Measurements on Recognition of Objects to Grasp with a Robot Arm in a Hybrid System: 6D Pose Estimation Based on DenseFusion Method

Getnet Demil[1], Bolutife Atoki[2], Professor Jenny Benois Pineau[3]
getnet-demil.jenberia@etu.u-bordeaux.fr[1], bolutife-oluwabunmi.atoki@etu.u-bordeaux.fr[2], jenny.benois-pineau@u-bordeaux.fr[3]

***Abstract*—**
According to the European Council [1], over 53% of the 87 million people in Europe living with a disability are facing upper limb disability. This project proposes an approach using a vision-aided prosthesis arm to assist people with upper limb disabilities in efficiently interacting with objects. A crucial part of the project involves using the DenseFusion algorithm to estimate the 6D pose of the object to be grasped. In this paper our primary focus was on accurately determining the position and orientation of objects in a kitchen environment, specifically utensils, to enable users to pick them up with ease. Building upon the foundational work of Fejer et al. [48], this paper extends the vision system's capabilities by implementing pose estimation through RGB, depth, and binary mask images of the object and scene. To ensure the robust performance of our proposed system, we generated a proprietary dataset using Unity and HTC VIVE headsets. The dataset comprises essential information for the DenseFusion algorithm, structured in the LINEMOD dataset format. The Unity-based process involves 3D modeling of the object to be grasped, and extracting intrinsic and extrinsic parameters, including rotational and translational matrices, and RGB images. For the success of unity modeling, we have adopted methods from a research team HYBRIDE from Institut de Neurosciences Cognitives et Intégratives d'Aquitaine (INCIA). To asses the system's performance we compute error functional MSE after the back projection of the estimated pose into the image plane.

I. Introduction

The quest for innovative solutions to enhance daily life for people with disabilities is more pertinent than ever, particularly in Europe where 87 million individuals grapple with various disabilities [1]. One in every four adults suffers from upper limb disabilities, which makes it challenging for them to perform routine activities with ease. To address this need, a project addresses the the methods of scene analysis for object grasping with a vision-aided prosthesis arm that empowers users by facilitating seamless engagement in daily tasks. Our study focuses on developing good pose estimation as a crucial step for this vision-aided prosthesis arm with a specific application in cluttered environments, such as kitchen, where grasping and picking up utensils pose unique challenges. Pose estimation is the process of determining the position and orientation of an object in a given environment. This is a crucial aspect of prosthetic arm perception because knowing the pose of objects is essential for effective interaction with them. By providing information about the pose, the prosthetic arm can navigate, manipulate objects, and perform tasks accurately. Building upon the foundational work of Fejer et al. [48], our research extends the capabilities of the vision system by enhancing the pose estimation using RGB images, depth, and binary mask images of the objects.

A. Problem Statement

Accurate position and orientation estimation of an object is crucial for a prosthetic arm to effectively interact with the object. It is called in literature "Pose estimation" [6]

3D pose estimation provides the position and orientation of an object in three-dimensional space, but it doesn't give complete rotational information. On the other hand, 6D pose estimation includes both translation and rotation in all six degrees of freedom. This offers a more comprehensive understanding of an object's pose, which is particularly important for tasks like reaching and grasping movements of a prosthetic arm.

The motivation behind this work is not merely theoretical; it stems from a genuine desire to address real-world problems faced by individuals with disabilities. Through this research, we aim not only to solve a tangible problem, but also to contribute to the ongoing dialogue surrounding inclusive technologies and their profound impact on the lives of individuals with disabilities.

B. Objectives

1) General Objective:

- 6D pose estimation based on Densefusion method

2) Specific Objective:

- Dataset preparation for the project
- Calculating the extrinsic parameter of the frame
- Extracting the intrinsic parameter from the HTC VIVE Headset
- Preparing a ground-truth information
- Calculating the bounding box of the instance
- Implementing Segmentation algorithm
- Implementing a binary mask extraction algorithm for a frame
- Implementing a depth map estimation algorithm
- Training and testing of DenseFusion method [2]
- Fine-tuning and optimization of the model

As we delve into the subsequent sections, we will present the detailed methodology, the significance of 6D pose estimation, and the innovative use of DenseFusion. Furthermore, we will showcase the results of our comprehensive evaluation, emphasizing accuracy and efficiency metrics, to substantiate the transformative potential of 6D pose estimation.

II. Related work

(Collet et. al., 2011) present MOPED, a framework for Multiple Object Pose Estimation and Detection that seamlessly integrates single-image and multi-image object recognition and pose estimation in one optimized, robust, and scalable framework. ICE is easy to parallelize, and easily integrates single- and multi-camera object recognition and pose estimation [3].

Learning articulated object pose is inherently difficult because the pose is high dimensional but has many structural constraints. (Zhou et. al., 2016) propose to directly embed a kinematic object model into the deep neutral network learning for general articulated object pose estimation [4]. (Pavlakos et. al., 2017) present a novel approach for estimating the continuous six degrees of freedom (6-DoF) pose (3D translation and rotation) of an object from a single RGB image [5].

State-of-the-art accuracy in class-based object pose estimation has been demonstrated on the large-scale PASCAL3D+ dataset. In 2017, Xiang et. al. introduced PoseCNN - a new Convolutional Neural Network for 6D object pose estimation. They also contributed a large-scale video dataset for 6D object pose estimation called the YCB-Video dataset. Despite significant progress in visual object detection and segmentation through the use of deep learning methods, object pose estimation remains a challenging task [6].

(Do et. al., 2018) introduce an end-to-end deep learning framework, named Deep-6DPose, that jointly detects, segments, and most importantly recovers 6D poses of object instances from a single RGB image [7]. (Kanezaki et. al., 2018) propose a Convolutional Neural Network (CNN)-based model "RotationNet," which takes multi-view images of an object as input and jointly estimates its pose and object category, and also show that RotationNet, even trained without known poses, achieves the state-of-the-art performance on an object pose estimation dataset [8].

(Hodan et. al., 2018) propose a benchmark for 6D pose estimation of a rigid object from a single RGB-D input image. The training data consists of a texture-mapped 3D object model or images of the object in known 6D poses [9]. (Du et. al., 2019) present a comprehensive survey on vision-based robotic grasping. Lots of objects pose estimation methods do not need object localization. They conduct object localization and object pose estimation jointly [10].

III. Proposed approach

The DenseFusion method [2] represents a significant advancement in the field of 6D object pose estimation from RGB-D images. It presents a generic framework for estimating the 6D pose of known objects, utilizing a fusion of RGB and depth data to predict the pose through pixel-wise feature extraction (Wang et al., 2019) [2].

