INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

LIRIS

INSA Lyon - Département Informatique

Report

of the

Graduation Project

The Deep learning methods on 6D Pose estimation [R&D]

Les méthodes d'apprentissage profond pour l'estimation de la pose 6D [R&D]

Yuxuan ZONG

Defended June 23, 2022

Project conducted from **February 28, 2022** to **August 19, 2020**

within the reception structure

IMAGINE-ECL LIRIS (Écully)

Referent	:	Jean Stefan DUFFNER, Professor	INSA Lyon
Tutor	:	Mitch Liming CHEN, Professor	ECL Lyon/IMAGINE-ECL LIRIS

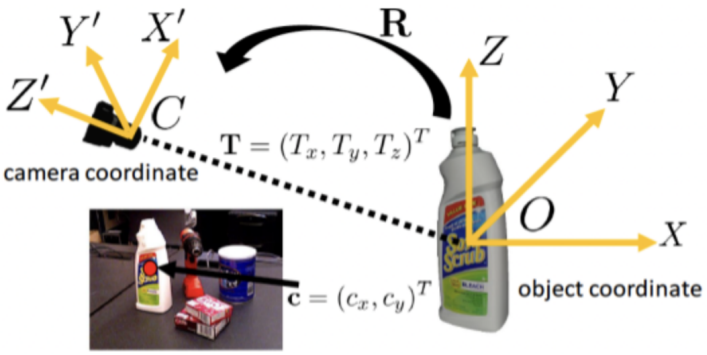
Contents

Annexes

1 Introduction.

1.1 Context.

6D pose estimation is an important problem under the study of the computer vision domain. It calculates the object's 3D translation and the 3D rotation based on the position of the camera. It is a technique which has been widely developed and used during the last ten years in the domains of robotics (robotic grasping (will be presented precisely later), robotic autonomous moving, etc) and augmented reality. [6] Further, the dataset of the 6D pose estimation has been greatly improved thank to the development of the detectors of the RGB-D images which allows to create and develop the dataset much more easier. Recently, there are more and more dataset which uses the synthetic data, which greatly increase the quantity of the frames in the dataset like NOCS and BIN-P. [7, 8]



Source of the graphic

https://arxiv.org/abs/1711.00199

visited March 19, 2022.

Figure 1: Illustration of the translation and rotation of 6D pose estimation

To accelerate the arrival of the "unmanned warehouse" era, Amazon has held annual public robot picking challenges since 2015, under the official name 'Amazon picking challenge'(APC). [9]During these years of the competitions, APC contains many different kinds of tasks, including the object manipulation, grasping, robotic route planning, etc. And the final goal of the challenge is to move the object through a robotic system to a target tote. In the first year, only half the team succeeded. As we need to manipulate the robots in an automatic way, accurate 2D object detection and 6D pose estimation will be necessary to improve the performance of this autonomous manipulation. 2D object detection mainly contributes to the location of the position of the object or the obstacle during the robot's movement in the view of the camera(detector) of the robot. The 6D pose will go further in the domain to detect the gesture of the object or the obstacle. It can resolve better in the situation that the object

has an abnormal surface shape. As a result of that, it can be better to navigate in the human environment by calculating grasping and avoidance strategies.

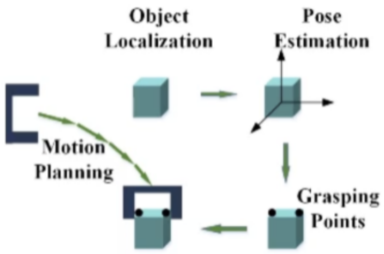


Figure 2: Example for using 6D pose estimation for robotic grasping

For the object grasping problem, we need to first label the object's own grasping point for the robot's claw in its own coordinate system. Having predicted the object's 6D pose in the real-world scene, the grasping point for the robot's claw can be calculated by using:

$$Pt_i = R \cdot Ps_i + T$$

where the Pt_i represent the i-th grasping point in the own coordinate system and the Ps_i is the position in the corresponding camera system, R and T represent the predicted rotation matrix and translation vector respectively. Then we just need to order the robot's arm to move to the corresponding position in the real-world coordinate system.

The 6D pose estimation is also widely used in the domain of augmented reality. For example, we can illustrate a coffee machine on an iPad by using the 6D pose estimation. The machine on the iPad is in the coordinate of the iPad's camera's system. When we press the button of the coffee machine on the iPad, the similar operation will be done on the real machine. The other AR games on the mobile phone like Pokemon Go also use the 6D pose estimation to predict the 3D bounding box of the real-world object in order to better illustrate them on the mobile phone.

1.2 Definition of our problem.

The problem of the 6D pose estimation can be divided into two levels: Instance-Level estimation and category-level estimation. For the instance-level pose estimation, the object we want to predict is known as we possess its CAD model. The known information includes not only which face is the forward(orientation of the object), but also the information like the grasping point or centre of the object. The input format of the dataset is usually RGB or RGB-D. With the model of the object, the depth of the input is not a necessary requirement for detecting all the degrees of freedom for all the 6D pose information(Known the intrinsic parameter and the size

of the object, which means the RGB dataset also works). One important work of this domain is PoseCNN [5], which can treat the texture-less, symmetric object with occlusion very well (Precise a little in the annex). The instance-level pose estimation works well for the industrial robotic manipulation, where the model of the product on the pipeline can be studied offline and the type of them are limited(small set of known instances), which makes the learning process efficient. However, the limited model means that at this level, the network cannot help to detect the pose for the other object, so that it will be hard to use it during the autonomous driving task (where the type of objects on the road is unlimited but it cannot generalise to novel objects).

The category-level pose estimation is a kind of 6D pose estimation which does not depend on the CAD model and can generalise to novel object instances. For example, category-level can identify all kinds of cups in the image without knowing the exact instance of one specific cup. For the definition of the centre or the orientation, as we don't have a CAD model, so for each object seen, we need to define them separately, which is impossible. As a result of that, we need to define them in a generalised way. For example, we consider the geometric centre as the centre of the object and the cup, the face which contains the handle as the front face, etc. The first method of 6D Pose estimation which use the idea of the category level is the work of He Wang et al. [7] NOCS, the short form for **N**ormalised **O**bject **C**oordinate **S**pace, can not only predict the 6D pose of the object, but also the size of the object.

Bin-picking is widely required in logistics and industrial assembly lines. For our situation, we need to **research a scene of picking different kinds of fruit and vegetables in a container**. This bin-picking problem will be fulfilled under the industrial production level. For different kinds of challenges, the different algorithms will have different effects. The original method proposed for the others challenges may not work for our cases.

For our case, we first need to identify whether our problem will be presented in the instance-level only, or we also need to make use of the category-level pose estimation. Then we need to identify the challenges needed to present in our dataset. Then we need to question whether the state-of-the-art-methods work well for our dataset. Finally, we need to find the source problem of the existing method and try to resolve it by proposing the new method.

1.3 Our contribution

We study in detail two methods which achieve the state-of-the-art method, G2L_net and DenseFusion and analyse its pros and cons.

We are going to propose a new dataset which combines the different kinds of challenges, especially for bin-picking and change of the viewpoint for both the category-level estimation and instance-level estimation.

Further, we are going to do some experiments on the proposed data by using the existing algorithms.

Finally, we are going to propose one or several new algorithms which work well for our own dataset.

1.4 Plan of the report

This report is structured as follows. In the next part, we will discuss several related works precisely in the domain of 6D pose estimation. These related works contain not only the algorithms for 6D pose estimation, but also for the dataset. Next we will do some analysis of the dataset and the methods by explaining why it is not good to solve our problems. After, we will take a brief view of our dataset for fruits and vegetables. Then I will talk about the future work to do after the redaction of this report, including the test to do and the possible methods to propose to solve the problems. Finally is the conclusion, this part contains not only a conclusion of this report, of this topic, but also contains my personal conclusion of this internship (PFE) although it still has more than two months to go. The reference together with the annex is attached at the end of this report.

2 Bibliographic study.

In this part we will discuss several related work in the domain of 6D pose of estimation. The algorithms are mainly being divided into 2 branches. The first branch is more concentrated on the accuracy of the algorithm, which corresponds to the domain of application of autonomous driving, etc. The other branch is more related to improving the efficiency, including the inference speed, of the algorithm, which is more used in the field of scene understanding of robotics in the industrial level. In this part, we will focus on two state-of-the-art methods: G2L_net and DenseFusion.

The training of the method will be useless if we don't talk about the dataset being used. In the latter part, we will also discuss several datasets which are mostly used for the 6D pose estimation and its properties like challenges involved, light changes, the type of the input data, etc.

For the 6D pose estimation using the Deep Neural Networks can also be split into two series, direct and indirect. For the direct one, the pipeline looks like below, The input data

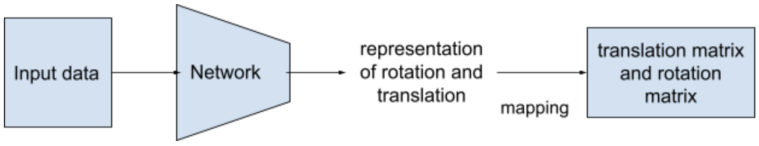


Figure 3: Workflow for direct 6D pose estimation

can be in the form of RGB-D, or even the point cloud, after feeding the data to the network, it will output the representation of the rotation and translation, in the form of the Euler angle or something equivalent. Then we can use the mapping algorithm to transform them to the translation matrix and rotation matrix. For the indirect one, given an image, for each pixel on the target object's surface, we can predict the 3D coordinate of this object on the object CAD

model. That is to say, we build a dense mapping from 2D value to a 3D value. This dense mapping contains the information of how to rotate and translate the object. It can be done by using the Perspective-n-Point algorithm(PnP) [10].

