 10s Test Frame No.102 dis:0.016923733055591583
2024-06-02 20:56:00,443 : Test time 01h 43m 10s Epoch 29 TEST FINISH Avg dis: 0.024497262229329173
```

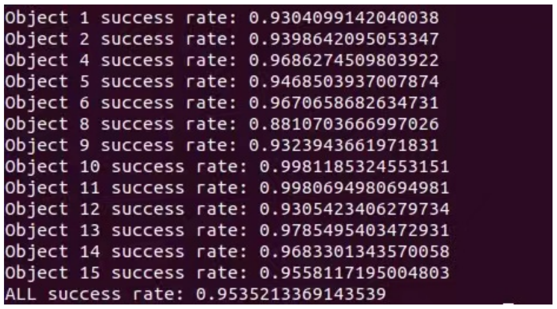
Fig. 12. The final average distance error for the last epoch was 0.024.

- However, despite the improvements in training efficiency, the final average distance error did not meet the threshold of the default refine margin set at 0.013. Consequently, the model did not progress to the refinement stage, and no refined model was obtained. This limitation highlights the trade-off between training efficiency and model accuracy, suggesting the need for further adjustments and optimizations in future iterations.

3) Evaluation:

- To determine the accuracy of pose estimation, DenseFusion adopts a threshold-based criterion utilizing the 3D bounding spheres of the objects. Specifically, for each object class, it first computes the diameter of the minimum bounding sphere encompassing the 3D model. An estimated pose is deemed correct if the distance between the predicted and ground-truth object positions is within 10 % of this diameter.
- We initially evaluated the performance of the official DenseFusion model, which was trained using the standard parameters and methodology provided by the authors. The evaluation was performed using the following command:

```
python3 ./tools/eval_linemod.py
--dataset_root
./datasets/linemod/Linemod_preprocessed
--model
trained_models/linemod/
pose_model_current.pth
--refine_model
trained_models/linemod/
pose_refine_model_current.pth
```

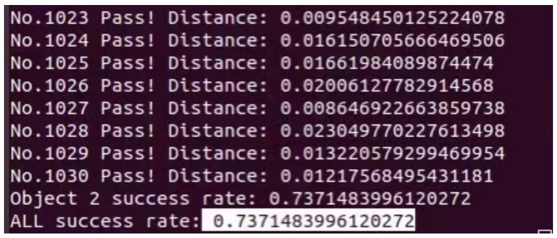


```
Object 1 success rate: 0.9304099142040038
Object 2 success rate: 0.9398642095053347
Object 4 success rate: 0.9686274509803922
Object 5 success rate: 0.9468503937007874
Object 6 success rate: 0.9670658682634731
Object 8 success rate: 0.8810703666997026
Object 9 success rate: 0.9323943661971831
Object 10 success rate: 0.9981185324553151
Object 11 success rate: 0.9980694980694981
Object 12 success rate: 0.9305423406279734
Object 13 success rate: 0.9785495403472931
Object 14 success rate: 0.9683301343570058
Object 15 success rate: 0.9558117195004803
ALL success rate: 0.9535213369143539
```

Fig. 13. The average success rate of the models provided by the author is about 0.953

The evaluation results indicated an accuracy of 95%, demonstrating the effectiveness of the DenseFusion model in accurately estimating object poses in cluttered and occluded environments.

- Subsequently, we evaluated the performance of our modified DenseFusion model, which was trained with adjusted parameters to expedite the training process. Since the training process is based on only the object 2 and there is no refine stage, the evaluation was conducted on only the object 2 without a refined model. The evaluation revealed a lower accuracy of 70% compared to the official model. This decrease in accuracy can be attributed to the modifications made to the training parameters, which compromised the model's ability to refine its predictions and handle complex scenarios effectively.