This approach is particularly valuable in cluttered scenes, where accurate pose estimation is challenging (Xiang et al., 2017). DenseFusion's iterative dense fusion technique allows for robust pose estimation, even in the presence of occlusions [6].

Furthermore, it has been shown to outperform other methods in terms of accuracy and efficiency, making it a state-of-the-art solution in the field (Xu et al., 2019) [17]. The method DenseFusion leverages the advantages of deep learning and convolutional neural networks (CNNs) to achieve accurate 6D pose estimation (Wang et al., 2019) [2].

By fusing RGB and depth data in the embedding space and employing a PointNet-like architecture [11] for feature extraction, DenseFusion demonstrates the potential of deep learning in addressing complex computer vision tasks (Lee et al., 2023) [13]. Additionally, the iterative refinement process employed in DenseFusion aligns with the trend in pose estimation approaches, where iterative refinement is used to enhance the accuracy of pose estimates (Periyasamy, 2022) [14].

Comparative studies have demonstrated the superiority of DenseFusion over other state-of-the-art methods, showcasing its effectiveness in challenging scenarios (Byambaa et al., 2022) [12]. Furthermore, the method has been compared with other techniques, such as DeepIM (Yi et al., 2019) [18], highlighting its competitive performance in the field of 6D pose estimation.

These comparisons underscore the significance of DenseFusion as a leading approach in the domain of object pose estimation. In conclusion, DenseFusion represents a significant advancement in 6D object pose estimation, offering a robust and accurate framework for estimating the pose of known objects from RGB-D images. Its utilization of deep learning, iterative refinement, and comparative performance against other state-of-the-art methods solidify its position as a leading solution in the field.

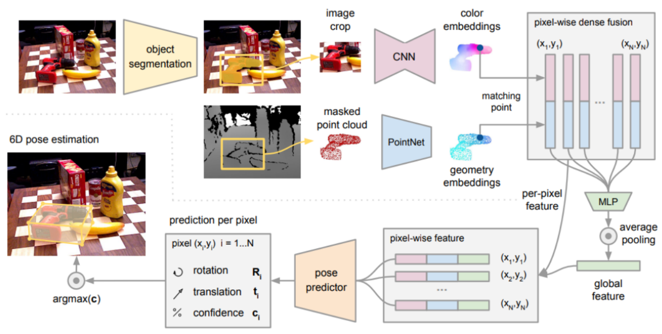


Fig. 1: Dense Fusion Architecture [2]

The block-diagram of DenseFusion method is illustrated in

TABLE I: State of the art approaches/methods

Papers	**DenseFusion**	**PoseCNN**	**BB8**	**DeepIM**	**DSAC++**	**PoseRBPF**
Wang 2019, [2]	✓	✗	✗	✗	✗	✗
Azeem 2020, [30]	✗	✓	✗	✗	✗	✗
Rad 2018, [31]	✗	✗	✓	✗	✗	✗
Li 2018, [32]	✗	✗	✗	✓	✗	✗
Brachmann 2016, [33]	✗	✗	✗	✗	✓	✗
Kae 2016, [34]	✗	✗	✗	✗	✗	✓
Xu 2018, [35]	✗	✗	✗	✗	✗	✓

Figure 9b. The model takes RGB-D images as input, extracts features, predicts an initial pose, iteratively refines the pose, fuses color and depth information densely, and outputs the refined 6D pose estimate for objects in the scene. Let us denote by X the input data to DenseFusion model:

$$X = (I_{RGB}, I_D) \tag{1}$$

- I_{RGB}: RGB image. - I_D: Depth image. Then the approach can be formalized as follows.

$$F = \text{FeatureExtractor}(X) \tag{2}$$

- The model extracts features from the RGB-D input to capture both appearance and geometric information.

$$P = \text{PosePredictor}(F) \tag{3}$$

- The model uses a neural network to predict the 6D pose (P) based on the extracted features (F).

$$P_{\text{refined}} = \text{IterativeRefinement}(P, X) \tag{4}$$

- Iterative refinement process to improve pose estimation accuracy.
- It involves reprojection of 3D model points onto 2D image and refining the pose.

$$\text{Output} = \text{DenseFusion}(I_{RGB}, I_D, P_{\text{refined}}) \tag{5}$$

- Combining color and depth information densely to improve pose estimation, especially in cluttered scenes. The filnal 6D pose, that is the set of estimated parameters

$$Y = (t_x, t_y, t_z, \theta_x, \theta_y, \theta_z) \tag{6}$$

is obtained as

$$Y = P_{\text{refined}} \tag{7}$$

The final parameters are - t_x, t_y, t_z - translations. - $\theta_x, \theta_y, \theta_z$ - rotation angles.

IV. Dataset Preparation

In this research we have used two datasets. One of them is a LineMOD dataset [20], [21] used by research community for the benchmarking of pose estimation algorithms. The second one - is a synthetic, domain specific dataset which we had to generate.

A. *Datasets and Structure*

Our first experiment with DenseFusion (in the previous semester) was on the benchmark 6D pose estimation dataset called LineMOD [20], [21], which comprises 15,783 images of 13 types of weakly textured objects with different shapes. The dataset is an appropriate testing ground for DenseFusion as it includes masks produced by a trained vanilla SegNet model, allowing for comprehensive evaluation of the model's performance [2] and the ground truth depth maps. It is widely used for benchmarking 6D object pose estimation algorithms and is characterized by its variety of weakly textured objects. The significance of this dataset in algorithm benchmarking makes it an ideal choice for testing and validation of the DenseFusion model. The model leverages this dataset to address the challenges of 6D object pose estimation from RGB-D images [20]–[22].

Due to the application area of the Pose estimation system for this research (which is for application for a vision-aided prothesis arm in a kitchen environment) and the fact that the setup and objects contained in the LineMOD dataset do not completely capture these requirements, the second dataset is a *Synthetic* one. Synthetic data are generated using Unity® [23] created by the research team HYBRIDE from the Institut de Neurosciences Cognitives et Intégratives d'Aquitaine (INCIA), .

B. *LineMOD dataset: Data Folder*

The hierarchy and elements of folders in the Linemod dataset are highlighted in the figure 2 below. Here "01" is an object identifier, "depth" contains a depth map, "mask" is the binary mask of the object, "rgb" contains corresponding rgb frames, "gt.yml" contains the ground truth parameters, "info.yml" contains the intrinsic parameter matrix of the camera used per frame, "train.txt" and "test.txt" contains frame numbers to use for training and testing respectively.