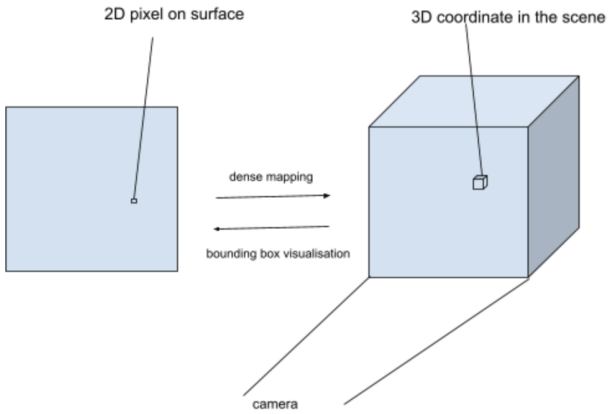


Figure 4: Indirect 6D pose estimation

2.1 G2L_net. [1]

G2L_net, the short form for Global to Local network. This network decomposes the 6D pose estimation into three sub-task, the Global localization, the Translation Localization and the Rotation Localization. The author tested the network on 2 benchmark dataset, LineMOD [4] and YCB-Video [5] and this method achieved the state-of-the-art for both of the two dataset. This method makes use of the idea of the PoseCNN [5], which does the 2D segmentation, translation and then a regression of the rotation. For the G2L_net, it does the segmentation in a 3D point cloud, then does the translation localization by using the method of PointNet [3]. Finally, this net does the rotation prediction by using a rotation residual estimator to estimate the residual between the residual and the ground truth, which makes the rotation prediction much more accurate. As I present this network to all the members of our group, I would also like to present some details of the network in this report. The pipeline of the G2L_net is illustrated below.

2.1.1 Global Localization

To locate the object's position in the whole scene, the G2L_net first proposes to train CNN to detect the object's bounding box in the 2D RGB image. In the paper, the author uses the YOLOv3 [11] to do this job. The output of this method is not only 4 numbers which predict the bounding box, but also a one-hot vector of size equal to the numbers of classes. This one-hot

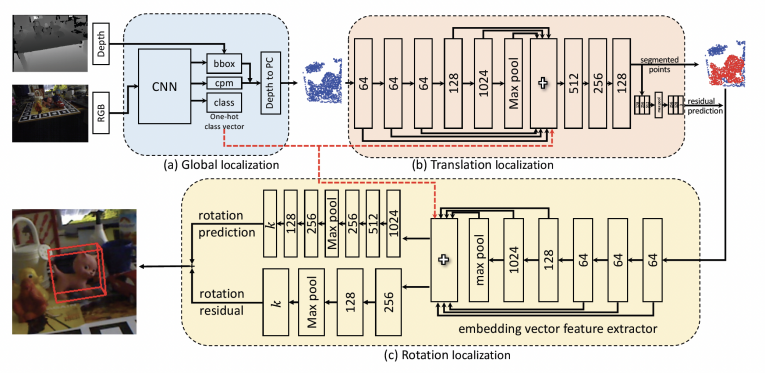


Figure 5: Pipeline for G2L_net

vector which predicts the label of the object will also be used later for the point cloud instance segmentation, translation and rotation estimation.

One important use of the 2D bounding box is to preselect the area which contains the object. This step can largely reduce the number of the points in the point cloud area feed to the next step. The global point cloud for the scene is generated by first using the RGB information and then adding the depth information for each pixel to extend it. For the early version of the PointNet [12], it uses the 2D bounding box and prolongs into the depth of the point cloud to get all the points in the point cloud which are potentially a part of the object. I drew a picture below to illustrate this process.

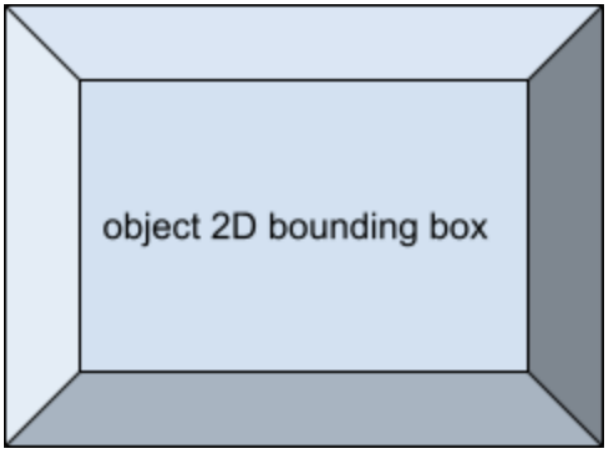


Figure 6: Frustum area for selecting the potential points

The surface towards us is the detected bounding box, according to the intrinsic parameter for the camera, the detected object must be in the space of shape frustum behind the box. If the camera is far enough from the scene, the frustum becomes a rectangle, which means we

reduce the 3D search space of the object into 2 axes (x,y) and all the points through the z axes will always be considered a potential part of the object. As a result of that, some points with a huge depth different from the object will also be considered as a possible solution. From my point of view, the idea is pretty good but we can make it better. For G2L_net, it proposes to use a 3D sphere to further reduce the 3D search space for the point cloud in a more compact space along the axis z. For the centre of the 3D sphere, we need to predict it by using the result of the 2D detection, which produces also a class probability map, which contains the probability that each pixel belongs to the object. We choose the pixel(x,y position in 2D space) with the largest probability and combine it with the information of its depth to get the centre of the 3D sphere. For the radius of the 3D sphere, we take the radius or the size of the object in the CAD model to predict. As in the picture from [1], only the points in the red sphere will be considered later, compared to [12], the blue points in the upper will be ignored.

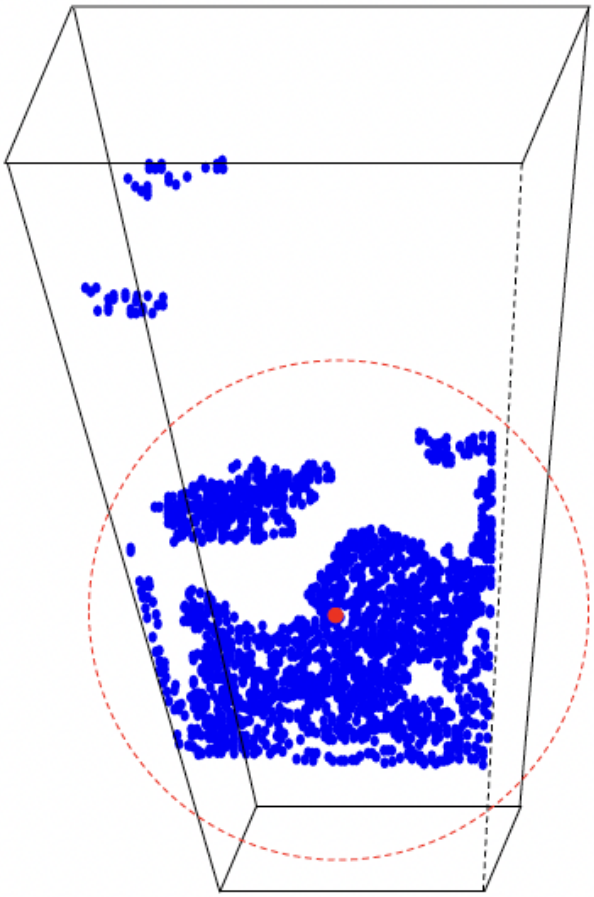


Figure 7: 3D sphere for global localization

2.1.2 Translation Localization(PointNet [3])

Although we have done the global localization of the data, it still has some issues. First the points in the 3D sphere contain not only the points belonging to the object, but also the non-object points. Second, as the translation vector is unknown, we cannot transform the points to its own coordinate system to do the rotation estimation. To cope with the issues, we need to do a 3D segmentation of the points to identify the residual distance of the ground truth of the translation and the predicted translation. The predicted translation could be considered as the mean value for all the points belonging to the object. In the pipeline for the translation localization, the upper line does for the 3D segmentation part, and the lower part makes use of the segmented points and does a prediction of the residual translation (more precisely, they are doing a prediction of the centre of the object with respect to the camera coordinate system).

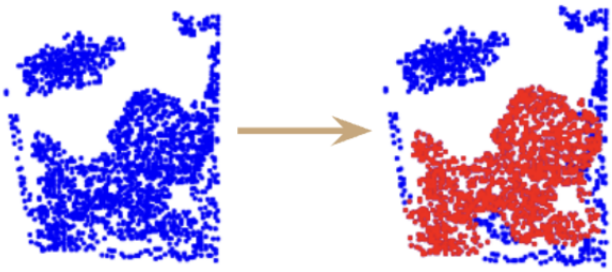


Figure 8: Point cloud segmentation

The network which has to be employed to do the segmentation is the PointNet [3]. It is a network design to cope with the issues of the object classification and object segmentation task where the inputs are the point cloud, which make the problem different to the other computer vision task. First, the points are unordered. From one side, unlike the pixels in the image are neatly ranged in row or in column, the points in the point net have only its own explicit coordinate to represent its location information. As a result of that, we cannot use the CNN to extract the implicit information together with the nearby points. From the other side, as it is unordered, the direction of the input should have no influence on the result, which means its results should be invariant to the $N!$ permutation of the input order. Second, although there is no implicit location relation between the points, the relative distance between each point is still meaningful. The interactions between the points cannot be ignored, which means we cannot treat each point separately. Third, the point cloud should be invariant with respect to the transformations. For example, rotation and translation will not modify the classification or the segmentation result.

Regarding these problems, the Point net proposed to use a symmetric function to deal with problems of the invariant for different order of input. The proposed one in the paper is max-pooling. Then for the classification and segmentation, we need to combine the local features (the features for each point) and the global features (the feature after doing a max-pooling on all

the points, which contains the information depends on all the data and identical to each other with respect to different points). But according to the pipeline, these two parts are separate. To cope with this constraint, the paper [3] proposed to feed it back to per point features by concatenating the global feature with each of the point features that after computing the global point cloud feature vector. Finally, as the result needs to be invariant to the different kinds of rigid transformation. It will predict the transformation matrix by using a T-net and then apply this network on the input points. For the first T-net(input transform block below), it is a network to predict the transformation of the input coordinate. For the second T-net(feature transform block below), it is used to predict the feature transformation matrix to align features from different input point clouds. For the second one, we will also add a regularisation term to help to optimise the problem because the dimension of the matrix is increasing much higher as the number of features increases. For the T-Net, its structure is similar to PointNet [3] itself with a little modification.

