5-2022

DeepRM: Deep Recurrent Matching for 6D Pose Refinement

Alexander Avery
aja9675@rit.edu

DeepRM: Deep Recurrent Matching for 6D Pose Refinement

Alexander Avery

DeepRM: Deep Recurrent Matching for 6D Pose Refinement

Alexander Avery

May 2022

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

RIT | Kate Gleason College of Engineering

Department of Computer Engineering

DeepRM: Deep Recurrent Matching for 6D Pose Refinement

Alexander Avery

Committee Approval:

Andreas Savakis *Advisor* Date
Department of Computer Engineering

Dongfang Liu Date
Department of Computer Engineering

Clark Hochgraf Date
Department of Electrical and Computer Engineering Technology

Acknowledgments

I would like to extend a special thanks to my advisor, Dr. Andreas Savakis, for all of his support and guidance throughout my academic studies. I would also like to thank Dr. Dongfang Liu and Dr. Clark Hochgraf for participating on my thesis committee. I am grateful to Dr. Raymond Ptucha, who inspired me to return to RIT for my masters degree. I would like to thank the Vision and Image Processing lab group, in particular the Ph.D. students, Bruno, Abu, and Navya for their sharing their experience. And finally, I would like to thank my friends and family, in particular my wife Kira, for their constant love and support.

To Kira, Pepper, Mom, and Dad, whom I could not have done this without.

Abstract

Precise 6D pose estimation of rigid objects from RGB images is a critical but challenging task in robotics and augmented reality. To address this problem, we propose DeepRM, a novel recurrent network architecture for 6D pose refinement. DeepRM leverages initial coarse pose estimates to render synthetic images of target objects. The rendered images are then matched with the observed images to predict a rigid transform for updating the previous pose estimate. This process is repeated to incrementally refine the estimate at each iteration. LSTM units are used to propagate information through each refinement step, significantly improving overall performance. In contrast to many 2-stage Perspective-n-Point based solutions, DeepRM is trained end-to-end, and uses a scalable backbone that can be tuned via a single parameter for accuracy and efficiency. During training, a multi-scale optical flow head is added to predict the optical flow between the observed and synthetic images. Optical flow prediction stabilizes the training process, and enforces the learning of features that are relevant to the task of pose estimation. Our results demonstrate that DeepRM achieves state-of-the-art performance on two widely accepted challenging datasets.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Recent advancements in the fields of computer vision and deep learning, paired with advancements in computational hardware, have unleashed an unprecedented number of new technologies and new techniques that are applied in virtually all industries. Specific to the field of computer vision, Convolutional Neural Networks (CNNs) have enabled breakthroughs in many visual tasks such as object detection, segmentation, and pose estimation. Although CNNs typically operate on 2D images and features, in this work we will extend this functionality to perform reasoning in 3D, and estimate the fully constrained 6 dimensional (6D) pose of rigid objects from RGB images.

Detecting objects and estimating their 6 dimensional pose (x, y, z, roll, pitch, yaw) in 3D space is a fundamental task in the field of computer vision. As such, it has many applications, the most common of which is robotic manipulation. For a robot to be able to effectively interact with an object, it must know the object's pose in relation to itself. In the case of robotic grasping, the object's position is used to determine the input to the inverse kinematic solver, which can then calculate the joint states necessary to grasp the object. Augmented Reality (AR) has also recently become a popular field requiring very precise pose estimation [3]. In this setting, pose estimation enables humans to interact with both physical and virtual

objects in a seamless manner. Applications range across industries such as healthcare, manufacturing, education, and gaming.

Although 6D pose estimation of rigid objects is a well studied problem, the majority of research has focused on a relatively low level of accuracy. This was driven by its main application of robotic grasping and manipulation. To successfully grasp an object, the estimate of the object's pose does not need to be extremely precise. The universally accepted metric for success is based on an an acceptable error magnitude of 10% of the target object's diameter [2, 4, 5, 6, 1, 7, 8]. While this may be sufficient for robotic grasping, it does not meet the requirements for other tasks such as precision assembly and augmented reality. Human to object interaction in augmented reality is particularly difficult due to the registration problem. This is the problem of aligning the virtual world with the real world, in a way that's seamless to the end user. Studies have shown that humans are very sensitive to this problem, and can detect very small amounts of error. Because of this, augmented reality technology has acceptable error tolerances in range of millimeters, fractions of degrees, and single pixels, depending on the target application [3].

Early breakthroughs [9, 10, 11] in the field of 6D pose estimation leveraged RGB-D data to address the shortcomings of RGB only data. Limitations with RGB only data are due to factors such as lack of visual cues in textureless regions, and ambiguity of observed object size due to variations in depth. Depth information also helps to detect partial occlusions, and distinguish between background and foreground. Given the numerous benefits of depth information, the most accurate 6D pose solutions today still utilize RGB-D data [12, 13, 14, 15]. RGB-D sensing technologies however, come with their own set of limitations and constraints. RGB-D sensors are typically much more expensive, require more power, have larger form factors, lower resolution, lower frame rates, and are more sensitive to external factors such as sunlight than typical RGB only sensors [1, 16, 17]. Furthermore, recent advancements in computer vision

and AI are enabling RGB only solutions to approach the same levels of accuracy as RGB-D. CosyPose [7] for example beat all but two other approaches in the 2020 BOP Challenge on 6D Object Localization [15] using RGB only data, whereas the top two and many others used RGB-D. Our intention in this work is to further close the gap between RGB and RGB-D approaches by focusing on processing RGB only data. By limiting the necessary information to RGB only, this enables our solution to be used across a wider range of applications where RGB-D data capture is not practical.

Given that maximizing accuracy is the main goal of this approach, it will specifically focus on 6D pose refinement via a convolutional neural network. As such, we will assume that detections and initial coarse pose estimates of objects are readily available from existing solutions. This approach has been followed by other CNN based refinement techniques such as [1, 18, 19], where initial poses are generated by PoseCNN [2], SSD-6D [4], and PVNet [6], respectively. Inspired by these same works, we also treat the refinement problem as an iterative process. Given an RGB image containing an object of interest, and an initial pose estimate, our network estimates the translation and rotation necessary to correct the initial estimate such that it matches the observed pose in the input image. It accomplishes this via a render and compare approach. At each iteration, the object is rendered with the latest pose estimate. The rendered and observed images are then passed to a CNN, which directly regresses the rotation and translation updates. For simplicity, the process is repeated for a fixed number of iterations during both training and testing.

Recurrent Neural Networks (RNNs) are powerful models that are typically used to process temporal sequences of data. RNNs are an extension to standard feed forward neural networks that add a memory component to the network. In the case of a vanilla RNN, the output of each neuron is recorded. At the following time step, it is both the previous output, as well as the standard feed forward input, that is used to determine the next output of the neuron. This memory mechanism allows the

network to perform tasks which require knowledge of past states to make effective predictions. Usefulness of this technique spans across many domains such as audio, video, financial data, weather, and many others. The vanilla RNN however has a weakness in that can not effectively model long sequences of data due to its simple feedback mechanism. To address this limitation, a Long Short Term Memory (LSTM) [20] architecture was introduced. LSTM units add additional internal states, as well as internal gates, to control the flow of data through them. The gating mechanism allows the cell to selectively retain or discard information at each time step. By allowing the cells to selectively gate data for an arbitrary number of time steps, both short term and long term dependencies are able to be captured. While introduced in 1997, LSTMs are still widely used today in many state-of-the-art works.

The proposed framework, called Deep Recurrent Matching or DeepRM, introduces a novel architecture which augments a convolutional neural network with recurrent neurons in the form of LSTM units to achieve state-of-the-art results on the task of 6D pose refinement.