C. *Synthetic dataset*

For the synthetic data set we need to obtain the same information as for the LineMOD dataset. The synthetic dataset was generated with HTC Vive [25]. HTC Vive are a series of virtual reality headsets developed by HTC Corporation in collaboration with Valve Corporation. The specification of the Head set is presented in table II. The device is depicted in

```
datasets
└── linemod
    └── Linemod_preprocessed
        ├── data
        │   └── 01
        │       ├── depth
        │       ├── mask
        │       ├── rgb
        │       ├── gt.yml
        │       ├── info.yml
        │       ├── train.txt
        │       └── test.txt
        ├── models
        │   ├── models_info.yml
        │   └── obj_01.ply
        └── dataset.py
```

Fig. 2: Directory structure of LineMOD datasets.

figure 3. These devices are famous for offering the possibility of entering an immersive VR world in which the user can play several VR, Mixed Reality (MR) or Augmented Reality (AR) games. We worked only on one object instance. Its identifier is 01, and it pertains to a water bottle generated using Unity® [23] that needs to be grasped. In the following we describe the generation of the required input data.

*1) **RGB Image Generation**:* The virtual environment (model) was developed on Unity® [23] and loaded on the HTC Vive headset, subjects were asked to wear the headsets and interact with the virtual environments. These interactions were recorded and split into frames to serve as the RGB images. The headset HTC Vive has many variables in the output JSON file among them timestamp: Time in the experiment at which the sample was acquired, tgtNumber: Id of the target the subject is reaching1, tgtType: Type of target. Once we generated the virtual model, we used a 90hz frame rate to produce the RGB image frames from Unity. .

Specifications	**HTC Vive**
Display	OLED
Resolution	1080 x 1200, pixels per eye
Refresh rate	90 Hz
Field of view	110º
Sensors	Accelerometer, Gyroscope

TABLE II: Specification of the HTC VIVE VR headset [25]

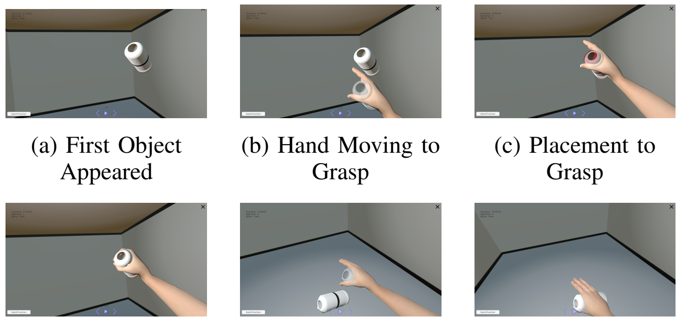
*2) **Binary Mask**:* We implemented the detectron2 [24] segmentation algorithm to get the binary mask. Detectron2 is a modular object detection and segmentation library developed by Facebook AI Research (FAIR) that implements state-of-the-art object detection and segmentation algorithms. It is a foundational model in the sense as it has been trained on a very large amount of images.

In the image processing pipeline utilizing Detectron2, the input RGB image undergoes semantic segmentation, a process that yields a segmentation map outlining distinct instances within the frame. This segmentation map, rich with boundary coordinates that demarcate different instances, serves as the foundation for binary mask extraction. Employing these boundary coordinates, binary masks are generated for each identified instance, resulting in a comprehensive representation of the spatial extent occupied by each object [24].

We have an image represented as a 2D array I with dimensions $M \times N$, where M is the height and N is the width. A binary mask B of the same dimensions can be created using boundary coordinates of the water bottle $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$.

The process of the extraction of binary masks for objects,

Fig. 3: HTC Vive Headset [25]



(a) First Object Appeared (b) Hand Moving to Grasp (c) Placement to Grasp

(d) The Object Grasped (e) New Position to Place (f) Object Placed at New Place

Fig. 4: Main sequences of the actions peformed in the video frames

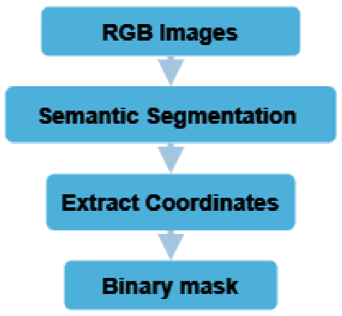


Fig. 5: Binary mask extraction using detectron2

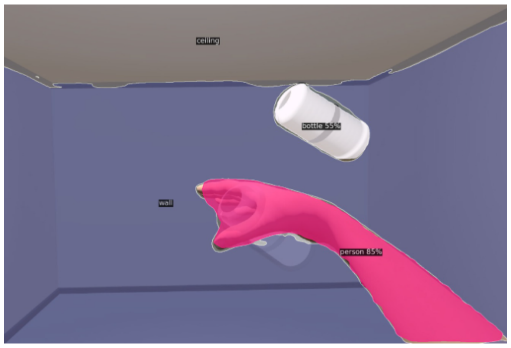
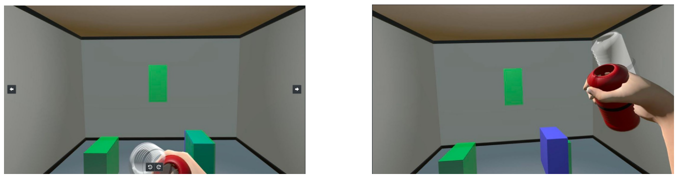


Fig. 6: Semantic Segmentation of the Scene



(a) The water bottle cut-out (b) The water bottle occluded

Fig. 7: Sample case where Detectron2 has failed to detect

B is illustrated in figure 6. We can distinguish between the binary masks for different objects in the scene (see figure 6 due to the object ID supplied by detectron2. In the following we will denote the binary mask as $B(u, v)$, where u, v are the coordinates of a pixel in the image plane. In the pixels belonging to the object of interest, $B(u, v) = 1$ and in the background pixels and other objects $B(u, v) = 0$. Then we apply this mask to our original RGB image to extract the object rgb values, the masked image is denoted by $M(u, v)$:

$$\mathrm{M}(u, v) = \mathrm{OriginalImage}(u, v) \cdot B_(u, v) \tag{8}$$

We started with a total of 37,872 RGB frames that were generated from Unity. However, after the Detectron2 operation, we were only able to obtain good binary masks for 2,320 frames. This was due to various reasons, such as hands overlaying water bottles, water bottles being cut out of the frame, or no bottles appearing in the frame at all.