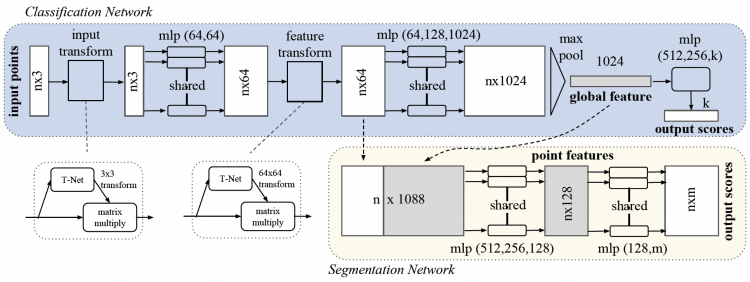


Figure 9: Pipeline for PointNet

2.1.3 Rotation Localization

Rotation localization is the most interesting part for G2L_net [1]. To cope with the issue of multiple viewpoints, the PointNet [3] proposes to extract global features from the whole point clouds, but these global features are highly correlated under similar viewpoints. As a result of that, the G2L_net [1]proposes to use the point-wise embedding vector feature(EVF). EVF is done on local canonical space, which means that we move the object to the centre(minus the translation predicted in the last part.) The point-wise embedding vector which will be predicted during the rotation localization part. The point-wise vector will be trained to make it toward the keypoints. The keypoints are some predefined 3D coordinates based on each object's CAD model. There are two principal ways to locate the keypoints. The one used by the G2L_net [1] is just using the 8 corners of the bounding boxes of the object. This method is also widely used in the 2D object detection task. One other possible solution is to use the FPS(farthest point sampling) algorithm, which has been proposed in the work of Sida Peng et al. [13]. The advantage of the first one is simplicity. We can directly get the keypoints in the model without further calculation. However, the number of the keypoints in this method will

be fixed to 8, which means we cannot change the number of the keypoints to study the best choice. According to the paper of the G2L_net [1], they chose to use the corners because of the simplicity and the result by using the FPS algorithm does not have too much difference with the corners. The following figure shows the point-wise vector pointing at one keypoint marked in green for each point of the object marked in red. The other keypoints are in black. All the vectors are predicted by the network of the rotation localization.

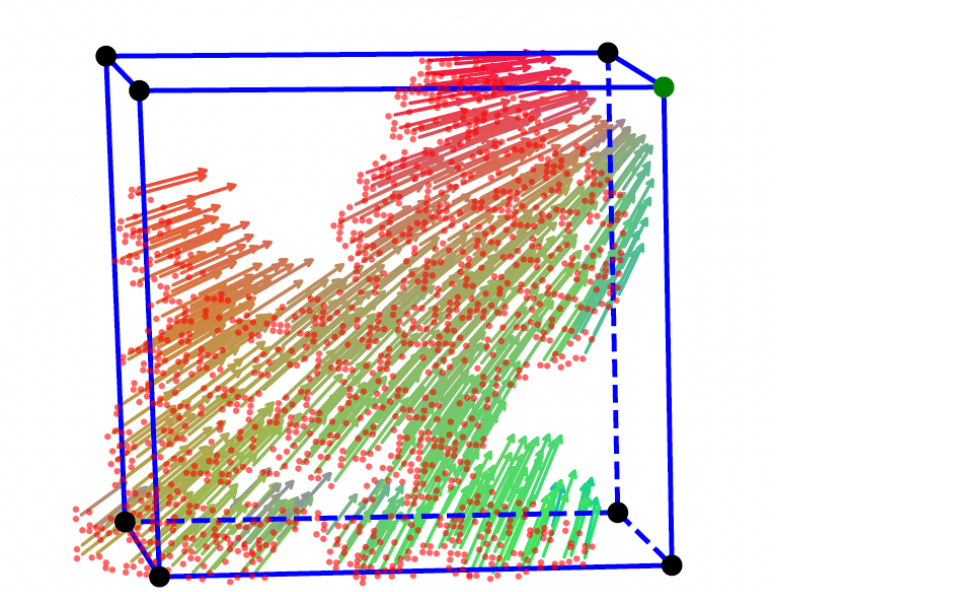


Figure 10: Point-wise Vectors(EVF)

After getting the point-wise embedding vectors, the paper [1] has built a network to take them as the input and output the rotation matrix. Actually, this network will just predict the position of the keypoints and then, use the Kasbach algorithm (in annex) to transform the keypoints position to the rotation matrix. During the inference process, it is necessary because we need a rotation and translation matrix to draw the bounding box on the picture, but in the training period, we can ignore this process because we can make the label also in the format of the keypoints. Ignoring this step can also save a lot of training time because the Kasbach algorithm is mainly done under the large matrix multiplication and singular value decomposition.

The last sub-network for the rotation localization is a refinement process. It is used to estimate the residual between the estimated rotation and the ground truth. As the author's experiment shows, when the rotation network cannot fully exploit the rotation vector features, the rotation residual estimator can have a huge impact on the result. As the estimate rotation predicted in the last part is P' and the ground truth of the rotation is P, the ground truth of the rotation residual estimator is $\|P' - P\|_2$.

The following figure shows in precise the rotation localization part. In the full pipeline figure, it just shows a part, but it shows that the . The block A is used to predict the unit vector for each point pointing to the keypoints. The loss is directly the MSE(Mean Square Error) between the ground truth directional vector and the predicted one. Block B uses the predicted point-wise embedding vector features from block A to predict the rotation. The loss for this block is the distance between the ground truth rotation and predicted rotation. Block C plays the role of the residual rotation estimation.

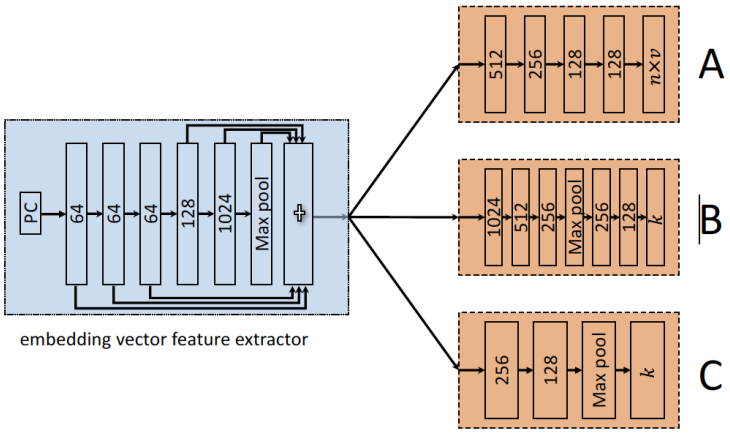


Figure 11: Architecture of rotation network

2.1.4 My experiment

For the test of the G2L_net [1] on my computer, due to the computational capability, I just use 70% percent of all the training data. Here below is one inference step in the testing level. The code for visualising the bounding box in 2D and 3D is written by me. For the following figure, the one on the left illustrates 2D object detection by using the yolov3 [11]. The figure on the right contains the ground truth 3D pose and the predicted one. The ground truth is in white and the predicted one is in blue. We can see that the result is generally good. But it still has a large improvement space.

One important remark during my experiment of the prepossessing of the dataset. For lineMOD, it does not contain the labels for the point cloud segmentation. It means that we need to generate them. We can use the point cloud from the CAD model and apply the ground truth translation and rotation matrix with respect to the camera's parameter. Then we can establish a threshold between these ground truth points and the point in the scene to identify whether they are a part of the object or not.

Listing 1: data prepossessing

Figure 12: Result of the 6D pose estimation done by G2L_net

```
def generate_positive_label(cadpoints,scenepoints, R, T,
DISTANCE_THRESHOLD = 0.1):
    applied_cadpoints = (R @ cadpoints.T).T + T
    expandedcad = applied_cadpoints.repeat(1000,1)
    expandedscene = scenepoints.repeat(1,50).reshape(-1,3)
    less_than_threshold = torch.where(torch.norm(expandedcad
    - expandedscene, dim = 1) < DISTANCE_THRESHOLD)[0]
    labels_positive = torch.unique(less_than_threshold/50).shape
    return labels_positive
```

Here is the example code to do this procedure, the variable cadpoints refers to the points generated by the CAD models. Similarly, the scenepoints refers to points in the points cloud, R and T refers to the ground truth matrix and the DISTANCE_THRESHOLD refers to the threshold to identify whether the point belongs to the object in the point cloud. Finally the output of this algorithm is a list of index of the scenepoints which belongs to the object.

2.2 DenseFusion [2]

As described in the introduction, the instance-level pose estimation can be resolved well with RGB information, but when encountering the challenge with heavy occlusion and light changing. DenseFusion is a network which can efficiently use the information of the depth together with the colour (RGB) information. It proposes to use two sub-networks to extract the colour features and the geometric(depth) features respectively. Then a network to fuse the two kinds of features to get the per-pixel dense features and the global feature. With them, the author uses a similar approach like the PointNet [3]. Finally, the author uses an iterative refinement network to refine the result, which is much faster than the common ICP method(Iterative Closest Point, precise a little in annex).