1.2 Motivation

The application of computer vision and artificial intelligence to the field of robotics has enabled robots to perform more complex tasks and operate in less structured environments than ever before. Robots are no longer limited to performing a fixed set of motions on a manufacturing line. Nowadays, robots can be found performing intelligent operations on on farms, in space, on roads, underwater and many other areas. This intelligent operation has allowed robots to replace humans in a variety of jobs that are unsafe, repetitive, or prone to human error. In many of these jobs, the task of 6D pose estimation is integral in enabling effective interaction with the environment. By improving both the accuracy and reliability of this task, we are able to continue expanding the areas where it can be applied. Furthermore, this

technique can also be applied to virtual reality applications, where the highest degree of accuracy is necessary to provide a seamless experience to the user.

1.3 Contributions

The main contributions of this thesis are as follows:

- A novel architecture, called Deep Recurrent Matching or DeepRM, which focuses on leveraging recurrent units to improve the iterative process of 6D pose refinement.
- DeepRM is the first scalable architecture designed for 6D pose refinement
- DeepRM achieves state-of-the-art results on the challenging YCB-Video [2] and Occlusion LINEMOD [10] datasets

1.4 Document Structure

The remainder of this document is structured as follows: Chapter 2 covers the background material and an overview of the related works in the fields of 6D object pose estimation, and 6D pose refinement. Chapter 3 provides a detailed description of the proposed DeepRM method. Chapter 4 describes the implementation of our method. Chapter 5 presents our results compared to the current state-of-the-art, as well as a variety of ablation studies on different components of our method. Finally, Chapter 7 provides our concluding remarks, and discusses possible future work.

Chapter 2

Background

2.1 Multi-Layer Perceptron

Artificial Neural Networks (ANNs) are a subcategory within the fields of artificial intelligence, and machine learning. The most common and basic type of ANN is the Multi-Layer Perceptron (MLP). The intention of this type of network is to replicate the structure and the behavior of the human brain. As such, the most basic building block of these networks is called the neuron, or perceptron. A neuron is typically characterized by four main components: inputs, weights, bias, and output. The scalar output of a neuron is determined by first multiplying the inputs by the weights, and then adding the bias. The output is also typically processed by an activation function, which adds a non-linearity to the response. The non-linear activation function is what differentiates this process from linear regression, and allows the network to model extremely complex representations of data. To form a MLP network out of neurons, neurons are first stacked into layers. These layers are then also stacked to form a 2D network of neurons. MLPs are generally fully connected, which means that all neurons in one layer are connected to all of the neurons in the following layer. The number of layers, and the number of neurons in each layer is arbitrary, except that there must be at least three layers to fit to data that is not linearly separable. Layers in-between the input and output are called hidden layers, and the more hidden layers

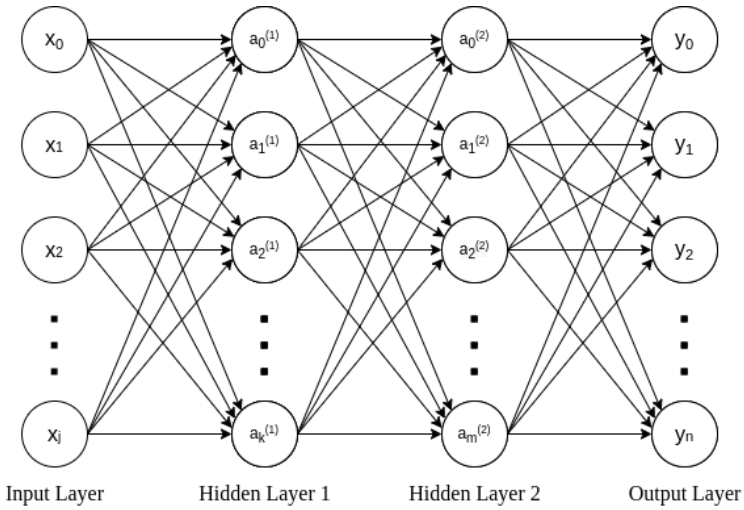

Figure 2.1: Example multi-layer perceptron (MLP) network with 2 hidden layers.

present, the deeper the network is considered. Figure 2.1 displays an MLP with two hidden layers, where arrows represent the connections between neurons.

2.2 Long Short Term Memory

As previously discussed, Long Short Term Memory (LSTM) [20] units are an extension to the standard MLP network structure that add a memory component to each neuron. This mechanism allows the network to perform tasks which require knowledge of past states to make effective predictions on present data. LSTM units achieve this by using internal states, as well as multiple internal gates, to control the flow of data through them. The internal states of the neuron are known as the cell state and the hidden state, while the gating mechanism consists of three separate gates: the forget gate, the input gate, and the output gate. Figure 2.2 illustrates the configuration of the 3-stage gating mechanism. Gates are realized via the sigmoid activation function, as its output is constrained between zero and one. This allows each gate to determine the percentage of information to pass through. Input to the LSTM consists of the current observation x_t, the previous cell state c_t, and the previous hidden state h_t, where h_t is simply the previous output of the unit. At each time step, x_t

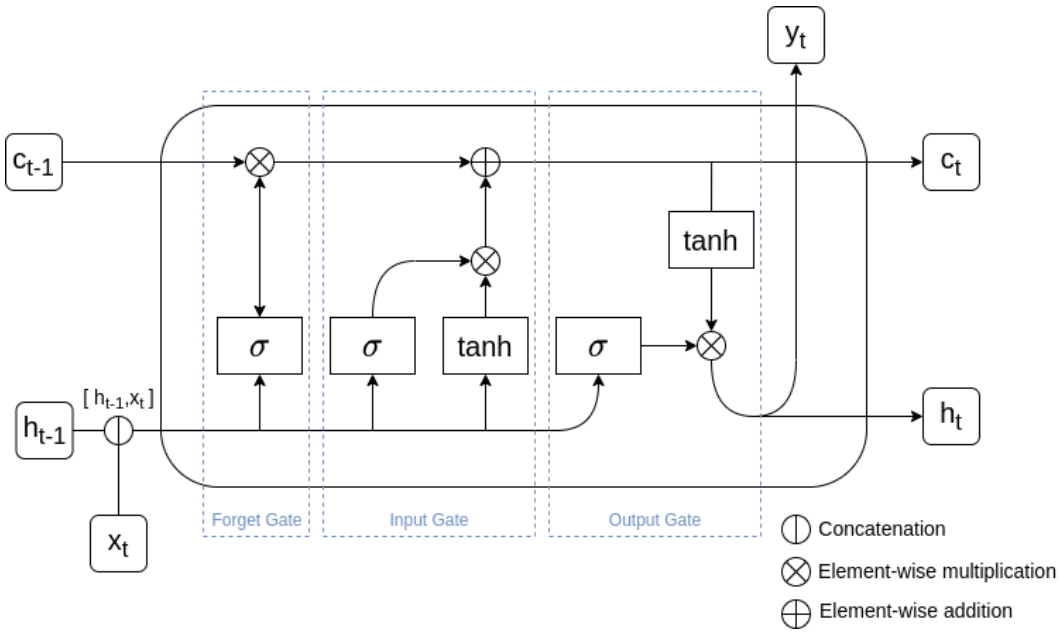

Figure 2.2: Illustration of an LSTM module.

and h_t are concatenated to form the input to the various gates. The forget gate then determines how much of the previous cell state to carry forward, based on the current input. As the output of the sigmoid operation is between zero and one, the update to the cell state is simply an element-wise multiplication. Similarly, the input gate determines what parts of the input should be added to the internal cell state. It does so by generating a vector of candidate values, and then filtering the candidates with a separate sigmoid operation. The filtered candidates are combined with the internal cell state via element-wise addition. Finally, the output of the LSTM unit is formed by once again gating the filtered cell state, based on a sigmoid activation of the input vector. Although complex, LSTMs are an extremely powerful learning mechanism, and are crucial to the success of countless techniques that require the processing of sequential data.

2.2.1 Gated Recurrent Units

To address the complexity of LSTMs, Gated Recurrent Units (GRUs) [21] were created as a simplified version of them. Instead of having forget, update, and output

gates, GRUs have a reset and an update gate. GRUs also eliminate the internal cell state, and only carry forward the hidden state. Like LSTMs, the sigmoid activation function is used to realize the gating mechanism. By reducing the number of gates and eliminating the internal cell state, the number of parameters and computational complexity is reduced significantly. Although simplified, GRUs are still able to model complex time-dependent relationships. This makes them suitable for use in resource constrained scenarios, where LSTM units may be prohibitive.