*3) **Ground-truth, gt.yml** :* The ground-truth, gt.yml information includes the object's bounding box, rotational matrix, translational matrix, and unique object ID.

*4) **Bounding box**:* The bounding box values were obtained from the binary mask output [**?**], which is essentially a black and white image of the object of interest. From this image, we can identify a boundary surrounding the white part of the image, which forms a boundary box.

The bounding box can then be defined using the minimum and maximum coordinates of the white region:

$$x_{\min} = \min_u \{u : B(u, v) = 1\}, \quad x_{\max} = \max_u \{u : B(u, v) = 1\}$$

$$y_{\min} = \min_v \{v : B(u, v) = 1\}, \quad y_{\max} = \max_v \{v : B(u, v) = 1\}$$

We can then use

$$x_{\min}, x_{\max}, y_{\min}, y_{\max}$$

coordinates to extract the region or draw the bounding box.

*5) **Intrinsic Parameter, info.yml**:* Intrinsic parameters often denoted as K or K_{matrix}, represent the internal characteristics of a camera. These parameters describe the relationship between the coordinates of points in the camera's image plane and their corresponding coordinates in the camera's 3D world coordinate system. For our project, all frames have the same K metrics, because we used the same camera and zooming factor. This value is extracted from the Unity interface, but it originally depends on the acquisition device, which in our case is the HTC VIVE VR headset camera.

*6) **Rotational and Translational Matrix**:* The process of extracting the rotational matrix R and translational vector t from the coordinates of a point P_w in the world coordinate system (target position) and its corresponding coordinates P_c in the camera coordinate system (head position), we can use these points to calculate the rotation matrix R and translation vector t.

Let's define the following variables:
- $P_w = [X_w, Y_w, Z_w]^T$ (3D coordinates of the object in the world coordinate system)
- $P_c = [X_c, Y_c, Z_c]^T$ (3D coordinates of the object in the camera coordinate system, obtained from the HTC headset)
- R (3x3 rotation matrix)
- $t = [t_x, t_y, t_z]^T$ (3D translation vector)

The relationship between the world coordinates and camera coordinates is given by:

$$P_c = R \cdot P_w + t$$

Now, if we have multiple corresponding points, we can set up a system of equations for each point:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = R \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Let's say we have N corresponding points. We can write this system of equations for all points in matrix form:

$$\begin{bmatrix} X_{c1} & Y_{c1} & Z_{c1} \\ X_{c2} & Y_{c2} & Z_{c2} \\ \vdots & \vdots & \vdots \\ X_{cN} & Y_{cN} & Z_{cN} \end{bmatrix} =$$

$$\begin{bmatrix} X_{w1} & Y_{w1} & Z_{w1} & 1 \\ X_{w2} & Y_{w2} & Z_{w2} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ X_{wN} & Y_{wN} & Z_{wN} & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In this matrix equation, the unknowns are the elements of the rotation matrix R and the translation vector t. The matrix on the right side is often denoted as the extrinsic matrix. We can solve for R and t using methods like singular value decomposition (SVD) or least squares.

The process of solving for the rotation matrix R and translation vector t using a least-squares approach. The goal is to minimize the difference between the observed camera coordinates P_c and the transformed world coordinates $R \cdot P_w + t$.

Given the system of equations:

$$P_c = R \cdot P_w + t \tag{9}$$

We have N corresponding points:

$$\begin{bmatrix} X_{c1} \\ Y_{c1} \\ Z_{c1} \end{bmatrix} = R \cdot \begin{bmatrix} X_{w1} \\ Y_{w1} \\ Z_{w1} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

$$\begin{bmatrix} X_{c2} \\ Y_{c2} \\ Z_{c2} \end{bmatrix} = R \cdot \begin{bmatrix} X_{w2} \\ Y_{w2} \\ Z_{w2} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

$$\vdots$$

$$\begin{bmatrix} X_{cN} \\ Y_{cN} \\ Z_{cN} \end{bmatrix} = R \cdot \begin{bmatrix} X_{wN} \\ Y_{wN} \\ Z_{wN} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Now, stack these equations into matrix form:

$$\begin{bmatrix} X_{c1} \\ Y_{c1} \\ Z_{c1} \\ \vdots \\ X_{cN} \\ Y_{cN} \\ Z_{cN} \end{bmatrix} = \begin{bmatrix} X_{w1} & Y_{w1} & Z_{w1} & 1 \\ X_{w2} & Y_{w2} & Z_{w2} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ X_{wN} & Y_{wN} & Z_{wN} & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let's denote the matrix on the left side as M and the extrinsic matrix on the right side as E:

$$M = \begin{bmatrix} X_{c1} \\ Y_{c1} \\ Z_{c1} \\ \vdots \\ X_{cN} \\ Y_{cN} \\ Z_{cN} \end{bmatrix}, \quad E = \begin{bmatrix} X_{w1} & Y_{w1} & Z_{w1} & 1 \\ X_{w2} & Y_{w2} & Z_{w2} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ X_{wN} & Y_{wN} & Z_{wN} & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now, We want to solve for the unknown extrinsic matrix E. The least-squares solution can be found by minimizing the difference between the observed and calculated coordinates:

$$\min_E \|M - E\|_2^2 \tag{10}$$

The solution can be obtained by solving the system of linear equations:

$$E = (E^T E)^{-1} E^T M \tag{11}$$

Once we have the extrinsic matrix E, we can extract the rotation matrix R from the upper-left 3x3 submatrix and the translation vector t from the rightmost column of E.

Given the least-squares solution:

$$E = (E^T E)^{-1} E^T M$$

We can find R and t as follows:

1. Extract R from the upper-left 3x3 submatrix of E:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

2. Extract t from the rightmost column of E:

$$t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

So, the complete least-squares solution for the transformation from the world coordinate system to the camera coordinate system is:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

These values represent the rotational matrix and translational vector that describe the transformation of points from the world coordinate system to the camera coordinate system.