First of all, the network does a 2D segmentation on the RGB image. The architecture used for it is the well-known encoder-decoder. The output will be in the same size of the image but with $N + 1$ channels where N is the number of classes. The value in each channel corresponds to the probability that each point belongs to the class. After that, we can extract only the

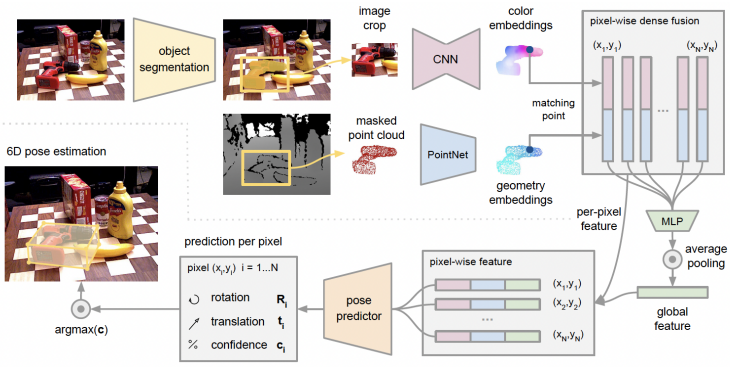


Figure 13: Workflow for DenseFusion except the refinement part

pixel of interest to do the further study. For getting the colour embedding, we just use the pixels and run a classical CNN on that. The network can create a feature of size d_rgb for each pixel. For the geometry feature, we expand the RGB information for each pixel extracted by the 2D segmentation with the depth information. We can then get the 3D point cloud in that area. As a result of that, we can run a PointNet [3] to extract the geometric features of size d_geo for each pixel. The only difference from the original PointNet [3] is they use an average pooling instead of max-pooling for a better performance. After this step, we can do a pixel-wise dense fusion which operates on each matching pixel and segmented points. Then from these dense features, we can extract a global feature for the whole scene by using a MLP(Multi Layer Perception). As in the paper's experiment [3], if we use only the global features, the result will be bad because of the heavy occlusion, the segmented points will also be related to some other objects or even the background. The paper chooses to add the global features back to each pixel-wise dense feature. In conclusion, each dense feature will be composed of three parts: colour feature in rose in the figure, geometric features in blue and the global features in green which is the same for all the pixels. Then the final prediction is done by using a pose estimator which outputs a rotation matrix, a translation matrix and a confidence score for each pixel-wise dense feature. Finally we choose the pose estimated with the best confidence score.

The loss function of this network is also very interesting. For the asymmetric object, the loss function for each pixel-wise dense feature defines as the distance between the points sampled on the object model applying ground truth pose and the corresponding points on the same model transformed by the predicted pose:

$$L_i^p = \frac{1}{M} \sum_j \|(Rx_j + T) - (R_i x_j + T_i)\|_2$$

Where the L_i^p represent the loss from the pose generate by the ith dense-pixel, M represent the numbers sampled from the given CAD model. R and T represent the ground truth pose and the R_i and T_i represent the pose generated by the ith dense-pixel. j is the index of the sampled points from the CAD model.

This method does not work well for the symmetric object(like a cup without a handle). For them, it can have an infinite number of pose estimations as it is difficult to define the corresponding points. It will also make the loss defined before ambiguous. As a result of that, for a symmetric object, we minimise the points on the point cloud transformed by the estimated pose and its closest point of the ground truth one. The loss function for each dense-pixel now looks like:

$$L_i^p = \frac{1}{M} \sum_j \min_{0<k<M} \|(Rx_j + T) - (R_i x_k + T_i)\|_2$$

In general, the loss of all the pixel-dense should be the sum of each one, like $L = \frac{1}{N} \sum_i^N L_i^p$, but this loss does not contain the confidence score where we cannot identify which pose estimation generated is better. In order to solve the problem, the new sum of the loss looks like below:

$$L = \frac{1}{N} \sum_i^N (L_i^p c_i - \lambda log(c_i))$$

Where the c_i is the confidence score of the ith dense-pixel and $log(c_i)$ is a regularisation term. If we make the to 0 (without the regularisation), it will learn to make every c_i to be small to satisfy the condition. However it is meaningless. With the regularisation term, if the c_i is small, it can make a small weight for the corresponding loss but a higher regularisation loss. In contract, if c_i is big, which means the L_i^p plays an important role and needs to be considered.

For the iterative refinement, it is almost the same as the normal network. It uses the predicted pose from the previous iteration and applies it to the point cloud. During each iteration, we don't modify the colour features because the new rotation and translation matrix will only have influence on the point cloud. After M iteration, the final result of the estimated pose is the product of the M matrix.

2.2.1 My experiment

The evaluation metric used for DenseFusion is ADD-S(average closest point distance). ADD-S calculates the mean distance from each 3D model point transformed by the predicted pose to its closest neighbour on the target model transformed by the ground truth pose. In the test, we use the 2 centimetre as a reference score(for one frame, it will output an average distance and we compare the distance with 2 cm). For the accuracy of an object, we consider the number of frames which has the ADD-S $<$ 2cm over all the number of frames. By using the pretrained model, we have a result as below. We get promising accuracy in the paper.

After that, I tried to train the DenseFusion on the LineMOD [4] on the computer of the lab with the GPU NVIDIA QUADRO K2200, after modifying(decreasing) the learning rate, the result converged much faster. In the pretrained model, it stops at the 493 epoch with a test distance of 0.0067, but for my test, at the 249 epoch, the test distance is already at a test distance of 0.0052. The experiment shows first that the dataset LineMOD [4] is not that complicated for a network like DenseFusion. Further, during my training, only the first 16

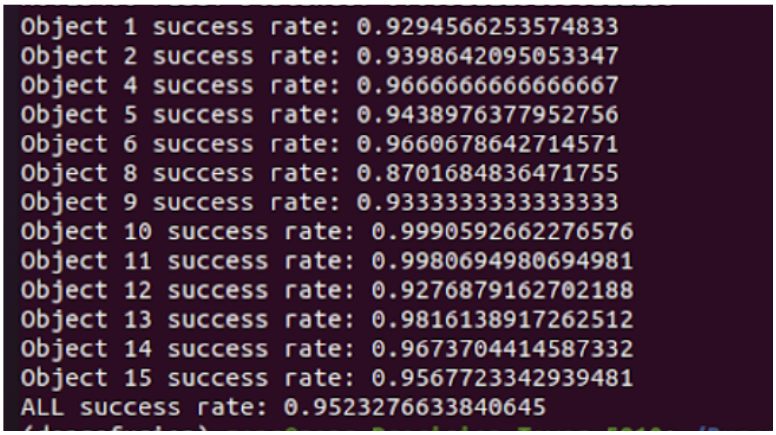


Figure 14: Inference stage by using the pre-trained model

epochs is the training for the normal network and it converges. For the rest of the epoch, they are all training for the refinement part. For the last epoch for the normal training, it can have a test distance about 0.009, which is not enough to have a high accuracy. It shows that although the refinement training is slow, it is very important.

The table1 shows the comparison of the pretrained models and our trained model. We have modify the learning rate and of the network for different objects in order to make the network more suitable according to different object. From the table, we can see that almost every object

Accuracy on ADD-S	Pretrained [2]	Ours
Object 1(ape)	92.3%	95.1%
Object 2(bench)	93.2%	96.1%
Object 4(camera)	94.4%	97.5%
Object 5(can)	93.1%	98.0%
Object 6(cat)	96.5%	97.7%
Object 8(driller)	87.0%	94.8%
Object 9(duck)	92.3%	96.7%
Object 10(**egg-box**)	99.8%	99.9%
Object 11(**glue**)	100.0%	99.5%
Object 12(hole)	92.1%	95.5%
Object 13(iron)	97.0%	99.4%
Object 14(lamp)	95.3%	98.2%
Object 15(phone)	92.8%	97.2%
Overall	94.3%	97.4%

Table 1: Accuracy score for different objects in LineMOD. Objects in **bold** are symmetric

has a great improvement of the accuracy. Further, the training time is also largely affected.

For the pretrained data, it reaches the test error of 0.0113 at 15 epoch but for the ours, we need only 6 epochs to achieve that state. For the refinement part, the difference becomes much larger. The refinement error for the pretrained model is about 0.0067 and it use 493 epochs to achieve such result. However, in our experiment, it needs only 100 epochs to achieve such result, which means generally 4.5 times faster. If we continue to train our model to 450 or 500 epochs, we can achieve a error of about 0.0056, which has a much better result, as shown in the table before.

In the future work, we would also like to modify the other parameters to see whether we can further improve the result.

2.3 Dataset

Before I go into details of different types of proposed dataset, I would like first to talk about the challenges in different kinds of the dataset. If you put a conspicuous object in front of a green curtain, even the simplest architecture(network) can do the 6D pose estimation correctly. In reality, the 6D pose estimation has different kinds of challenges for different kinds of tasks. For example, if we want to use it in the area of autonomous driving, the light changing is an important challenge which we must consider. If not, the car will lose direction in the night and it can cause safety problems.

For the instance-level pose estimation, it has often the challenges listed below:

Variation of Viewpoint: The short form for it is the VP. In the testing scene, the location of the target object in it is various. All of the space in the test scene in the real world could be a possible position for the target object. As the space gets wider, we need more data to train the 6D pose estimator, in order to get the best viewpoint which can capture all the information for detecting the 6D pose of the object.

Texture-less Objects: Texture is an important information for RGB cameras, which can capture and represent a scene by 3 basic colours: red, green and blue. An object of interest can easily be distinguished from background or any other instances available in the scene, in case it is sufficiently textured. This is mainly because that texture on the surface allows to define discriminative features to represent the interested object. However, when objects are textureless, this discriminative property disappears, thus making methods strongly dependent on depth channels in order to estimate 6D poses of objects. [6] As the light changes with the different depths of the object in the RGB image, the 3 basic colours reflected will be modified. This means that only making use of RGB can work for instance-level pose estimation(as described in introduction), but it works badly if the dataset has the texture-less challenge.

Occlusion: The occlusion is one of the most interesting and the most common challenges for 6D pose estimation problems. When the target we study is partly or even completely being blocked by other objects or obstacles, the occlusion problems arise. We can solve it by using a guesser to try to understand the situation with just the part of the object which has not been blocked. But doing it in the normal way will decrease the precision and the recall(algorithms which give up to predict the occluded object.)

Clutter: Cluttering is a challenge related to the image with a very complex background. It is associated with applications like autonomous driving. Some algorithms use the background image to train the network to make the interested objects more evident in front of the complicated background. [14] But it will make the detection depend a lot on the background and could be hard to have a generalised one.

Similar-Looking Distractor: The similar looking distractor is one another important challenge for instance-level pose estimation. The distractor could be an object we also need to research(apples and tomatoes are two classes of objects and they appear in the same scene.) or the distractor of no interest. Without the similarity in colour(bananas and sausages), the RGB channels can be very helpful to resolve it. However, for the algorithms which choose to ignore the colour features and concentrate more on the shape features for the detection [15], the result could be very bad.