2.3 Convolutional Neural Networks

Nowadays, Convolutional Neural Networks (CNNs) are an extremely commonly used neural network architecture for processing imagery data. These networks were originally inspired by the visual cortex of the human brain. In 1959, Hubel et al. [22] discovered the existence if "simple" and "complex" cells in the human visual cortex. The purpose of simple cells is to detect primitive features such as edges and bars of various orientations. These features are commonly known as Gabor filters in the field of image processing. Complex cells then combine this information over various receptive fields to form a deeper and more complex representation of the observed data. This same functionality forms the basis for modern convolutional neural networks.

A typical CNN consists of convolutional layers, pooling layers, activation layers, and fully connected (MLP) layers. The convolutional layers apply a set of filters over the 2D image using a sliding window approach. This operation is a standard 2D convolution, where the weights of the filters are learned through backpropagation. The extracted information has a height, width, and channel dimension, and is called a feature map. The number of filters applied determines the channel dimension of the resulting feature map. Similar to the human vision system, these layers are stacked on top of each other, where each layer extracts a more condensed and complex representation of the observed data. To achieve such a condensed representation, pooling

layers are used between convolutional layers to reduce the spatial dimensionality of the data. This significantly reduces the overall computational requirements of the network. Typical pooling techniques include block averaging (AvgPool) as well as maximum pooling (MaxPool). Alternatively, instead of pooling, a stride greater than one can be used during convolution to reduce the resulting feature maps' spatial dimensions. This technique is becoming more popular than traditional pooling in many recent works. As with MLPs, non-linear activation functions are also used throughout convolutional neural networks. This enables the network to learn highly non-linear relationships and representations of data. Activations are commonly found after the convolution or pooling operations, but in certain cases can be omitted. While fully-convolutional networks exist, the traditional approach for tasks such as image classification is to stop downsampling at a certain level, and flatten the final feature map. The flattened feature map is then passed through a set of fully connected layers to determine the final output. VGG-Net [23] for example flattens the final 7x7x512 feature map, and passes this through three fully connected layers of sizes 4096, 1000, and 1000, followed by a softmax layer to perform the task of image classification.

2.3.1 EfficientNet

Compared to fully connected layers, convolutional layers are highly efficient in terms of the number of parameters required. This is because the convolutional filter weights are shared across the entire input. Weight sharing also provides some amount of spatial invariance, resulting in better generalization capabilities. However, while CNNs are known for their efficiency compared to other techniques, standard architectures such as AlexNet [24] still require tens of millions of parameters for the simple task of image classification. This limits the use of CNNs in resource constrained scenarios such as mobile phone applications. To address this limitation, many works have focused on making CNN architectures more efficient [25, 26, 27, 28, 29]. An exam-

ple of one such work is EfficientNet [25]. EfficientNet is a very popular and widely used architecture due to its scalability and efficiency. EfficientNet achieves its high efficiency via the use of inverted bottleneck layers, called MBConv layers [29], which were first introduced by Sandler et al. in MobileNetV2 [26]. The inverted bottleneck is a modification to the widely used residual block [30], except instead of using a wide-narrow-wide, structure, it uses a narrow-wide-narrow one. This decreases the width of bottleneck points in the network, significantly reducing the total amount of memory that must be allocated concurrently. At the same time, it maintains the desired ability of gradients to flow through the residual connections, addressing the problem of vanishing gradients. EfficientNet additionally leverages squeeze-and-excitation blocks [27] to further improve performance while balancing computational complexity. Squeeze-and-excitation blocks are a simple channel-wise attention mechanism, realized by global average pooling, 1x1 convolution, and element-wise multiplication. For scalability, the EfficientNet framework provides a method to scale up input resolution, width, and depth of a baseline network using a single scaling factor, ϕ. Optimal parameters were obtained in [25] via a neural architecture search that balanced accuracy with FLOPS, resulting in a discrete set of models for ϕ 0-7. These models are known as EfficientNet-B0 through EfficientNet-B7, and range from 5.3M to 66M parameters, respectively.

2.4 Optical Flow

Optical flow is defined as the perceived motion of individual pixels on an image sensor across consecutive image frames. Traditional methods assume a brightness constancy constraint in the case of grayscale images, or a color constancy constraint in the case of RGB images. Once the consistency across frames is assumed, the motion of unique pixels can be computed. Optical flow is capable of capturing both the position and the velocity of objects in the observed 3D scene, or similarly, the motion of the image

sensor itself. Due to the projection onto the image plane, the resulting information is captured in a 2D representation. This information is generally encoded into an image like format, where the resolution is the same as the input images, but has two channels for the x and y components of the optical flow vectors. Due to its ability to capture and represent motion in a compact form, there are many processing tasks in which optical flow is used as a prior. These tasks include: motion detection, object tracking, segmentation [31], and visual odometry [32]. Furthermore, variations of the optical flow technique can be used to accomplish tasks such as stereo disparity [33], and 6D object pose refinement [1]. In the case of 6D pose refinement, Li et al. compute the optical flow between a real image of an object and a rendered image of an object, as opposed to two real images taken in a sequence. In this scenario, each scene can be considered static, so the only observed optical flow is due to the relative displacement of the rendered object to the observed. As such, this information can be used to refine the rendered object's pose such that it more closely matches the target.

2.5 Synthetic Data for 6D Pose

One of the main challenges in the field of 6D pose estimation is generating large scale datasets containing accurate ground truth poses. It is very difficult, time consuming, and costly to generate this data [34]. As such, many datasets for 6D pose estimation are orders of magnitude smaller than the standard image classification datasets such as ImageNet [35] and MS COCO [36]. Due to this limitation, virtually all state-of-the-art 6D pose methods leverage synthetic data generation to augment their training data [15]. Using this technique, extremely precise 6D pose annotations can be procedurally generated with ease. However, while synthetic data successfully addresses the lack of sufficient training samples, another problem arises: the synthetic-to-real domain gap. Datasets which are then dominated by synthetic training samples can

lead to poor performance in the real world. To address the synthetic-to-real domain gap, two main approaches are used: generating more photo-realistic renders, and the use of domain randomizaion. For the 2020 BOP Challenge on 6D Object Localization [15], the BlenderProc4BOP renderer was publically released. BlenderProc4BOP is a light-weight physically-based renderer (PBR), capable of generating more photo-realistic images than the simple OpenGL based renderers commonly used at the time. PBR achieves high photorealism through ray tracing, as opposed to rasterization as in OpenGL. Ray tracing enables a more accurate simulation of complex illumination effects, such as scattering, refraction, and reflection. By making the synthetic data more realistic, the size of the domain gap is decreased, improving real world performance. The main downside to ray tracing is its computational requirements. On a modern GPU, BlenderProc4BOP takes 1-3 seconds to render a 640x480 RGB-D image [15]. This prohibits its use during the training process. Because of this limitation, we leverage pre-existing, publicly available, PBR generated synthetic data in this work, rather than rendering it ourself.

Domain randomization [37] is another powerful technique that attempts to bridge the synthetic-to-real domain gap. Domain randomization involves randomizing the synthetic training data such that when the model sees the real world during testing, it appears as simply another perturbation of the random training data. By adding this randomness to the training process, the model learns to generalize better to previously unseen environments. In the context of 6D pose estimation, this usually consists of rendering objects on top of random backgrounds, or replacing the background of real images. Random backgrounds typically consist of solid colors, textures, 3D environments, or images from publically available datasets like MS COCO [36]. Works such as [38] demonstrate that the combination of domain randomization with photo-realistic rendering can achieve results comparable to state-of-the-art, using only synthetic data. Figure 2.3 provides an example of such images used in this work.