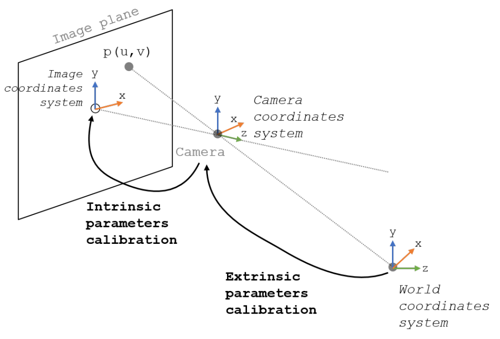


Image plane
p(u,v)
Image coordinates system
y
x
Camera coordinates system
y
x
z
Camera
Intrinsic parameters calibration
Extrinsic parameters calibration
y
x
z
World coordinates system

Fig. 8: Rotational and Translational Parameters

*7) **Depth Map :*** The term depth map in the LineMOD dataset refers to an image where each pixel's value represents the distance from the camera to the corresponding point in the scene. In other words, it is a 2D representation of the depth information in a scene. In the LineMOD dataset, the depth map provides additional information about the scene geometry, which can be useful for tasks such as object recognition and pose estimation. For depth extraction form our Synthetic data we need to know the coordinates of object points in the camera system. Here we propose two methods.

Method 1, using 3D model

Step 1: To change the position and orientation of an object from the object coordinate system to the camera coordinate system, we extract the rotational and translational matrices from Unity for each frame. Then, we apply these matrices to the 3D model and compute the coordinates of the objects in the camera view.

$$P_c = R \cdot P_w + t \tag{12}$$

However, in our experience, this process may cause the object view to be slightly rotated. We observed this effect ob several views of our object of interest. It is illustrated in figure 9.

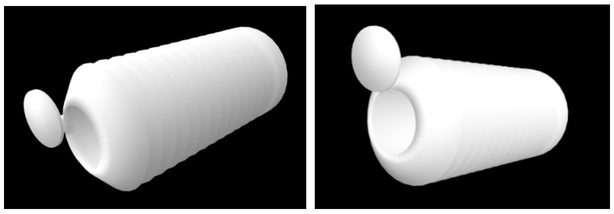

(a) original 3D (b) 3D after R and T matrix

Fig. 9: Comparision of the original 3D and after applying the R and T matrix

Step 2: The next step is the projection of the 3D point from camera coordinate system to the image plane. It has to be realized by the multiplication of P_c with the intrinsic parameters of camera matrix K

$$p_{2D} = K * P_c \tag{13}$$

Nevertheless, as we observed the errors in the transformation of coordinate systems from world system to camera system, this projection obviously led to errors. Hence, we could not follow this direct way to get corresponding 2D points with their depth. Therefore, we applied an Deep estimator of depth from a single rgb image, the MiDas system [].

Method 2, MiDaS

MiDaS (Multiple Depth Estimation Accuracy with Single Network) is a deep learning-based method for monocular depth estimation from a single image. It predicts the relative depth of objects in a scene, and the output is displayed using Matplotlib. MiDaS is based on an encoder-decoder architecture, where the encoder part is responsible for high-level feature extraction and the decoder generates the depth map from these features via up-sampling [19].

The input image is an RGB image in vector form (typically a tensor) and the output is a scalar depth value for each pixel.

- $I \in \mathbb{R}^{H\times W\times 3}$ as the input RGB image, where H is the height, W is the width, and 3 corresponds to the three color channels (Red, Green, Blue).

- $D \in \mathbb{R}^{H\times W}$ as the predicted depth map.

Let F represent the MiDaS model, typically a convolutional neural network (CNN). MiDaS predict the depth map D from the RGB image I using a neural network with parameters W:

$$D = F(I; W) \tag{14}$$

The preprocessing step transforms the RGB image I into a suitable input for the model:

$$I' = P(I) \tag{15}$$

This formulation assumes that D is a scalar depth value for each pixel in the input RGB image I. The final heatmap result is illustrated as figure 10. Notes that we have applied MiDas model without fine -tunning on our dataset, as we do not have ground truth depth maps.

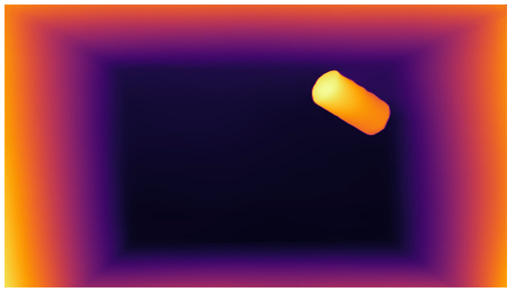


Fig. 10: Depth Image

*8) **Train and Test set:*** In our synthetics database, as previously stated, we currently have 2320 frames available, out of which 70% (1624) have been allocated for training our model for 6D pose estimation, while the remaining 30% (696) have been set aside for testing its accuracy.

V. Experimental evaluation

In this section, we report on the experiments conducted on both LineMOD dataset for the tests of the DenseFusion model and on the Synthetic dataset generated with the HW/SW setting from HYBRID team of Incia (Head set and Unity). Hence we will first describe the set-up of evaluated DenseFusion method,then present quality metics we used and finaly present the results on both datasets.

A. Settings

here we describe different parameter settings, evaluation metrics and datasets.

1) Setup of DenseFusion Method:

- Model Initialization
  The model iniatlization was done by loading the model architecture (both estimator and refiner) and their weights from file.
- Loss Function The loss function used is the distance between the points selected on the targets meshes of objects in LineMOD dataset transformed by real pose and commensurate points on the same mesh in the

estimated pose and is given below as:

$$L = \frac{1}{N} \sum_{i=1}^{N} \|T(p_i) - \hat{T}(p_i)\| \tag{16}$$

where:
L is the loss function.
N is the total number of points selected on the target meshes.
p_i is the i-th point selected on the target meshes.
$T(p_i)$ is the transformation of p_i by the real pose.
$\hat{T}(p_i)$ is the transformation of p_i by the estimated pose.
$\|.\|$ denotes the Euclidean norm.

2) Training Setup:

- Hyperparameter Configuration
  The hyperparameters providing the best performance are given below as:
  Learning Rate = 1e-5
  Batch Size = 8
  Number of Epoch = 200

  A lower learning rate value was chosen in order to prevent catastrophic forgetting (when model forgets what it has previously learned on the original task and quickly overwrites the weight values), and to help stabilize the fine-tuning process through a more gradual adjustment of the weights to the new task, preventing the model from the deviating from the previously learned features.
- Training Pipeline
  A new python script was created for the finetuning process which set hyperparameters based on the values provided earlier, loading the model architecture and weights, loaded both training and evaluation datasets, resumed the training process from the pretrained weights, and performed an evaluation (using average distance to be discussed later) after each epoch on the test set and saved the weights of the model if it has the best (lowest) distance value.

3) Evaluation Metrics: The quantitative metrics used include mean pixel reprojection error and Transformation MSE.