For the category-level pose estimation, the most famous challenge is the Intra-class variation:

Intra-class variation: Through the defintion of the class, although the instances defined under the same category should have the similar property, the instances are always not the same. The colour and the scaling(the ratio stays the same) differences can be solved easily by proposing the algorithms[15] depend more on their shape(less depends on the size and colour of the object). But the changes in the shape could not be easily identified. Further, if we consider one instance as the centre of the class and all the instances which have a smaller 'distance' to this object belong to this class, the 'distance' could be difficult to be defined in a literal way.

2.3.1 LINEMOD [4]

The linemod is a dataset which has been published in 2012. It is a dataset for instance-level pose estimation only. The labels contain the 6D pose estimation of the object and the corresponding depth. All the labels are annotated in a manual way and this dataset contains 15 objects. Each object has a video with in total 18,327 frames. For each object, it has about 1,100 1,300 frames. One important disadvantage of the Linemod is that the train data and the test data for one object is from the same video. This means that the illumination condition and the appearance of the objects are very similar in the training and testing data, which can't test the robust of the method. It has the challenges like the viewpoint change, the cluttering and the texture-less object. The sample image could be found at figure 12. One major problem of the dataset is that it has only one object annotated in each frame, even if all the objects are presented. And it doesn't include the important challenge like the occlusion. To resolve such problems, the Occluded Linemod [4] appears.

2.3.2 YCB-Video [5]

The YCB-Video Dataset Xiang et al. [5] features 21 YCB objects of varying shape and texture. The dataset contains 92 RGB-D videos, where each video shows a subset of the 21 objects in different indoor scenes (each frame will annotate several objects). The videos are annotated

with 6D poses and segmentation masks. Unlike the Linemod, it use the 80 videos' frames to generate the training data and the testing data are from the rest 12 videos, which helps to resolve the problem I described in the Linemod. Besides the normal occlusion challenge, the YCB-Video also adds some noise pictures for the data augmentation, which make the training more robust. Although the dataset is much bigger than the LINEMOD [4], it has the problem of the limited pose variability and data redundancy because the scenes are still limited. The figure below shows a sample image of the YCB-Video. We can obeserve clearly the challenge of occlusion.

Figure 15: Sample image for YCB-Video

2.3.3 Bin-P

The Bin-P [8], short form for 'Fraunhofer IPA Bin-Picking Dataset', is a dataset designed especially for bin-picking algorithms. It includes 520 fully annotated real-world point cloud and correspondence depth images and more than 200,000 synthetic scenes. The synthetic images are borrowed from the works of R. Bregier et al. [16] with some further treatment. The other 2 objects are the real one. For the synthetic one, they drop the CAD model from a distance. For the first time, they drop one object, then record the data and clear the scene. For the second iteration, they drop two objects, and do the same thing. The iteration ends when a predefined number of iteration N is reached. For the real world data, the procedure has a little difference. It looks a little bit like the famous game Mikado, they first fill the bin with N objects and then remove them one by one without modifying the pose of the other objects. They record each remove and finally reverse the scene. The sampled data looks like below:

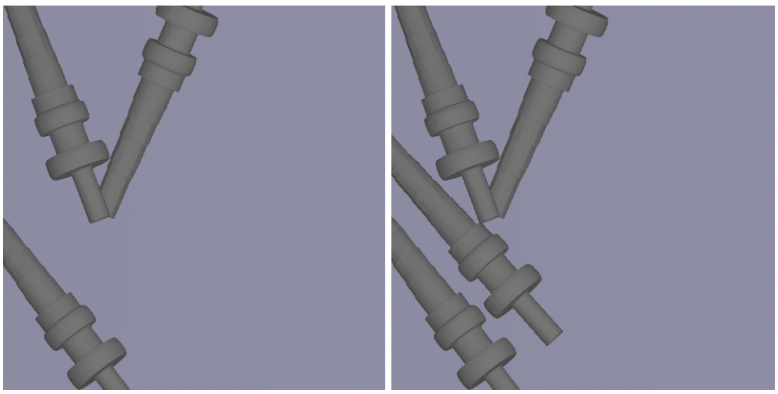


Figure 16: Sample image for Bin-P

The table below conclude the basic parameters of the three datasets. In the column Creation, the R represent the image is the real one and the S represent the data is synthetic.

Dataset	Level	Observation	Creation	Objects	Challenges	Type
LineMOD	Instance	18,273 frames from 15 videos	R	15	VP + C + TL	Household
YCB-Video	Instance	133,827 frames from 92 videos	R	21	O	Household
Bin-P	Instance	520 Real and 206,000 synthetic image	R+S	8+2	VP + SC + SO + MI + BP	Industrial

Table 2: Baseline dataset conclusion.

3 Analysis of the methods

3.1 Comparison of the methods

The G2L_net and the DenseFusion are both the state-of-the-art method for 6D pose estimation based on instance-level. The table below compare the result of the two methods on dataset Linemod and YCB-Video:

From the result side, we can see that the G2L_net can achieve 23 Frame per second for the inference speed and 98.7% accuracy score on the Linemod [4] dataset where the accuracy score is based on the ADD metrics. However the Dense fusion can only achieve the 16 Frame per Second for the inference speed and 94.3% accuracy, which means that it cannot reach the level of G2L_net [1] for both speed and accuracy. Without the last refinement step of the refinement,

Accuracy on ADD-S	Linemod [4]	YCB-Video [5]
G2L_net [1]	98.7%	92.4%
DenseFusion [2]	94.3%	92.8%
DenseFusion [2] without refinement	86.2%	91.2%

Table 3: overall accuracy score for DenseFusion and G2L_net .

the result can be even worse, just an accuracy of 86.2%. If we go into detail of each class, we found that in the Linemod [4] dataset, the two symmetric objects' accuracy score (egg box and glue) is much higher than the other asymmetric objects(>99.5%). I used to think that it should be a strong point in the architecture design for the symmetric object. But since I found that for densefusion, it can still get a very high accuracy without the refinement(99.9% and 99.4% for each, for the other much earlier algorithms, it is also higher). I think the calculation of the symetrique object should be separate from the other accuracy score. For the asymetrique object, the correspondence of each point is fixed. One point in the ground truth point cloud is corresponding to one point in the predicted point cloud. The accuracy will not only depend on the true accuracy, but will also depend on the choice of the corresponding points. This one-one mapping may make some corresponding points far from each other. However, for the symetrique objects, we cannot ask the evaluation system to find the exact corresponding points. For example, for one symetrique object in Figure 16, it has a rotation axis y, where all the points in the point cloud are symmetric with respect to that axis. For one point in the CAD model, we can easily identify its coordinate on the axis x and axis z. But when the object is predicted in the camera coordinate, we cannot identify the coordinate of the corresponding point. We can only predict the coordinate on the y axis and how far it is from the y axis. So in the final rotation prediction part, there is one axis of rotation that is useless and when we consider the accuracy, we will only consider the nearest point to overcome this kind of ambiguity.

So in my point of view, it will be interesting to separate the accuracy of the symetrique and asymetrique objects. For the G2L_net, the accuracy for the asymetrique objects is about 98.4% on average and for the DenseFusion with refinement, it has only the accuracy of 93.2%, where the difference between DenseFusion and G2L_net is larger. For the DenseFusion without refinement, the overall accuracy is only about 83.7% where a lot of objects have a result below 80%. For this comparison, we can conclude two things. First, the procedure of the refinement is important for DenseFusion. 10% of accuracy is not ignorable although the training of the refinement model takes much more time than the normal network as explained before. Second, the G2L_net is well-designed to resolve the challenges like the texture-less object, cluttering, etc. But the DenseFusion encountered some problems when it met the dataset like it. This is verified by my experiment, although the overall accuracy is a little bit higher(95.2%), the main principle remains the same. We need to note that, during our experiment,both the Densefusion and the G2L_net worked pretty well for the object iron(accuracy > 98%). In my opinion, the main reason comes from the dataset. The dataset owns more samples than the other ones and

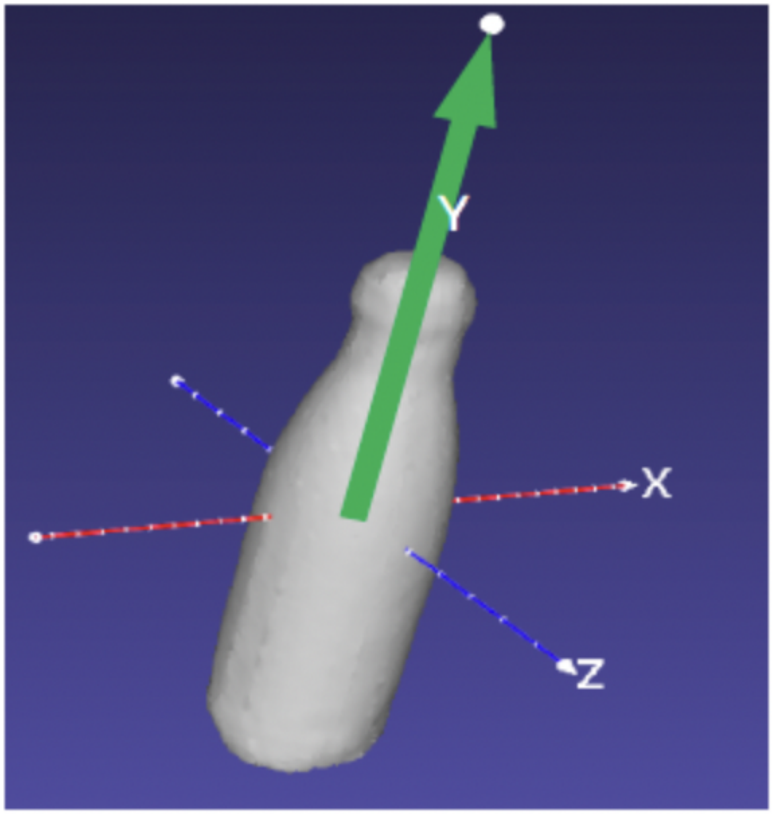


Figure 17: 3D view of a symetrique object

the object is placed in the centre of the table where the background is much simpler. For the YCB-Video [5], the situation is completely reversed. For G2L_net, it has just about 92.4% of accuracy on this dataset. The inference speed is still about 21 Frame per second. On the other algorithm YCB-Video, the accuracy achieved 96.8% and with an inference speed of 16 fps. Although it is slower than the G2L_net, it is enough for the real-time application like the grasping. Even for the DenseFusion without the refinement, it can also achieve an accuracy of 91.2%, which is comparable to the result of G2L_net. For some special objects in the YCB-Video dataset, which are texture-less and symetrique, like bowls, the accuracy difference before and after the refinement is large(10%). This shows that the refinement can help to resolve the problems of texture-less objects. In conclusion, the test of the YCB-Video dataset shows that the DenseFusion can treat the challenge like the strong occlusion. Further, for the Linemod dataset, there is only one object annotated in each frame but in YCB-Video, there are several. DenseFusion can also resolve such problems very well.