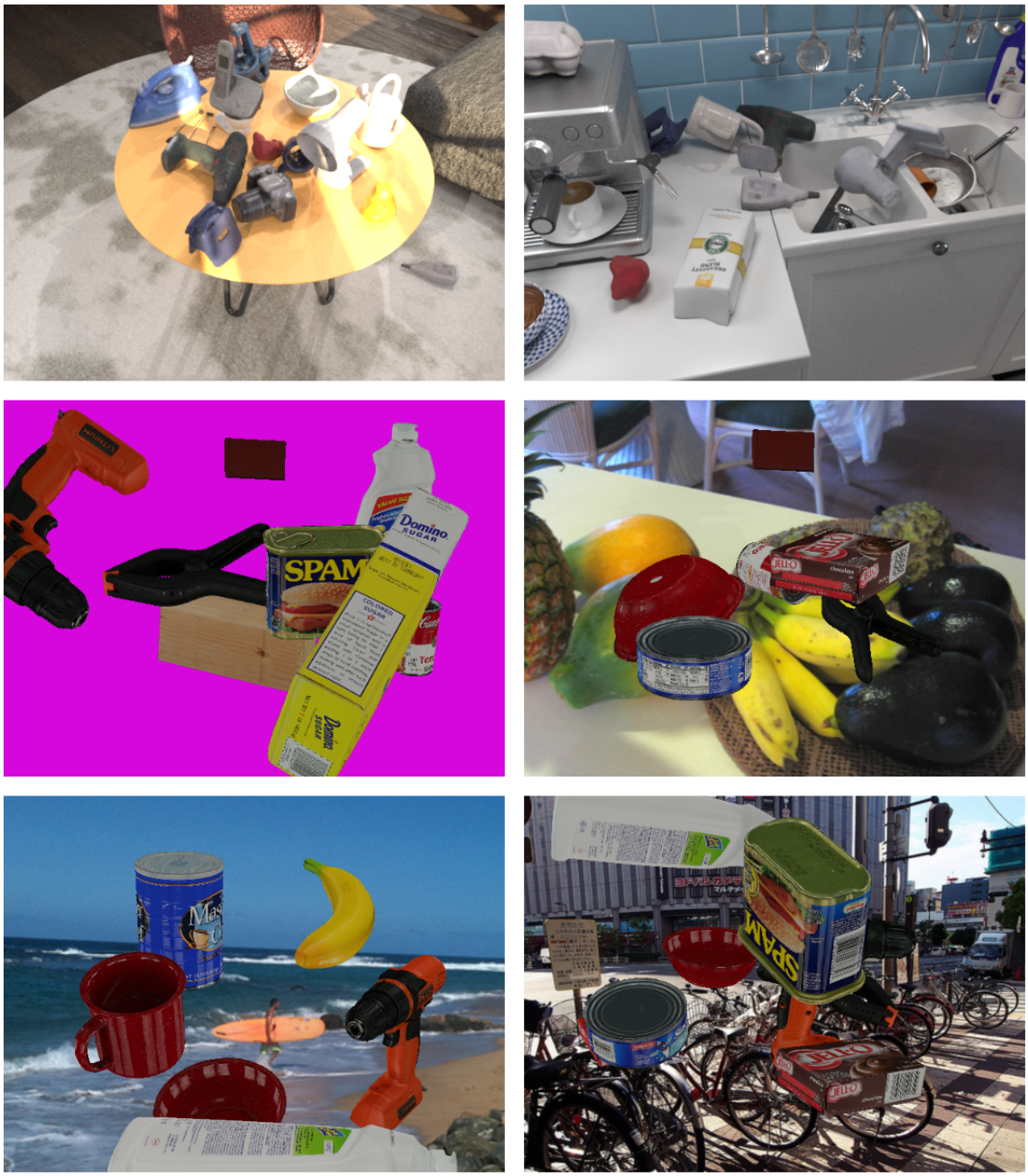

Figure 2.3: Examples of synthetic data. The top row contains photo-realistic PBR renders. Note the the accurate lighting effects and reflective surfaces. The middle and bottom rows contain examples of domain randomized images.

2.6 Related Work

While this work focuses specifically on the task of pose refinement, it is heavily dependent on the initial task of 6D pose estimation. Because of this, we will first introduce the task of 6D pose estimation, its challenges, and an overview of the related work in this field. We will then introduce the task of pose refinement, and provide a detailed analysis of related works. Due to the vastness of this field of study, we will focus solely on RGB based methods for both tasks.

2.6.1 6D Object Pose Estimation

The purpose of 6D object pose estimation is to determine an object's fully constrained pose within 3D space. In addition to the 3D translational components, the orientation of the object must also be determined, imposing an additional 3 axis of freedom. Combined, this results in a total of 6 degrees of freedom. Extracting such information from a single 2-dimensional RGB image is an ill-posed and very challenging problem. This is largely in part to the loss of dimensionality when projecting the 3D scene onto the 2D image sensor. After the projection, we can no longer disambiguate the size of observed objects with variations in depth. To compensate for this phenomenon, RGB-based methods generally require pre-existing geometric information of target objects, such as 3D CAD models. With the actual size of an object known in 3D, we can then recover depth information based on the observed size in pixels, along with the camera intrinsics. This however, is not the only challenging problem. In addition to the standard issues with RGB images such as proper exposure, focus or motion blur, noise, and color accuracy, we also encounter challenges with partial occlusions, lack of visual features on textureless objects, and pose ambiguity due to object symmetries. The difficulty of this task and the variety of challenges has led to a diverse set of solutions. The majority of these solutions can be roughly grouped into three main

categories: keypoint based methods, dense correspondence based methods, and direct methods.

2.6.1.1 Keypoint Based Methods

Keypoint based methods for pose estimation divide the problem into two main parts: the correspondence problem, and the Perspective-n-Point (PnP) problem. First, a point to point correspondence must be made that relates points in the 2D image space to 3D points on the object of interest. Then, for a calibrated camera, solving for pose given the 2D to 3D correspondences can be formulated into a minimization problem, known as the PnP problem. Solving the PnP problem is typically accomplished by minimizing the reprojection error of the 3D model points to their 2D pixel space projection. This error is a function of the pose of the object. Therefore, by minimizing it with respect to the object's rotation and translation, 6D object pose can be estimated. Since this is a traditional minimization problem, there are many ways with which to solve it. In the field of machine learning, an important factor in choosing a solution is whether or not it is differentiable. Traditional methods favored non-differentiable PnP solvers and focused on solving the correspondence problem. However, recent works aim to use differentiable methods to enable complete end-to-end training.

Early methods such as MOPED [39] utilized SIFT [40] and SURF [41] feature extraction techniques to select keypoints that could be matched with their corresponding locations on the 3D models. Then, given an input image, local features could be matched to 3D model points, and the PnP problem could be solved. Although SIFT and SURF features are designed to be highly discriminative and locally invariant, this method still suffers from mismatched correspondences. To mitigate this issue, robust statistical estimation techniques such as M-estimators or RANSAC [42] must be used on the correspondences. By leveraging invariant local features and

powerful statistical estimation techniques, these methods are fairly robust to partial occlusion. They are however not capable of handling textureless objects, due to their lack of discriminative features.

With the recent advancements in the field of machine learning, particularly the widespread use of CNNs, virtually all novel techniques replace traditional feature descriptors with learned ones. Two-stage keypoint based methods following this approach can be further be split into two types: those that predict the 8 corners of the target object's 3D bounding box [5, 43, 44, 38], and those that predict keypoints on the model's surface [6, 45, 46]. As the first convolutional neural network based solution, BB8 [43] uses a VGG [23] inspired network architecture to first coarsely segment a target object, and then pass a cropped image to a separate network to predict the 2D projections of the 3D bounding box. Once the 2D-3D bounding box correspondences are made, object pose is estimated by solving the PnP problem. A non-differentiable PnP solver is used, preventing end-to-end training of the network. Because of this, the network is trained to minimize the 2D projection loss of the 3D bounding box, as opposed to directly minimizing a loss value that is directly related to the pose. Following BB8, Tekin et al. [5] also predict the 2D projections of the 3D bounding box, however they adopt the highly efficient YOLO [47] architecture to directly predict the projections in a single shot. This method is called YOLO-6D, and was the first method to provide results that were accurate enough that they didn't require additional refinement for certain applications. BB8 [43] and YOLO-6D [5] inspired many derivative works [44, 48, 49, 27], where these works aimed to solve a variety of challenges such as occlusion, truncation, and lack of diverse features.