- Average pixel reprojection error

  This error is computed between pixel coordinates projected from 3D coordinates into the image plane and the ground truth pixel coordinates in the 2D image plane as the mean squared error (MSE):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \|p_{2d} - \hat{p}_{2d}\|^2 \tag{17}$$

  Here, p_{2d} are the 2D coordinates of ground truth bounding box in 2D image plane
  $\hat{p}_{2d}$ are the projected 2D coordinates of bounding box in the image plane
  n is the number of coordinates i.e 4 for a bounding box.
  Then the MSE per frame is averaged on the whole test data set of N samples:

$$\text{Average MSE} = \frac{1}{N} \sum_{i=1}^{N} MSE_i \tag{18}$$

- Transformation MSE (T_MSE)
  This samples a number m of points from the 3D model, changes their coordinate system from object coordinate system to the camera coordinate system using both the ground truth and predicted pose matrices and computes the Euclidean norm of the difference of the coordinates:

$$P'_{\text{GT}} = R_{\text{GT}}.P + T_{\text{GT}} \tag{19}$$

$$P'_{\text{pred}} = R_{\text{pred}}.P + T_{\text{pred}} \tag{20}$$

  where P'_GT and P'_{pred} are transformed 3D points with ground truth parameters and predicted values respectively, R_GT and R_pred: ground truth and predicted rotational matrices, T_GT, and T_pred : ground truth and predicted translational vectors, m is the number of points

$$\text{T\_MSE} = \frac{1}{m} \sum_{i=1}^{m} \|P'_{\text{GT}} - P'_{\text{pred}}\|^2 \tag{21}$$

  It is then averaged on N samples of the dataset.

$$\text{Average T\_MSE} = \frac{1}{N} \sum_{i=1}^{N} T\_MSE_i \tag{22}$$

- Correct Prediction Rate.
  This evaluation was performed during the first implementation (on the LineMOD dataset) and it involved comparing each transformation MSE against a ground-truth maximum error value provided in the LineMOD dataset, such that a correct prediction is obtained if the transformation MSE is less than the maximum error value Th, then the percentage of correct predictions is got for the whole dataset.

$$\text{Correct Prediction} = \text{T\_MSE} \leq \text{Th} \tag{23}$$

$$\text{Correct Prediction Rate} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100\% \tag{24}$$

4) Hardware and Software Solutions:

- Hardware Setup
  The workstation used for running the model and fine-tuning has the following specifications: 10 cores Xeon W-1290 64Gb CPU, having a RTX 3060 (12Gb) GPU.
- Software Setup
  The implementation was done on Linux OS, with the PyTorch Deep learning library, and other libraries including torchvision, tensorflow, fvcore, iopath, Pillow, scipy, numpy, pyyaml, cuda-python, matplotlib, opencv-python, trimesh, imgaug, pyyaml, wget, numpy, imutils, timm, plyfile, scipy, cffi, numpy-stl.

5) Compared Configuration of DenseFusion Method: This section highlights the evaluation results of the model trained on the LineMOD dataset compared against the results of the fine-tuned model (on the synthetic dataset)

- Pre-Trained Models
  The pretained models (on the LineMOD dataset) used include the pose estimator model (pose_model_) trained for 9 epochs, the pose refiner model (pose_refine_model_) trained for 493 epochs.
- Fine-tuned Models
  The above models were fine-tuned on the synthetic dataset for 200 epochs in an attempt to improve the model's performance on our dataset.

6) Datasets: As we stated in section IV in this work we used two datasets. The first one is *LineMOD* benchmark dataset [20], which contains RGB images with objects and available ground truth depth, while the second is the synthetic dataset which was compiled to match the directory hierarchy of the LineMOD dataset and its elements were obtained using the techniques described in IV-C. LineMOD datset comprises 1236 samples in one object id. The split of it was into 70/30% training and test sets respectively.

For synthetic dataset we used 2,320 samples in overall. The split was of 70% for training and 30% for testing.

B. LineMOD Experiment results

The results on the LineMOD dataset (without fine-tuning) are given below in table III

no of samples	mean	min	max	std
1483	589.238	35	100	75.47

TABLE III: Avegrage pixel reprojection error for pre-trained model applied to LineMOD dataset

The results of the Correct Prediction Rate evaluation with the optimal threshold value Th in IV proposed in [20] are provided in the table V below:

Object	Duck	Egg crate	Ketchup	Landline
Threshold	0.108	0.164	0.175	0.212

TABLE IV: Threshold Th values on LineMOD dataset

Object	Duck	Egg crate	Ketchup	Landline
Correct Pred Rate (%)	38.3.6	85.9	93.4	100

TABLE V: Correct Prediction rate evaluation results on LineMOD dataset

Transformation MSE: The transformation MSE per epoch over the 493 epochs of the lineMOD dataset is provided in figure 11 below.

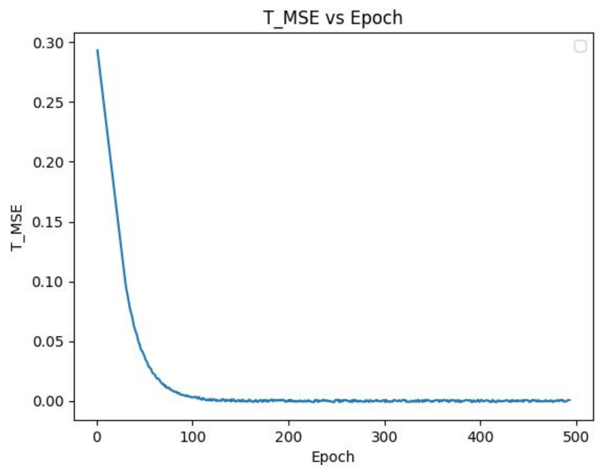


Fig. 11: Plot of Average Transformation MSE vs Epoch on a test set of LineMOD dataset

1) Average pixel reprojection error: The mean pixel reprojection error when using the DenseFusion model on the synthetic dataset trained for 200 epochs for both the estimator and refiner segments is provided in the table VI, while the values when using the pre-trained model only is provided in table VII
2) Transformation MSE: The transformation MSE per epoch over the 200 epochs of the fine-tuning process is provided in figure 12 below.