3.2 Advantage and disadvantage for G2L_net [1]

The use of the 3D sphere to do a global localization of the object is a good idea in my opinion. It can largely decrease the number of points being fed to do the 3D point segmentation. Compared to the DenseFusion, which needs to do a 2D segmentation to get all the pixels belonging to the object. G2L_net provides the 2D detection to get the bounding box, which is much faster than the 2D segmentation. Further, compared to the denseFusion which uses only the 2D segmentation and then directly uses the CNN or PointNet [3] to do the feature extraction, the information may not be very correct. When we encountered a situation where the occlusion and light changing is very frequent. The extracted pixels from the 2D segmentation may not be very precise. It doesn't use the information of the depth at this step. However, for the G2L_net, it uses the YOLOv3 [11] and the 3D sphere mechanism to pre-selected the 3D points who may belong to the object. Then it uses a PointNet to do an accurate point segmentation. It makes full use of the depth information, which can make the segmentation result much more accurate. It seems that this method uses more time in the segmentation part compared to the DenseFusion. However, during the PointNet [3] which does the segmentation, it has already extracted the features and they can be used later in the rotation prediction step. The experimental result shows that for the segmentation part, it takes almost the same time as the 2D object detection part.

Although it is a good idea, it still has several problems(disadvantages in this step). For example, for the diameter of the sphere, the paper uses the diameter predefined in the CAD model. In this way, the pre-selected points will be as few as possible but if the 2D detection is not accurate enough(or perhaps the centre of the 3D sphere is not accurate enough), it will eliminate several points who belong to the object. As a result of that, the accuracy of the rotation and translation prediction. Further, these points which we have wrongly eliminated may belong to the points on the border of the instance, which means that it will have a close relation with the keypoints of the object. So it will decrease the accuracy more than expected and it will not be robust for the outliers. One other problem comes from this fixed size pre-selected step, it is only suitable for the instance-level pose estimation. When we need to solve the category-level pose estimation, it could not work as the intra-class variation of the size is totally undefined. During the category-level pose estimation, we don't have the predefined CAD models.

One important disadvantage of the G2L_net is the lack of treatment for the colour information. For the 6D pose estimation task, what we want to know is the geometric information of each scene, but we cannot ignore the information we can get from the RGB. For the objects with the texture, the information we get from the RGB can be quite useful. The colour information not only gives us information about the border of the 2D object, but also some important information about the object itself, because in the instance-level pose estimation, the colour on the surface of the original object will not change. Although the light of the scene can make some slight differences, the colour still plays an important role. However, in the architecture of the G2L_net, the use of the colour information only appears in the 2D object

detection and 3D segmentation part. It is common but for the rotation detection part, the colour information could still be useful. But in the pipeline, the estimation of the EVF or even the rotation will largely be affected by the relative location of the keypoints and ignore the effect of the colour(like in figure 9, the shape plays an important role here). Compared to the DenseFusion, which extracts the colour feature with a separate CNN network, it has the same status with the geometric features and the information is used for every sub-network. The effect can be interpreted easily with the result. With less usage of the colour information, the G2L_net cannot solve the noise challenges involved in the YCB-Video very well.

The last interesting point of the G2L_net is the use of EVF. Compared to other methods which directly regress the global point features to get the pose estimation, the usage of the EVF can exploit the viewpoint information for the dataset like Linemod. For linemod, one object owns over 1000 frames with only one scene, which means it is a dataset with lots of viewpoint information. If we can make full use of this information, the network will be robust to the viewpoint changes in the inference phase. EVF is the vector pointing to the keypoints. When the pose changes, the position of keypoints with respect to the camera's coordinate will change, but from the point of view of each point belonging to the object, the relative position of each keypoint(vector pointing to the keypoint) will not change. As a result, the EVF can make the training result more stable. One another point is the residual of the rotation. For the refinement part of the denseFusion, it is done after the main network, which is slow in the reference phase. But in the G2L_net, the residual of the rotation is done in a parallel way, which means that it can save a lot of time.

In conclusion, the G2L_net does well in locating the object and calculating the rotation, but it has some problems in how to treat the colour feature.

3.3 Advantage and disadvantage for Densefusion [2]

The keypoint of the Densefusion is to use a CNN and a PointNet [3] to extract the colour features and the geometric features separately. This is a pretty good idea which can make each pixel contain the information of RGB and geometric at the same time. The colour information can contribute a lot during the process of getting the pose, but I doubt that the proportion of the colour feature is too large with respect to the geometric information. As a challenge of the 6D pose estimation problem, the main target is to study the gesture of the object in a 3D space. In my opinion, the shape of the object should be put in a more important position. Before feeding the pose predictor, the information concerning the shape is limited. A shape-based DenseFusion should have a better result. However, in DenseFusion, the colour features and the geometric features are concatenated directly without further treatment. The colour features which have the same role with the geometric features are a negative point of the network. For this kind of problem, we can try to fusion the two features in a different way. For example, we can first do a PointNet [3] to do the shape feature extraction as before. Then we add directly to original RGB information to the prepossessed shape feature for further information extraction. The negative point of this is that it will become harder to be done in parallelization structure.

One other important problem of the colour and shape information fused are two much isolated. Both the CNN and the PointNet [3] are deep net but in the DenseFusion, all the fusion work is done only at the final step, which ignores a lot of interesting information during the two-subnet. If the DenseFusion can add some fusion mechanism inside the CNN and the PointNet [3], the result could be better. For example, we can do an exchange of information for every ten layers. To make the geometric feature at that time already owns the information of the colour and the reverse either.

The other important problem is the 6D pose estimated by the DenseFusion is regressed directly. From my point of view, it is the main source of the low accuracy of the DenseFusion and if we want to get a better result, it needs a refinement part, which costs more time. In comparison, the G2L_net does well in this part(As I have explained in detail of the comparaison in the part of the Advantage and disadvantage for G2L_net, I will not do it again in this part).

On the other hand, the use of all the pixel-wise dense features seem to be an advantage of the network. For each feature, the predictor could output not only the rotation and translation matrix, but also a confidence for this pose.This is similar to the procedure done in YOLO [11]. For YOLO [11], it will divide the 2D image into several different grids. For each grid, it will generate several different possible anchor bounding boxes. During the prediction part, it will output the bounding box for each anchor box together with its confidence score. For the DenseFusion, it is almost the same procedure. The 2D segmentation generates and pre-selected the pixels which are possible to generate the bounding box. Then we can use this kind of information to generate the pixel dense feature and then predict the pose with confidence score. This can make sure that the pose can be generated by all the points related to the object and the usage of the confidence score can easily remove the pose with small probability. This procedure can be easily parallelized so it will not cost too much time. But in my opinion, I think there is still some room for improvement. First, it doesn't take into account the error for the 2D segmentation part. If one pixel belongs to the object but it has been forgotten, it may have an impact on the final result. As the points on the border of the object may play an important role in the prediction of the pose, it should be better to preserve them than ignore them. One possible solution is to decrease the threshold for the 2D segmentation part in order to consider more points. One another solution is to extend the border detected by several pixels. For me, I would like to choose the first one. If we used the second one, it could include so many points which don't belong to the object actually, and it will result in the diminution of the precision for the global feature.

In conclusion, the DenseFusion does well in introducing the idea of confidence for selecting the best pose, but it needs to improve how to fuse the colour and geometric features well.

4 Proposed dataset

4.1 Why we need a new dataset?

First of all, and obviously, the existing dataset doesn't fulfil our target. The existing dataset doesn't concern the vegetables and fruits only. Linemod and YCB-Video own the object like bananas but the others distract objects like iron and camera, which is different from the application situation. In the real industrial picking scenes, it is really hard to see the foods and the electronic devices are in the same container for autonomous picking . We need to build new ones specific to our demand.

Then, the challenge of the existing dataset cannot satisfy our requirement. We want to research a scene of picking different kinds of fruit and vegetables in a container. For now, the most relevant existing dataset is the Bin-P [8]. However, there are some important challenges which have not been involved during the dataset. In the scene, there is only one kind of instance in each scene. However, it is impossible in the real application. In the bin, it should contain different kinds of objects at the same time to make sure the robots can not only predict pose correctly, but also the corresponding instance. As a result, we can put several different objects in the same bin for the training procedure. Moreover, we need to choose the object precisely to make the trained model robust. For example, if we make apples and bananas in the container at the same time, the network can handle them with a high confidence score, but if we compare the apple and the tomato, the difference will be smaller. It will force the trained model to remember more details about the two instances(objects) to make the network more robust. Besides, if we do a category-level pose estimation, we can add different instances of the same object in the same scene in order to make the network be able to identify them correctly under the distractor. This demands a high level of accuracy of the network. If we put a threshold too small, it may not distinguish the apples and the tomatoes but if we put it too big, it may not be able to identify some of the apple instances. In conclusion, the similar object distractor will be a very important challenge to make our algorithm robust.