To address the problems of occlusion and truncation, PVNet [6] introduces a pixel-wise voting network, where each pixel predicts a unit vector pointing to the keypoints. Pixels vote for keypoint locations using RANSAC, resulting in an estimator that is still capable of detecting keypoints, even when they aren't visible due to occlusion or

truncation. PVNet additionally adopts the approach of using keypoints on the model surface, as opposed to the bounding box corners, to achieve a much lower variation in the predictions. Although PVNet moved the keypoints to the object surface, best results were still achieved with 8 keypoints, similar to methods using bounding boxes. To address the lack of diverse or distinguishable features, other works such as HybridPose [45] utilize a hybrid intermediate representation to incorporate additional geometric feature information. In addition to the typical keypoints, these features include edge vectors and symmetry correspondences. Leveraging a more diverse feature set enables improved performance under occlusion, but requires additional computational overhead to both detect the features, and to filter outliers.

2.6.1.2 Dense Correspondence Based Methods

Dense correspondence based methods are typically an extension of the keypoint based methods. Instead of predicting a sparse set of keypoints, dense methods aim to predict 3D coordinates for every pixel in the target image. By drastically increasing the number of 2D-3D correspondences, performance is maintained even under high levels of occlusion. To handle the additional noise inherent to the dense predictions, PnP+RANSAC is also used by most methods to achieve robustness to outliers. A few examples of dense correspondence methods are DPOD [50], Pix2Pose [51], CDPN [1], and EPOS [15]. To achieve the dense predictions, encoder-decoder style networks are typically used.

2.6.1.3 Direct Methods

In contrast to two-stage keypoint based approaches, direct methods are formulated such that they directly output object pose from an input image. Traditional methods accomplish this via template matching, however this approach is sensitive to a variety of factors such as occlusion, background clutter, illumination, and changes in

scale [1, 6]. Again following the trend in machine learning, recent works employ techniques such as convolutional neural networks and mutli-layer perceptrons (MLPs) to directly predict 6D pose from a single RGB input image. PoseCNN [2] and SSD-6D [4] were pioneers in this area, introducing the first methods following this approach. PoseCNN uses the VGG16 [23] network as a backbone to extract high dimensional feature maps. These shared feature maps are then utilized by three downstream tasks: semantic segmentation to localize and distinguish objects, translation prediction, and rotation prediction. To predict translation, PoseCNN uses a pixel-wise voting approach, similar to PVNet [6] to predict the object centerpoint in 2D. Along with the centerpoint location, the network also predicts the centerpoint depth. Once depth is estimated, the 3D translation can be recovered according to the thin lens equation (for a calibrated camera). Separately, rotation is predicted by directly regressing the quaternion representation of the rotation. This is accomplished by flattening a region-of-interest (RoI) of the final feature map, corresponding to the target object, and passing this vector through a 4096-4096-4 fully-connected MLP. PoseCNN also introduces a novel loss function, ShapeMatch-Loss, which addresses pose ambiguity due to object symmetries, by utilizing an Iterative-Closest-Point (ICP) like calculation. Under this formulation, the shapes of objects are aligned, rather than forcing a one-to-one mapping between the predicted and ground truth pose. The major downside to this approach is its $O(N^2)$ complexity.

Unlike PoseCNN, SSD-6D [4] modifies a much more lightweight backbone, SSD [52], to solve for 6D pose. Rather than direct regression of continuous values for rotation and translation, SSD-6D discretizes the camera viewpoint space and treats the prediction as a classification problem. While this network is very fast by itself, the inherent coarseness due to discretization requires a separate post-refinement step to achieve accuracy comparable to competing techniques. A more recent approach, EfficientPose [53], achieves both high accuracy and high efficiency by leveraging the

state-of-the-art EfficientDet [54] backbone architecture. Bukschat et al. [53] extend the EfficientDet architecture to solve for pose by simply adding additional rotation and translation subnetworks to the existing ones for class and bounding box prediction. Furthermore, the EfficientDet architecture is also designed to be scalable based on computational constraints, making it ideal for use in real world applications.

While many direct methods achieve results comparable to keypoint based methods, they struggle overall to meet the same levels of performance. This is because keypoint PnP based methods more effectively leverage the pre-existing geometric information necessary to solve the ill-posed problem of pose estimation. However, two-stage approaches still have their downsides. They must be trained on surrogate loss functions, and they are typically not end-to-end differentiable, limiting the learning process. To obtain the end-to-end differentiability of direct methods, the geometry-guided accuracy of PnP methods, as well as the robustness of dense methods, Wang et al. [8] developed GDR-Net (Geometry-guided Direct Regression Network). GDR-Net [8] uses an encoder-decoder style CNN to predict dense pixel-wise correspondences. But then instead of using a non-differentiable PnP solver, it uses a a convolutional Patch-PnP network to directly regress pose. Using this approach, GDR-Net achieves state-of-the-art results, comparable with refinement based methods. However, the main limitation with GDR-Net is that to achieve these results, a separate model for every target object must be trained. Extending the approach of GDR-Net, Di et al. [55] propose a network called SO-Pose [55], which achieves results similar to GDR-Net, but only requires a single model for all objects. Inspired by multi-layer techniques used in 3D reconstruction, SO-pose predicts both the dense 2D-3D correspondences, as well as self occlusion information, resulting in a two-layer representation of the object pose. Then, by enforcing cross-layer consistencies across the correspondence field, self-occlusion, and 6D pose, the solution becomes very robust to error in any single component.

2.6.2 6D Object Pose Refinement

Although recent methods such as GDR-Net [12] and SO-Pose [55], achieve high levels of accuracy compared to prior works, the ill-posedness of the problem still makes this task very challenging for RGB-only methods. As a result, refinement techniques are necessary to achieve the performance requirements of high-precision applications. Similar to traditional pose estimation techniques, early methods used either hand crafted feature descriptors, or template based matching techniques for refinement. DeepIM [1] introduced a novel neural network architecture to iteratively refine the pose of an object in a target image by matching it to a rendered image of the object's initially estimated pose. DeepIM is based on the FlowNetS [56] optical flow architecture, and directly regresses the translational and rotational updates necessary to minimize the difference in the observed and rendered images. Many recent state-of-the-art works improve upon DeepIM by addressing a variety of factors, but virtually all of them follow the same basic render-and-compare approach. CosyPose [7] for example replaces the FlowNetS backbone with EfficientNet. [25], removes the optical flow head, and directly regresses rotation in a 6D rotation parameterizaion [57] as opposed to a quaternion. Trabelsi et al. [16] introduce a combined pose proposal and refinement network. Focusing on the refinement network, [16] extracts and warps feature maps based on the optical flow between observed and rendered images. The warped feature maps then pass through a spatial multi-attention layer to highlight important features, before directly regressing the pose update. Similar to DeepIM, 4 iterations of refinement are used. The most recent work in RGB pose refinement is RNNPose [58]. RNNPose uses a shared feature encoder to extract feature maps from the observed and rendered images. The feature maps are then correlated to generate a 4D global correlation volume. Additionally, a separate 3D context encoder extracts information from the 3D model vertices. The correlation features and context information are then concatenated, and passed to a Gated Recurrent Unit (GRU) network

to extract the 2D-3D correspondence field needed to estimate pose. Finally, the estimated pose is iteratively refined by alternating updates of the correspondence field and the pose estimate. This combination of feature encoding, context encoding, 4D correlation volumes, and recurrent structure is inspired by the RAFT [59] optical flow architecture, but extended for the task of pose estimation.

Chapter 3

DeepRM Methodology

In this section, we present a detailed description of the DeepRM architecture.