```
No.1023 Pass! Distance: 0.009548450125224078
No.1024 Pass! Distance: 0.016150705666469506
No.1025 Pass! Distance: 0.01661984089874474
No.1026 Pass! Distance: 0.02006127782914568
No.1027 Pass! Distance: 0.008646922663859738
No.1028 Pass! Distance: 0.023049770227613498
No.1029 Pass! Distance: 0.013220579299469954
No.1030 Pass! Distance: 0.01217568495431181
Object 2 success rate: 0.7371483996120272
ALL success rate: 0.7371483996120272
```

Fig. 14. The success rate of our model is about 0.737

4) Visualization:

- In addition to quantitative metrics such as accuracy and precision, we implemented a visualization module to provide a more intuitive assessment of our trained model. This visualization tool allows us to observe the alignment between the predicted pose of the object and its ground truth bounding box, facilitating a visual understanding of the model's accuracy.
- One key aspect of our visualization is the rendering of the object's point cloud, which represents the object's 3D geometry. By visualizing the point cloud, we can directly compare the model's estimated pose with the ground truth.
- The main goal of the visualization is to align the predicted pose of the object with its ground truth bounding box. This alignment allows us to visually assess the accuracy of the model's pose estimation.

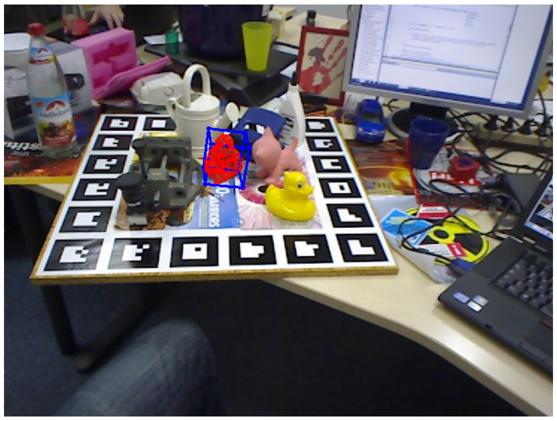


Fig. 15. Visualization using the models provided by the author

- To further evaluate the performance of our trained model, we compared its visualization results with those obtained using the pre-trained model provided by the authors. The pre-trained model has high accuracy in aligning the point cloud with the object and accurately estimating the 6D pose, as evidenced by the perfect alignment between the point cloud and the object model, and the precise positioning and orientation of the standard bounding box. In contrast, our trained model, while not achieving the same level of accuracy as the pre-trained model, still demonstrates notable performance. The point cloud aligns well with the object, and the standard bounding box

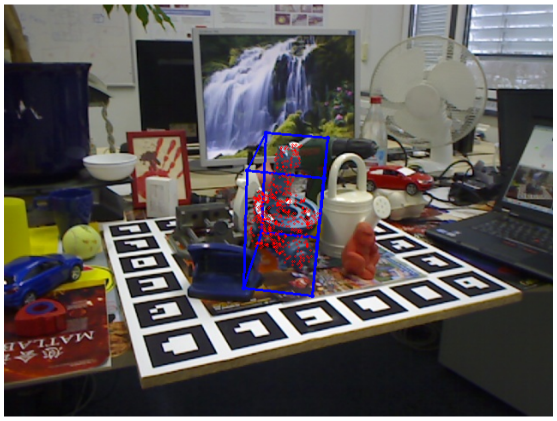


Fig. 16. Visualization using our model

is positioned and oriented accurately, albeit with some minor deviations.

IV. Conclusion

According to the literature, the DenseFusion algorithm exhibits strong robustness and can accurately predict object poses even in the presence of occlusions. Thanks to its pixel-level dense fusion technique, this algorithm can efficiently perform pose estimation, making it well-suited for integration into real-time systems for handling immediate tasks. However, this algorithm has specific requirements for data acquisition; it needs RGB-D images and high-quality data, along with multi-perspective camera setups, to achieve optimal performance. These make the algorithm suitable for use in fixed multi RGB-D environments for real-time issues such as logistics sorting and waste classification. However, due to the older nature of DenseFusion and the stringent version requirements for many necessary packages, the reproduction difficulty is high, which led us to choose a newer algorithm, DOPE, for simulation. After undergoing simulations with the DOPE algorithm, we found that using the officially provided pre-trained model, the average accuracy can reach about 95percent. After modifying some training parameters and retraining, the accuracy also reached around 70percent. According to the results and the literature, the DOPE algorithm is capable of performing pose estimation in real-world conditions under extreme lighting and background changes, and it can be fully trained on synthetic data with domain randomization and real data to enhance generalization capabilities. This algorithm requires only one-shot learning for efficient pose estimation, without iterative refinement. While using synthetic data can benefit training, it may not perform well in complex textured environments, and the algorithm is only optimized for known objects of existing models, which may pose challenges in dynamic environments. The DOPE algorithm is suitable for a wider range of scenarios because it only requires images for pose estimation, unlike the DenseFusion algorithm, which requires RGB-D cameras. It can be used in household scenarios such as interaction with objects by robotic vacuum cleaners.

Acknowledgments

References

[1] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. 2017.

[2] Tielin Zhang, Yang Yang, Yi Zeng, and Yuxuan Zhao. Cognitive template-clustering improved linemod for efficient multi-object pose estimation. Cognitive Computation, 12(4):834–843, 2020.

[3] C. Wang, D. Xu, Y. Zhu, R Mart´ın-Mart´ın, C. Lu, L. Fei-Fei, and S. Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. 2019.

[4] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6):381–395, 1981.

[5] W. He, S. Sridhar, J. Huang, J. Valentin, and L. J. Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. IEEE, 2019.

[6] K. Park, T. Patten, and M. Vincze. Pix2pose: Pixel-wise coordinate regression of objects for 6d pose estimation, 2019.

[7] S Saha, N. K Bambha, and S. S Bhattacharyya. Design and implementation of embedded computer vision systems based on particle filters. Computer Vision and Image Understanding, 114(11):1203–1214, 2010.

[8] M. Schwarz, H. Schulz, and S. Behnke. Rgb-d object recognition and pose estimation based on pre-trained convolutional neural network features. In IEEE International Conference on Robotics Automation, pages 1329– 1335, 2015.

[9] C. Wang, D. Xu, Y. Zhu, R Mart´ın-Mart´ın, C. Lu, L. Fei-Fei, and S. Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. 2019.

[10] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. 2017.

[11] A. Zeng, K. T. Yu, S. Song, D. Suo, and J. Xiao. Multiview self-supervised deep learning for 6d pose estimation in the amazon picking challenge. IEEE, 2017.

[12] Tielin Zhang, Yang Yang, Yi Zeng, and Yuxuan Zhao. Cognitive template-clustering improved linemod for efficient multi-object pose est

[13] S. Wang et al., ”3D Shape Perception from Monocular Vision, Touch, and Shape Priors