One other important challenge we need in our dataset is the occlusion. In the real world application, it should have enough instances in the bin so that the heavy occlusion is an inevitable problem. For the dataset like Linemod, it doesn't include this challenge. How to implement this challenge into our dataset is a good question. In the Bin-P, for the synthetic and the real instance, they use two different strategies. And Because of their objects are in strange shape, the random dropping from a high distance can usually create totally different scenes. However in our case, most of the fruit and vegetables are round(more generalised, in a regular shape). If we do the same thing, the scenes created usually tend to have the same structure. As in the figure below, the blue circle represents the instances. The round objects always try to find the space between the lower layers and stand in that particular position. We cannot say this is a problem because in the real-world application, it is also a common situation. Make use of this can also make the trained model be familiar with that. However, without enough data(the scene generated may be almost the same), the trained model may not work well. We can use

the automatic dropping to create 80% of the scenes and create the rest in the manual way, for example, putting the instances in the position we want. One other solution is we can add some obstacles in the background(like the stones) to make things different. Some other possibility is adding the instance with strange shapes like bananas.

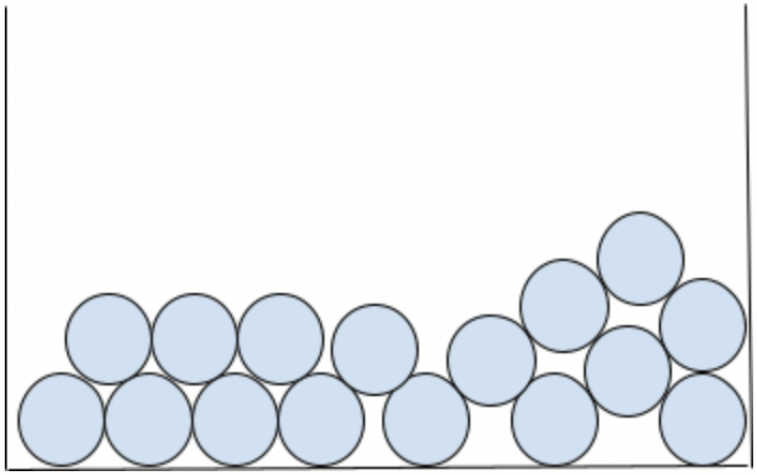


Figure 18: A simple view of the bin consist only of round instances

4.2 Our new dataset

First of all, we need to research that our case will be at an instance-level or in a category-level estimation. From one side, the 6D pose estimation at the instance-level is easy. We can have the object's CAD model to represent it and use the algorithms like the G2L_net and the DenseFusion to do the pose estimation. For the situation of our research, the same kinds of fruits and vegetables have similar geometric and colour properties. As a result of that, the CAD model model of an instance can nearly represent a set of the objects. For example, red apples always have the colour red and are 10 centimetres in diameter. Although we know that these apples in bins are not exactly the same and they have some slight differences, they can still be well represented by the CAD model of an apple which generalises all these features. However, there are still some problems. As the network of instance pose estimation is very deep, according to the butterfly effect, a slight difference in the shape or colour of the ground truth object can have a huge impact on the result. For G2L_net, the impact is limited because we can use the range of the diameter to replace the procedure of fixing the diameter during the 3D point pre-selection part and the colour features are not extracted separately. But in the DenseFusion, the false in colour can make the colour features far from the one we want. Further, for each dense pixel, the confidence for the corresponding points will be not correct. This can make us make the wrong choice for the pose, which will also largely influence the final result. So the category-level pose estimation is a good choice. In conclusion, we can make our

first version of the dataset to the instance-level to see the accuracy of the result and then do a category-level pose estimation to do a comparison of these two dataset and the corresponding result.

Then, we need to identify the challenges involved in the dataset. First of all, we must have the Similar-Looking Distractor and the occlusion challenge, as described in the last part. The cluttering challenge is also important. As we need to add some obstacles in the bin to make the scene different from one to the other, the background will become complicated. As a result of that, we need to add the challenge of the cluttering. For the viewpoint changes, it is also useful. As the picking of the object is a moving procedure for the robot, the viewpoint from the camera is always moving, which means for each frame during the grasping, the viewpoint is different. The challenge of the texture-less is useless because all the fruits and vegetables are instances with texture.

The picture below shows the sample image of our dataset. The picture on the right is the color map, and on the left is the depth. We combine these two to get the point cloud.

(a) Color map (b) Depth map

Figure 19: Our dataset

5 Future work

As my internship will not be finished after the redaction of this report and the defence, I would like to talk about several works we are going to do later. This includes the test of our dataset and to propose and test several new algorithms to make the result better.

5.1 Test of our dataset

After the dataset has been created, we need to test its performance under the proposed algorithms. For the instance-level pose estimation, we can test them by using the two principal algorithms I have talked about in this report, the G2L_net [1] and DenseFusion [2]. As the DenseFusion can work well under the situation where the noise is big and has heavy occlusion.

We also want to test if it can work well under our dataset. For the G2L_net, we can make it as a comparison. During the test, several things need to be fixed to make the comparison more persuadable. The first one is the quantity of the training data, as the computational cost of the two methods are different and the limitation of the computational capability of the machine of the lab, we propose to limit the training data in order to save time. For example, we can choose to train less objects in one training procedure and consider the others as the obstacles. I can also create a scene with heavy cluttering.

Then we need to test the second version of our dataset, the category-level estimation, by using the method NOCS [7].

5.2 Possible proposed new methods

If the existing new methods don't work well, we need to propose some new methods which fit the challenges involved in our dataset. Some proposed change has been proposed during the analysis of the disadvantage and the advantage of the G2L_Net and the DenseFusion. Besides, for the CVPR 2022 which has just been held, there are also several interesting papers which we can make use of, like FS-6D [17], which is based on the DenseFusion [2], and Uni-6D [18] which make use of the positional encoding to stock the information without using the convolution network.

6 Conclusion

During this report, we first study the concept of the 6D pose estimation, knowing that it could be divided into two levels, instance-level and category-level. Then, we have done a deep study of two state-of-the-art methods G2L_net and DenseFusion: their idea, their architecture, their dataset, their experiment, etc. Then, I do some analysis of the two methods, what are the advantages and what are the disadvantages and do a horizontal comparison of them. Finally, we proposed a new dataset for our application. After the defence, we would like also to propose some new methods and test the G2L_net and DenseFusion based on our dataset.

7 Personal conclusion

Besides the conclusion of the report, I would also like to do a personal conclusion of the internship although it still has two months to go.

I started the internship at the beginning of March because I am on exchange for the first semester and it ends late. I took a little introductory course on Computer Vision during the exchange and it aroused my interest in doing research in this domain. As a student who didn't have so much in Computer Vision before, this internship is challenging for me. First of all, I need to use several weeks to get to know the basic conceptions of the 6D pose estimation by watching

several related videos. Then, for the hands-on, I will also do some basic implementation of the Computer Vision method, like the YOLO and 3DGCN. After that, I went deep into the network which the mentor asked me to work on. For them, I read word by word of their paper and their code to try to understand the details. Besides them, I have also read several papers about the other algorithms for comparison.

Before I went to do the internship, I thought that the job at a lab would be boring. However, it is totally different. During the weekly meeting, everyone in our group has the chance to present what he/she has done during the last week and discuss the ideas. For example, I, together with a Ph.D. student, presented the G2L_net work to all the members of the teams to give them a first impression and the result. This is quite cool that everyone may go deep into their domain and exchange in order that everyone will not miss some good papers. However, from this internship, I have also encountered some problems. Before I attended this internship, I thought the most important in the whole machine learning pipeline, is the network architecture. But the data prepossessing is also very important. How to transform the RGB-D data into the point cloud? How to feed the data correctly to the network together with its labels? These are some of the main difficulties I met.

One other problem is about parallelization. Before the start of the internship, I have just the experience of vector operation parallelization but during this intern, I have met several times the matrix operation parallelization, which is a little bit difficult for me.

Finally, I would like to say thank you to Mr. Chen Liming, my mentor at the lab together with Mr. Dellandréa Emmanuel, who gives me a lot of advice to help me improve fast in this domain. Without them, I cannot achieve the result. Further, I would like to say thank you to M. Duffner Stefan, who always keeps an eye on my progress and helps me with some problems in the algorithms.

References

[1] Wei Chen, Xi Jia, Hyung Jin Chang, Jinming Duan, and Ales Leonardis. G2l-net: Global to local network for real-time 6d pose estimation with embedding vector features. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[2] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. 2019.

[3] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[4] Stefan Hinterstoisser, Cedric Cagniart, Slobodan Ilic, Peter Sturm, Nassir Navab, Pascal Fua, and Vincent Lepetit. Gradient response maps for real-time detection of textureless

objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):876–888, 2012.

[5] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. 2018.

[6] Caner Sahin, Guillermo Garcia-Hernando, Juil Sock, and Tae-Kyun Kim. Instance- and category-level 6d object pose estimation. *Computer Vision and Pattern Recognition (cs.CV)*, June 2020.

[7] He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J. Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[8] Kilian Kleeberger, Christian Landgraf, and Marco F. Huber. Large-scale 6d object pose estimation dataset for industrial bin-picking. *Computer Vision and Pattern Recognition (cs.CV)*, December 2019.

[9] C. Eppner, S. Hofer, R. Jonschkowski, R. Martin-Martin, A. Sieverling, V. Wall, and O. Brock. Lessons from the amazon picking challenge: Four aspects of building robotic systems. *Robotics: Science and Systems*, 2016.

[10] Zhong-Dan Lan. Long Quan. Linear n-point camera pose determination. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Institute of Electrical and Electronics Engineers,*.

[11] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018. cite arxiv:1804.02767Comment: Tech Report.

[12] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018.

[13] Sida Peng, Yuan Liu, Qixing Huang, Xiaowei Zhou, and Hujun Bao. Pvnet: Pixel-wise voting network for 6dof pose estimation. In *CVPR*, 2019.

[14] Arul Selvam Periyasamy, Max Schwarz, and Sven Behnke. Robust 6d object pose estimation in cluttered scenes using semantic segmentation and pose regression networks. 2018.

[15] Wei Chen, Xi Jia, Hyung Jin Chang, Jinming Duan, Shen Linlin, and Ales Leonardis. Fs-net: Fast shape-based network for category-level 6d object pose estimation with decoupled rotation mechanism. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1581–1590, June 2021.

[16] R. Bregier, F. Devernay, L. Leyrit, and J. L. Crowley. Symmetry aware evaluation of 3d object detection and pose estimation in scenes of many parts in bulk. In *n IEEE International Conference on Computer Vision (ICCV)*, 2017.