3.1 Proposed Method

An overview of the proposed DeepRM method is illustrated in Figure 3.1. Inspired by DeepIM [1], it follows an iterative render-and-compare approach to refine the pose of an object in a single RGB input image. Given an initial pose estimate of a target object, an image of the target object is rendered. The rendered image is then matched with the real image of the object to predict a se(3) transform to the initial pose estimate that better aligns the object in the rendered image to the observed image. The se(3) transform consists of a translation and rotation vector, where the rotation is represented as a normalized unit quaternion. Se(3) denotes the Special Euclidean group, which refers to the set of proper rigid transformations within the Euclidean group. Such transforms within the Euclidean group preserve the Euclidean distance between transformed points. Because each update reduces the error between the rendered and observed images, this process can be repeated iteratively to incrementally refine the result. This method compensates for lack of external information such as depth by leveraging pre-existing geometric and visual properties of target objects, i.e. textured CAD models. By rendering objects in a way that is geometrically consistent with the observed scene, 3 dimensional spatial

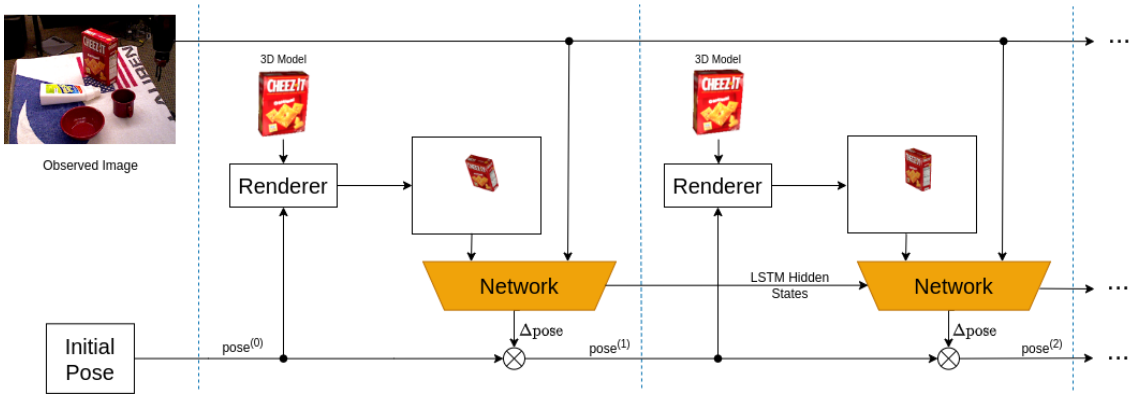

Figure 3.1: An overview of the proposed method. An initial pose estimate is used to render a target object. The observed and rendered images are passed through a convolutional neural network to predict a se(3) transformation that updates the previous pose estimate. This process is repeated several times to incrementally refine the estimate. In addition to the updated pose estimate, hidden states from recurrent LSTM modules are propagated to each iteration.

information can be recovered from the RGB only image data. While inspired by DeepIM, our proposed method improves upon this technique with several features such as high resolution cropping, disentangled loss, variable renderer brightness, a more efficient backbone, and a recurrent architecture.

3.1.1 DeepRM Architecture

The DeepRM neural network architecture is illustrated in Figure 3.2. The observed and rendered RGB images are concatenated channel-wise to form a 240x320x6 dimensional tensor. The 6-channel tensor is passed as input to the backbone convolutional neural network to extract feature maps, where the final 8x10x384 feature map from the backbone is flattened and passed through three shared, fully-connected, LSTM layers before the final translation and rotation heads. The multi-scale feature maps from the backbone network are also used in the optical flow head during training.

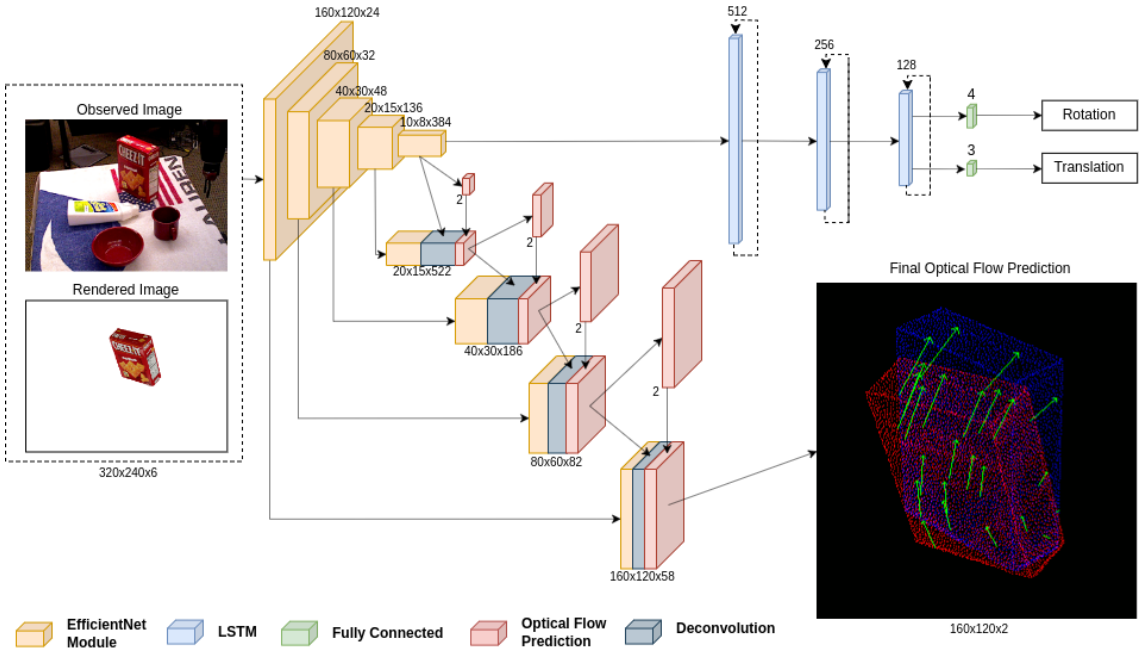

Figure 3.2: Convolutional neural network architecture of the proposed method. The observed and rendered RGB images are concatenated to form a 6-channel tensor. The 6-channel tensor is then passed as input to the backbone network to extract feature maps. The final 8x10x384 feature map is flattened and passed through three shared, fully-connected, LSTM layers before the final translation and rotation heads. The multi-scale feature maps from the backbone network are also used in the optical flow head during training.

3.1.2 Backbone

DeepRM uses the EfficientNet architecture [25] as its backbone for extracting feature maps. As discussed in Section 2.3.1, EfficientNet is a very popular and widely used architecture due to its scalability and efficiency. To demonstrate that our method can scale along with the EfficientNet backbone, we train and evaluate on EfficientNet configurations B0-B3. For each version, we adjust the size of the fully-connected LSTM layers in the pose regression heads. This is because the final feature map to the first fully-connected layer dominates the model's parameter requirements. We achieve the best results with EfficientNet-B3, and thus use this as the main configuration, unless otherwise specified.