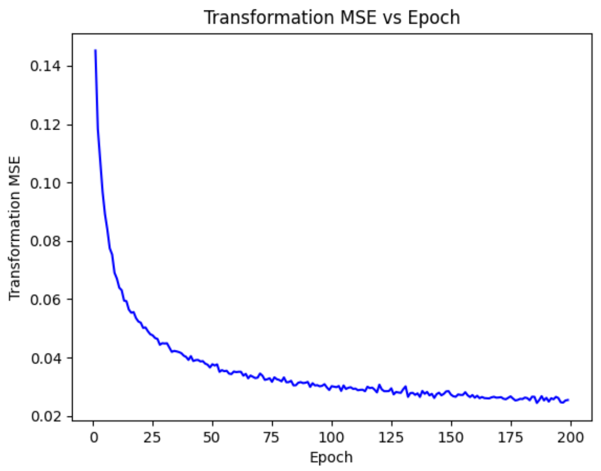


Fig. 12: Plot of Average Transformation MSE vs Epoch on a test set of synthetic dataset

no of samples	mean	min	max	std
696	4264.954	16	9976	2654.53

TABLE VI: Average pixel reprojection error for the model fine-tuned on a synthetic dataset.

C. Results on Synthetic Dataset

The results on the synthetic dataset are given below: First of all, we show the plot of Average Transformation MSE vs training epochs. It can be seen that with the increase in the number of epochs, the error decreases and stabilizes around 200 epochs.

3) Correct Prediction Rate The results of the Correct Prediction Rate evaluation with the optimal threshold value $Th = 0.537$ is provided in the table VIII below:

no of samples	mean	min	max	std
696	186560.6	1464	42632610	1956092.11

TABLE VII: Average pixel reprojection error for pre-trained model applied to Synthetic dataset

Object	Bottle
Correct Pred Rate (%)	60.2

TABLE VIII: Correct Prediction rate evaluation results on Synthetic dataset

One of the reasons the fine-tuned transformation MSE drops but still has a relatively high value even after fine-tuning, may be attributed to Noise / inaccuracies in the input data from estimating the depth values. This performance could also be due to the fact that the object in the scene lacks distinctive and easily detectable features, which could cause the model to have difficulty in accurately localizing and estimating its pose.

The study highlights the need to address depth estimation accuracy, scene complexity, overfitting, data quality, hyperparameters, data augmentation, and post-processing techniques for improved model performance on the synthetic dataset.

VI. Conclusions

We have analyzed various methods for estimating 6D pose from RGB-D images in this project. After considering the literature, we have chosen the DenseFusion method as it is one of the most powerful approaches. This technique merges a dense representation of features that includes both color and depth information based on the confidence of their predictions. We have tested the DenseFusion method on the LineMOD dataset, which was used by the authors of the original paper.

Since our project focuses on the development of a vision-aided prosthesis design, we created a synthetic dataset using the software developed by the HYBRID team at INCIA. Unlike the LineMOD dataset, this dataset does not contain depth information. Therefore, we developed two approaches to build this dataset to be used in the DenseFusion method. We tested the detector2 foundational model for object segmentation and MiDAS for molecular depth map estimation.

To estimate the depth in the synthetic dataset, we used the deep learning-based method for molecular depth estimation from a single image as we were unable to extract the depth from the 3D synthetic environment. As a result, we need to improve the quality metrics we have obtained when the real depth is extracted from the 3D model. We plan to collaborate with the HYBRID team INCIA to achieve this.

In the future, we aim to test the entire developed framework in an augmented reality scenario with a real-world cluttered environment. We will also propose adapted solutions for this scenario.

ACKNOWLEDGMENT

We are grateful to Professor Jenny Benois-Pineau for her unwavering commitment, invaluable guidance, and steadfast support throughout the project. Her expertise, dedication, and encouragement have inspired us to reach new heights. We would like to express our sincere gratitude to our TRDP I advisors, Dr. Nagy Zoltán and Pr. Szolgay Péter, for their invaluable guidance and support throughout the course of this project. We would also like to acknowledge the significant contribution of Dr. Attila Fejér, whose previous work laid the foundation for our project.

References

[1] European Council, ”Infographic - Disability in the EU: facts and figures,” consilium.europa.eu. https /www.consilium.europa.eueninfographicsdisabilityeufacts-figuresenergy (accessed Feb. 13, 2023).

[2] Wang, J., et al(2019). DenseFusion: 6D Object Pose Estimation by Iterative Dense Fusion.

[3] Alvaro Collet; Manuel Martinez; Siddhartha S. Srinivasa; ”The MOPED Framework: Object Recognition and Pose Estimation for Manipulation”, THE INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH, 2011. (IF: 7)

[4] Xingyi Zhou; Xiao Sun; Wei Zhang; Shuang Liang; Yichen Wei; ”Deep Kinematic Pose Regression”, ARXIV-CS.CV, 2016. (IF: 5)

[5] Georgios Pavlakos; Xiaowei Zhou; Aaron Chan; Konstantinos G. Derpanis; Kostas Daniilidis; ”6-DoF Object Pose From Semantic Keypoints”, ARXIV-CS.CV, 2017. (IF: 7)

[6] Yu Xiang; Tanner Schmidt; Venkatraman Narayanan; Dieter Fox; ”PoseCNN: A Convolutional Neural Network For 6D Object Pose Estimation In Cluttered Scenes”, ARXIV-CS.CV, 2017. (IF: 9)

[7] Thanh-Toan Do; Ming Cai; Trung Pham; Ian Reid; ”Deep-6DPose: Recovering 6D Object Pose From A Single RGB Image”, ARXIV-CS.CV, 2018. (IF: 4)

[8] Asako Kanezaki; Yasuyuki Matsushita; Yoshifumi Nishida; ”RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews From Unsupervised Viewpoints”, CVPR, 2018. (IF: 6)

[9] Tomas Hodan; Frank Michel; Eric Brachmann; Wadim Kehl; Anders GlentBuch; Dirk Kraft; Bertram Drost; Joel Vidal; Stephan Ihrke; Xenophon Zabulis; Caner Sahin; Fabian Manhardt; Federico Tombari; Tae-Kyun Kim; Jiri Matas; Carsten Rother; ”BOP: Benchmark For 6D Object Pose Estimation”, ECCV, 2018. (IF: 6)

[10] Guoguang Du; Kai Wang; Shiguo Lian; Kaiyong Zhao; ”Vision-based Robotic Grasping From Object Localization, Object Pose Estimation To Grasp Estimation For Parallel Grippers: A Review”, ARXIV-CS.RO, 2019. (IF: 4)

[11] Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017). PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. arXiv preprint arXiv:1612.00593

[12] Byambaa, M., Koutaki, G., & Choimaa, L. (2022). 6d pose estimation of transparent object from single rgb image for robotic manipulation. Ieee Access, 10, 114897114906. https:/doi.org10.1109access.2022.3217811