[17] He Yisheng, Wang Yao, Fan Haoqiang, Chen Qifeng, and Sun Jian. Fs6d: Few-shot 6d pose estimation of novel objects. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

[18] Xiaoke Jiang, Donghai Li, Hao Chen, Ye Zheng, Rui Zhao, and Liwei Wu. Uni6d: A unified cnn framework without projection breakdown for 6d pose estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, March 2022.

Annexes

Annex A: About the rotation matrix

Although a rotation matrix is a matrix of 3 by 3, which contains 9 different number, they still have a degree of freedom 3 because for the rotation rotation matrix R, we need to satisfy that $R^T R = I$, $det(R) = +1$ and the dot product of each column and each row with itself is 1(Normal) and the dot product of each different row and each different column is 0(Orthogonal). Let's look at it more precisely: for the first column of the matrix, it has 3 numbers, 2 of them are free because we need to use the third one to control the norm to be 1. For the second row, there is only one degree of freedom, one of the others is used to control the norm and the other is to make sure that it's orthogonal to the first row. For the third column, it will completely depend on the first two given that it should be orthogonal to them. Actually, it is the vectorial product of the first two columns. All the rotation matrix can be represent in the form of $SO(3)$ where SO represents Special Orthogonal. Special means the determinant of the matrix is limited to $+1$. As the orthogonal matrix could have the determinant of 1,where $+1$ means the pure rotation in the real world cases and -1 means a rotation plus a mirror flip of the object, which is impossible for real world problems.

The 3 angles (α,β,γ) for rotations, are actually the Euler Angle, for each of the angles, it can form a matrix with 1 in the row and columns with respect to the rotation axis. For more details, please refer to the Wikipedia page.

The code for transforming the Euler angle to rotation matrix. The input can be parallelized in the pytorch tensor of shape $(N, 3)$ where N is the number of groups of Euler angles.

Listing 2: creating rotation matrix from RPY value

```
def generateRotationMatrixfromRPYMatrix(rpy):

    length = rpy.shape[0]
    R1 = torch.Tensor(length,3)
    R2 = torch.Tensor(length,3)
    R3 = torch.Tensor(length,3)
```

```
R1[:,0] = np.cos(rpy[:,2])*np.cos(rpy[:,1])
R1[:,1] = np.cos(rpy[:,2])*np.sin(rpy[:,1])*np.sin(rpy[:,0]) -
np.sin(rpy[:,2])*np.cos(rpy[:,0])
R1[:,2] = np.cos(rpy[:,2])*np.sin(rpy[:,1])*np.cos(rpy[:,0]) +
np.sin(rpy[:,2])*np.sin(rpy[:,0])

R2[:,0] = np.sin(rpy[:,2])*np.cos(rpy[:,1])
R2[:,1] = np.sin(rpy[:,2])*np.sin(rpy[:,1])*np.sin(rpy[:,0]) +
np.cos(rpy[:,2])*np.cos(rpy[:,0])
R2[:,2] = np.sin(rpy[:,2])*np.sin(rpy[:,1])*np.cos(rpy[:,0]) -
np.cos(rpy[:,2])*np.sin(rpy[:,0])

R3[:,0] = -np.sin(rpy[:,1])
R3[:,1] = np.cos(rpy[:,1])*np.sin(rpy[:,0])
R3[:,2] = np.cos(rpy[:,1])*np.cos(rpy[:,0])

R = torch.cat([R1,R2,R3],1)
R = R.reshape(-1,3)

return R
```

Annex B: PoseCNN

As the method G2L_Net and the DenseFusion both depends on the PoseCNN(both the idea and the architecture).But it is not a core part of my report, so I would like to present a little about it in the annex.

The main idea of the PoseCNN is to do the translation and rotation prediction separately. For the translation, also named as the **2D object localization** in the paper, it also separates it in two parts. The first part tries to predict the object's centre position in the image as $c = (c_x, c_y)^T$, and the second part is to predict the depth of the centre of the object in the image, note as T_z. For the rotation, named as **3D rotation regression**, this paper just does some regression on the object, by using the Quaternion representation. As the object to detect in the image has only one instance, the network first does a segmentation task to identify the pixels belonging to the object (The upper branch of the architecture). After getting the segmentation mask, the middle branch of the network will predict the relative position of the centre for each pixel of the mask. Then we do a voting to get the overall position for c. The lower branch uses the features from the segmentation mask to do a regression to get the Quaternion representation. The final result will combine together the translation and the Quaternion representation to get the final 6D pose. One another interesting thing proposed in this paper is the loss. The point cloud applied by the ground-truth pose will be considered as

the target. And the point cloud applied by the predicted pose will be seen as the prediction. For a non-symmetric object, the final loss is the MSE of the distance for each point in the target and prediction respectively. For the symmetric object, it will be difficult to identify which point refers to the target. So it chooses the nearest point to the predicted point in order to calculate the loss. It can make full use of the contribution of the loss for each point by comparing the loss directly used on the numerical values of the pose. [5]

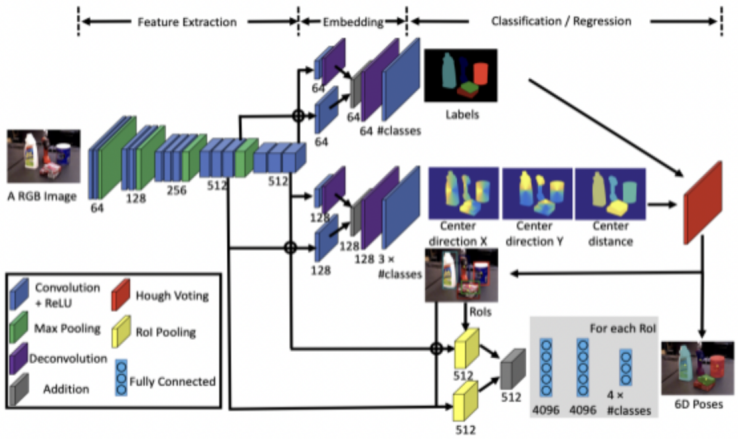


Figure 20: Architecture for PoseCNN

Annex C: Kabsch Algorithm

Assuming that P and Q are two sets of points under two different coordinates, kabsch is the algorithm to calculate the rotation matrix between the two sets. The main algorithms is:

- Normalise P and Q
- Calculate $C = PQ^T$
- Do a singular value decomposition on C, $[V, S, W] = SVD(C)$
- Calculate a corrector $d = sign(det(C))$ (It is closely related to the rotation matrix where the $det(R) = +1$)
- Calculate the final rotation matrix $W \cdot diag(1, 1, d) \cdot V^T$

Annex D: ICP algorithm

ICP algorithm is a method which can calculate the rotation matrix and the translation matrix from the source cloud point and the target point cloud in an iterative way.

First we apply the pre-estimation rotation and translation to the source point cloud. Then for each point in the source point cloud, we can find its closest point to the target point cloud. We assume the two points in a pair correspond to the same point in reality. So we can do a least square to minimise the loss between the two corresponding points to get a new R and new T as below:

$$L(R,T) = \frac{1}{N_p}\sum_{i=1}^{N_p} \|p_t^i - R \cdot p_s^i - T\|_2^2$$

Where the p_t^i represent the ith point in the target point cloud and p_s^i is the corresponding point in the source point cloud. After getting the new Rotation and Translation, the previous correspondence may change, so we need to find the new correspondence and minimise the loss in an iterative way. The running speed of this ICP algorithm is highly depend on how far is the pre-estimated R and T from the one we want.

The Deep learning methods on 6D Pose estimation [R&D]

Yuxuan ZONG

Abstract

For robotics manipulation of the real object, it is important to get to know the gesture of the object in order that the robot can get in the maximum contact with the object. It can also be utilised in the other domains like augmented reality and autonomous driving. The objective of 6D object detection is to predict the translation and the rotation of the object from its own coordinate to the camera(detector) space. The 6D is the short form of the '6 degree of freedom', which corresponds to the 3 axes of translation (x,y,z) and the 3 angles (α,β,γ) for rotations. In fact, this information will be presented finally in the form of the translation vector of size 3 and a rotation matrix of size 3 by 3, more precisely, SO(3). (You can find some details about it in the annex) In this report, we will first focus on two existing method, which both achieve the state-of-the-art: G2L_Net [1] and DenseFusion [2]. Then we will introduce several most used datasets for 6D pose estimation which introduce different kinds of challenges. After that, we will analyse the pros and cons of the two algorithms and propose a new dataset for 6D pose estimation. Finally, we will propose a new algorithm which works well on the new dataset with different existing challenges in the future.

Keywords: Deep Learning; 6D Pose estimation; G2L_Net; DenseFusion; 3D Semantic Segmentation

Résumé

Pour la manipulation robotique d'un objet réel, il est important de connaître le geste de l'objet afin que le robot puisse entrer au maximum en contact avec l'objet. Elle peut également être utilisée dans d'autres domaines comme la réalité augmentée et la conduite autonome. L'objectif de la détection d'objets 6D est de prédire la translation et la rotation de l'objet dans une scène donnée. Le 6D est la forme courte des '6 degrés de liberté', qui correspondent aux 3 axes de translation (x,y,z) et aux 3 angles (α,β,γ) pour les rotations. En fait, ces informations seront finalement présentées sous la forme d'un vecteur de translation de taille 3 et d'une matrice de rotation de taille 3 par 3. Dans ce rapport, nous nous intéresserons d'abord à deux méthodes existantes, qui réalisent toutes deux l'état de l'art: G2L_Net [1] et DenseFusion [2]. Ensuite, nous présenterons plusieurs datasets les plus utilisés pour l'estimation de la pose 6D, qui présentent différents types de défis. Ensuite, nous analyserons les avantages et les inconvénients des deux algorithmes et proposerons un nouveau jeu de données pour l'estimation de la pose 6D. Enfin, nous proposerons un nouvel algorithme qui fonctionne bien sur le nouveau jeu de données avec différents défis existants à l'avenir.

Mots-clés : Apprentissage profond ; Estimation de la pose 6D ; G2L_Net ; DenseFusion ; Segmentation sémantique 3D