3.1.3 Pose Regression Heads

To directly regress the pose update in the form of a se(3) transformation, fully connected layers of size 3 and 4 are used for translation and rotation, respectively. We note that the term "direct" here refers to the fact that the network predicts the pose update in an end to end differentiable manner. This is in contrast to many other works [5, 6, 43, 48, 44] which use a 2 stage process of detecting keypoints, and then running a non-differentiable Perspective-n-Point solver on the detected keypoints to estimate pose. Our network however, is not directly predicting the translation or the rotation in their final forms. For the translation components, the network predicts the transformation in pixel space, which is then converted to 3D using the thin lens equation. Equation (3.1) shows the three translation parameters that are directly regressed by the network:

$$\begin{aligned} v_x &= f_x \left[\frac{x}{z} - \frac{x'}{z'} \right] \\ v_y &= f_y \left[\frac{y}{z} - \frac{y'}{z'} \right] \\ v_z &= log \left(\frac{z'}{z} \right) \end{aligned} \tag{3.1}$$

Where f_x and f_y represent the x and y focal lengths of the camera lens. $[x, y, z]$ represents the target translation, and $[x', y', z']$ represents the initial pose estimate. As the focal lengths are consistent across all samples in our datasets, f_x and f_y are fixed to 1 during training and testing. The depth parameter, v_z, is estimated as the relative change in depth between the source image and the target. However, following [1], we use the log of the prediction factor to ensure that a predicted value of 0 results in a depth scale factor of 1.0. This stabilizes the initial training of the network when fully connected layers are initialized with a zero mean distribution. As Eq. (3.1) represents the implicit representation of quantities learned by the network, Eq. (3.2)

shows the equations actually used for recovering 3D information from the initial pose estimate $[x', y', z']$, and the predicted values $[v_x, v_y, v_z]$:

$$\begin{aligned} z &= \frac{z'}{e^{(v_z)}} \\ x &= z\left[v_x + \frac{x'}{z'}\right] \\ y &= z\left[v_y + \frac{y'}{z'}\right] \end{aligned} \tag{3.2}$$

Forcing the network to regress a scale change for depth, and an translation in pixel space has several benefits: 1) it limits the amount of reasoning that needs to be done in 3D, 2) it eliminates the need for camera intrinsics, and 3) it eliminates the need to know the scale of the target object.

For the rotation component, the network predicts the four quaternion components, which are then normalized to form a unit quaternion. By normalizing the output of the network, the network does not need to learn the complex relationship between each quaternion component, but rather simply the ratio between components.

3.1.4 Optical Flow Head

While the task of 6D pose estimation requires complex 3-dimensional reasoning, the task of optical flow is much more straight-forward, and easily learned in 2D. Yet the visual features necessary for predicting optical flow are very similar to those necessary for pose estimation. This is especially applicable when using our pixel-space formulation for translation as discussed in Section 3.1.3. We leverage this relationship by adding an optical flow head to the network during training. This reinforces the learning of features applicable to our task, and stabilizes the training process, particularly in the early stages of training. Other works such as DeepIM [1] have demonstrated that this technique also lends itself to pose estimation on

previously unseen objects, however DeepRM was not evaluated for this.

The optical flow head used in DeepRM follows the deconvolution architecture of the FlowNetS [56] network. This combines the multi-scale feature maps extracted from the backbone network with several layers of deconvolution, resulting in multi-scale optical flow predictions. FlowNetS defines a custom backbone network, but for our purposes, we replace this with EfficientNet [25] as discussed in Section 3.1.2. This includes modifying the spatial resolutions, as well as matching the deconvolution channel dimensions to those of the corresponding spatial dimension in the backbone network. Because we match the dimensions of the backbone, this allows the optical flow head to easily scale along with the backbone network. An illustration of the this component is shown in Figure 3.2, where the channel-dimensions correspond to the EfficientNet-B3 backbone.

3.1.5 Recurrent Fully-Connected Layers

While many other works [1, 7, 18, 16] in pose refinement leverage an iterative process to incrementally improve upon an initial coarse estimate, most do not leverage any type of recurrent network features. However, recurrent architectures have been successfully used to improve the iterative processes of other visual processing tasks such as optical flow prediction [59], saliency detection [60], and instance segmentation [61]. We hypothesize that adding gated recurrent mechanisms such as GRUs and LTSMs to iterative processes should generally either maintain or improve their current levels of performance. Considering the case where all gated connections are disabled, we simply have the original network configuration. We can then focus on learning only the information that is necessary to improve performance from one iteration to the next. Based on our hypothesis, we apply this theory to the task of 6D pose refinement and present a novel recurrent network architecture suited for this task.

As shown in Figure 3.2, DeepRM uses a standard convolutional neural network as

the backbone, but adds recurrent fully-connected layers to perform pose regression. These fully-connected layers leverage LSTM [20] units to propagate information from one iteration to the next. For maximum performance, the final 8x10x384 feature map is connected to a 512 dimensional LSTM layer. This results in a large number of parameters (∼16M) for the weights between these two layers, however this can be scaled down with a relatively low impact to performance. An analysis of performance under various fully-connected dimensions is provided in Section 5.2.

3.1.6 High Resolution Cropping

Although the resolution of our input data is 640x480, we choose to use a network input size of 320x240. We then crop to the region of interest around the object, based on the initially estimated pose. This cropping strategy has several benefits: 1) it reduces background clutter 2) it leverages the higher input image resolution. 3) it reduces the memory and computational requirements of the network. Objects close to the sensor will inherently appear larger in the image, and may require downsampling. However, downsampling has a low impact on performance, because the objects are well observed to begin with. Conversely, objects far away from the image sensor will appear small. In this case, we can simply crop the original image around the target object, maintaining the same pixel density as the original image. This allows our solution to leverage the higher input resolution of the input data, while maintaining the computational efficiency of a lower resolution within the neural network. Although in practice, we crop more aggressively around the target object, where the crop may need to be upsampled to reach the network input size. The benefit of a more aggressive crop is that a greater amount of background clutter is removed.

3.1.7 Loss Functions

3.1.7.1 Disentangled Point Matching Loss (DPML)

To learn 3D pose, we use the point matching loss (PML) function as in [1], but disentangle the translational components as in [62]. PML incorporates both rotational and translational error in a single scalar metric, conveniently eliminating the need to balance the separate elements. Additionally, the disentangled formulation isolates the influence of the x,y translation with the relative change in depth. For a ground truth pose $\boldsymbol{p} = [\boldsymbol{R}|\boldsymbol{T}]$, and an estimated pose $\widetilde{\boldsymbol{p}} = [\widetilde{\boldsymbol{R}}|\widetilde{\boldsymbol{T}}]$, the point matching loss is defined as the average ℓ_1 norm over a subset of n model points:

$$\mathrm{PML}(\boldsymbol{p}, \widetilde{\boldsymbol{p}}) = \frac{1}{n} \sum_{i=1}^{n} \left|\left| (Rx_i + T) - (\widetilde{R}x_i + \widetilde{T}) \right|\right|_1 \tag{3.3}$$

where x_i denotes the *i-th* model point. Extending this formula to disentangle the translational components, we first split the ground truth $\boldsymbol{T}$ into $[\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}]$ and the predicted $\widetilde{\boldsymbol{T}}$ into $[\widetilde{\boldsymbol{x}}, \widetilde{\boldsymbol{y}}, \widetilde{\boldsymbol{z}}]$. We then utilize a combination of the ground truth and predicted values as follows:

$$\begin{aligned} \mathrm{DPML}(\boldsymbol{p}, \widetilde{\boldsymbol{p}}) = \Big[& \mathrm{PML}(\widetilde{R} \,|\, [\widetilde{x}, \widetilde{y}, \widetilde{z}]) + \\ & \mathrm{PML}(\widetilde{R} \,|\, [\widetilde{x}, \widetilde{y}, z]) + \\ & \mathrm{PML}(\widetilde{R} \,|\, [x, y, \widetilde{z}]) \Big] / \, 3 \end{aligned} \tag{3.4}$$

We note that our formulation is slightly different than [62] and [7] in that it does not disentangle the rotation component. This was found experimentally to be much more stable during training, and provide better results than the fully disentangled representation.

3.1.7.2 Multi-Scale Endpoint Error (MS-EPE)

For the auxiliary optical flow head, we use the same multi-scale endpoint error (MS-EPE) loss as [56]. Endpoint error (EPE) is the most commonly used error metric for optical flow, and is the average Euclidean distance between the ground truth and predicted flow vectors for a given flow map. Multi-scale endpoint error combines the endpoint error across multiple scales to enforce the refinement of coarse feature maps to the high resolution output. Also following [56], we fix the weights for each spatial feature resolution from the top down as: 0.005, 0.01, 0.02, 0.08, 0.32, respectively.