[13] Lee, G., Kim, J., Kim, S., & Kim, K. (2023). 6d object pose estimation using a particle filter with better initialization. Ieee Access, 11, 11451-11462. https://doi.org/10.1109/access.2023.3241250

[14] Periyasamy, A. (2022). Convposecnn2: prediction and refinement of dense 6d object poses.. https://doi.org/10.48550/arxiv.2205.11124

[15] Wang, C., Xu, D., Zhu, Y., Martín-Martín, R., Lu, C., Li, F., … & Savarese, S. (2019). Densefusion: 6d object pose estimation by iterative dense fusion.. https://doi.org/10.1109/cvpr.2019.00346

[16] Xiang, Y., Schmidt, T., Narayanan, V., & Fox, D. (2017). Posecnn: a convolutional neural network for 6d object pose estimation in cluttered scenes.. https://doi.org/10.48550/arxiv.1711.00199

[17] Xu, Y., Wang, L., Yang, A., & Chen, L. (2019). Graspcnn: real-time grasp detection using a new oriented diameter circle representation. Ieee Access, 7, 159322-159331. https://doi.org/10.1109/access.2019.2950535

[18] Yi, L., Gu, W., Ji, X., Xiang, Y., & Fox, D. (2019). Deepim: deep iterative matching for 6d pose estimation. International Journal of Computer Vision, 128(3), 657-678. https://doi.org/10.1007/s11263-019-01250-9

[19] Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler and Vladlen Koltun, Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer (2020), IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI).

[20] Hinterstoisser et al.: Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes, ACCV 2012

[21] Datasets BOP: Benchmark for 6D Object Pose Estimation. https://bop.felk.cvut.cz/datasets/.

[22] SINGLESHOTPOSE GitHub: Let's build from here. https://github.com/Microsoft/singleshotpose/blob/master/README.md.

[23] Unity Technologies. (2023). Unity® software [Software documentation]. Unity Technologies. https://docs.unity3d.com/Manual/VROverview.html

[24] Yuxin Wu and Alexander Kirillov and Francisco Massa and Wan-Yen Lo and Ross Girshick, dtectron2, 2019

[25] HTC VIVE. (2023). HTC VIVE — VR Headsets, Games, Accessories, and Metaverse Portal. Retrieved December 17, 2023, from https://www.vive.com/

[26] Jiang, H., et al. (2020). Efficient 6D Object Pose Estimation using Hierarchical Search. In Proceedings of the European Conference on Computer Vision (ECCV).

[27] Chen, J., et al. (2017). Efficient and Robust 6D Object Pose Estimation using Point Pair Features. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[28] Liu, X., et al. (2018). Robust 6D Object Pose Estimation via Semi-Supervised Learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[29] Zhang, Y., et al. (2021). PoseCNN++: Enhancing 6D Object Pose Estimation with Attention Mechanism. In Proceedings of the International Conference on Robotics and Automation (ICRA).

[30] Azeem, A., et al. (2020). PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. Retrieved from urlhttps://arxiv.org/abs/1711.00199.

[31] Rad, M., & Lepetit, V. (2018). BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth. Retrieved from urlhttps://arxiv.org/abs/1802.09232.

[32] Li, Y., et al. (2018). DeepIM: Deep Iterative Matching for 6D Pose Estimation. Retrieved from urlhttps://arxiv.org/abs/1804.00175.

[33] Brachmann, E., et al. (2016). DSAC++: Differentiable RANSAC for Camera Localization. Retrieved from urlhttps://arxiv.org/abs/1611.05705.

[34] Kae, A., et al. (2016). PoseRBPF: A Rao-Blackwellized Particle Filter for 6D Object Pose Tracking. Retrieved from urlhttps://arxiv.org/abs/1608.01981.

[35] Xu, D., et al. (2018). CPN: Cross Propagation Network for 6D Object Pose Estimation. Retrieved from urlhttps://arxiv.org/abs/1809.07761.

[36] Rad, M., & Lepetit, V. (2017). Real-Time Seamless Single Shot 6D Object Pose Prediction. Retrieved from urlhttps://arxiv.org/abs/1711.08848.

[37] Wohlhart, P., et al. (2017). Segmenting Unknown 3D Objects from Real Depth Images using Mask R-CNN Trained on Synthetic Data. Retrieved from urlhttps://arxiv.org/abs/1711.08704.

[38] Liang, M., et al. (2019). Deep Continuous Fusion for Multi-Sensor 3D Object Detection. Retrieved from urlhttps://arxiv.org/abs/1907.04829.

[39] Brachmann, E., et al. (2020). Learning to Find Good Correspondences for Faster R-CNN-Based 6D Object Pose Estimation. Retrieved from urlhttps://arxiv.org/abs/2001.10254.

[40] Zhang, Y., et al. (2019). Method for Fast and Accurate 6D Pose Estimation in Cluttered Scenes. Retrieved from urlhttps://arxiv.org/abs/1901.05709.

[41] Wang, L., et al. (2020). Attention-based Model for 6D Object Pose Estimation. Retrieved from urlhttps://arxiv.org/abs/2002.09275.

[42] Chen, J., et al. (2021). Deep Fusion of Geometric and Appearance Features for 6D Object Pose Estimation. Retrieved from urlhttps://arxiv.org/abs/2103.08742.

[43] Liu, X., et al. (2022). Learning Discriminative Features for 6D Object Pose Estimation. Retrieved from urlhttps://arxiv.org/abs/2204.03654.

[44] Wang, T., et al. (2023). Joint Object Detection and Pose Estimation using Point Clouds. Retrieved from urlhttps://arxiv.org/abs/2302.08456.

[45] Zhang, Y., et al. (2022). Learning Pose Consistency for 6D Object Pose Estimation. Retrieved from urlhttps://arxiv.org/abs/2207.09123.

[46] Li, X., et al. (2023). Learning Geometric and Semantic Context for 6D Object Pose Estimation. Retrieved from urlhttps://arxiv.org/abs/2304.05987.

[47] Wang, R., et al. (2022). Deep Reinforcement Learning for Active Object Pose Estimation. Retrieved from urlhttps://arxiv.org/abs/2206.07891.

[48] Fejér, A.; Nagy, Z.; Benois-Pineau, J.; Szolgay, P.; de Rugy, A.; Domenger, J.-P. Hybrid FPGA–CPU-Based Architecture for Object Recognition in Visual Servoing of Arm Prosthesis. J. Imaging 2022, 8, 44. https://doi.org/10.3390/jimaging8020044