The disentangled point matching pose loss (DPML) is then weighted with the MS-EPE optical flow loss, to realize the full loss function used during training:

$$\text{Total Loss} = \text{DPML} + 0.1\ \text{MS-EPE} \tag{3.5}$$

Chapter 4

Implementation Details

4.1 Datasets

4.1.1 YCB-Video

The YCB-Video dataset [2] is a a large scale dataset, with a total of 133,827 images over 92 unique scenes. It is a result of the collaboration between Yale, CMU, and Berkeley University to generate a large-scale dataset for 6D pose estimation, thus the name YCB. Images contain labeled 6D poses of 21 target objects. The majority of images contain 4-5 objects in the scene, resulting in high levels of occlusion, as well as a variety of challenging lighting conditions. The 21 objects are a diverse selection of common household items, which include various levels of symmetry (i.e. non-symmetric, discretely symmetric, and continuously symmetric objects).

4.1.2 Occlusion-LINEMOD

The Occlusion LINEMOD dataset [10] is an extension upon the original LINEMOD dataset [9]. LINEMOD consists of 13 common household objects, split into 13 cluttered scenes. Roughly 1000 images are provided for each object. Many target objects are present in each image, however only a single object is labeled per image. The target object in each image is also generally very visible. To create a more challenging dataset, Occlusion LINEMOD was introduced. Occlusion LINEMOD provides

Figure 4.1: Example images from the datasets. The top row contains images from the Occlusion LINEMOD dataset. The middle and bottom rows contain images from the YCB-Video dataset.

ground truth labels for all objects in one of the 13 scenes. This results in high levels of partial occlusion, significantly increasing the difficulty of the dataset. Following the convention of other works such as [8], [55], we train on LINEMOD, and evaluate on Occlusion LINEMOD. Although, due to the limited amount of real data provided in LINEMOD, we additionally augment the training set with physically-based rendering (PBR) images that are publicly available from the 2020 BOP Challenge [15].

4.2 Rendering

There are two separate contexts in which we use rendering in this work, rendering synthetic data, and rendering the input to the neural network. As synthetic data rendering was discussed in Section 2.5, we focus here on rendering the input to the neural network. Given an initial pose estimate, an RGB image must be rendered of the target object. This image is then stacked with the observed image to form a 6-channel tensor, and passed into the neural network. This process is repeated for several iterations, updating the render each iteration as pose is refined. During training, this needs to be done on every image in the batch. During inference, any time spent rendering delays the inference results. Therefore, it is very important that an efficient renderer is used. For this reason, we choose not to use a photo-realistic renderer such as BlenderProc4BOP [15], but rather a fast and lightweight OpenGL based renderer. For the input to the network, it is more important that renders are consistent and objects are well exposed, as opposed to accurately modeling complex lighting effects. Regarding proper exposure of objects, we note that related works such as [1] and [8] use a fixed light source intensity for all objects. We found that this can result in over-illumination of light colored objects, and under-illumination of dark ones. To address this, we manually tune the renderer brightness for each individual object, and then use these values during training and testing. This simple modification improved the baseline results by 0.4%. Figure 4.2 illustrates an example

scene, with objects rendered according to their ground truth poses.

4.3 Parameter Settings

DeepRM is implemented in PyTorch, and uses the same OpenGL based renderer as [1]. For both YCB-Video and Occlusion LINEMOD datasets, we use the ADAM optimizer [63], with a base learning rate of 1e-4. Although due to the differences in each dataset, we use different batch sizes, training durations, and learning rate schedules for each dataset. For YCB-Video, 16 images are used per batch, with 4 objects per image, resulting in an effective batch size of 64. Similar to DeepIM [1], the model is trained for 20 epochs, with fixed learning rate decays of 0.1x at epochs 10 and 15. Although best results are obtained earlier at epoch 19, likely due to overfitting past this point. For Occlusion LINEMOD, 48 images are used per batch, with 1 object per image, resulting in an effective batch size of 48. Number of epochs are scaled up to 190, to account for the difference in the size of the dataset and batch size compared to YCB-Video. Additionally, for Occlusion LINEMOD only, a warmup period of 10% base learning rate is used in the first 4 epochs. Both datasets are trained with 6 refinement iterations during training. Then during testing, 8 iterations of refinement are used for YCB-Video, and 6 iterations are used for Occlusion LINEMOD.

4.4 Evaluation Metrics

To evaluate the performance against other state-of-the-art methods, we follow [1, 6, 7, 18, 16, 55, 12] and use the Average Distance Diameter (ADD) metric [9]. More specifically, we use two specific variations upon it, depending on the dataset, ADD(-S) 10% for LM-O and area under the curve (AUC) ADD(-S) for YCB-Video. With the ground truth and estimated pose of a given object, the ADD metric is the average error of every 3D point correspondence between the ground truth and estimated

Figure 4.2: An example scene from the YCB-Video dataset, and a set of rendered objects, according to their ground truth poses.

object poses. In this case, error is defined as the Euclidean distance between two 3-dimensional points. Note that this requires a 3D model of the object, although the model in this case does not need to contain any texture information, simply the 3D geometric information such as the model's vertices. ADD is therefore calculated as:

$$\text{ADD} = \frac{1}{m} \sum_{x \in M} \left|\left|(Rx + T) - (\widetilde{R}x + \widetilde{T})\right|\right|_2 \tag{4.1}$$

where R and T are the ground truth rotation and translation, $\widetilde{R}$ and $\widetilde{T}$ are the estimated rotation and translation, and M is the set points in the 3D model. The ADD metric is convenient due to the fact that it accounts for both rotational and translational errors in a single scalar value. This feature also makes it very suitable to be used as a loss function. There is however a major limitation with this method for objects containing symmetries. By matching vertices 1-to-1 between the estimated and ground truth poses, objects which contain symmetries are not properly handled. For example, the foam brick object from the YCB-Video dataset contains a discrete symmetry where it can be rotated 180 degrees, resulting in a pose that is indistinguishable from the original. This rotation would undesirably result in a high ADD error if encountered during evaluation. To account for this limitation, we use the ADD-S metric for symmetric objects, which instead computes the closest point distance, rather than the matching point distance:

$$\text{ADD-S} = \frac{1}{m} \sum_{x_1 \in M} \min_{x_2 \in M} \left|\left|(Rx + T) - (\widetilde{R}x + \widetilde{T})\right|\right|_2 \tag{4.2}$$

To then combine per class ADD or ADD-S scores, depending on the object symmetry types, we adopt the notation ADD(-S). ADD(-S) selectively evaluates each class, based on whether it is symmetric or non-symmetric. This enables a concise summary of performance on each dataset.

While ADD metric provides a scalar value representing the amount of error in

a single pose estimate, it does not account for events such as missed detections. To effectively evaluate overall performance, we use the ADD(-S) 10% metric on the LM-O dataset, and the AUC ADD(-S) metric on the YCB-Video dataset. ADD(-S) 10% is defined as the percentage of pose estimates where the ADD(-S) metric falls below 10% of the target object's diameter. The 10% threshold is unanimously used among other works, and was driven by the most common downstream task of pose estimation, robotic grasping, where 10% is generally sufficient to enable successful grasping of objects. The 10% metric however imposes a hard threshold, and does not display how a method performs outside of this threshold. Because of this, the AUC ADD(-S) metric is instead used on the YCB-Video datset, following [2, 8, 7, 16, 55]. To calculate the area under the curve, the error threshold is swept from 0 to 0.1 meters.

Chapter 5

Results

We evaluate DeepRM on the YCB-Video [2] and Occlusion LINEMOD [10] datasets. Results are compared with the methods providing the initial coarse predictions, as well as the current state-of-the-art.

5.1 Comparison to State-of-the-Art

Results on YCB-Video dataset. Table 5.1 presents the results of DeepRM compared to the current state-of-the-art on the YCB-Video dataset for the AUC ADD(-S) metric. Initial predictions are obtained from PoseCNN [2], where DeepRM outperforms the leading state-of-the-art network CosyPose [7], by 2.1%. GDR-Net [12] is also very comparable, but only when a unique model is trained for each target object. For a shared model, DeepRM outperforms GDR-Net by a large margin of 6.4%.

Table 5.2 provides a further detailed description of the results on the YCB-Video dataset. Both the AUC ADD, and AUC ADD-S metrics are reported for all target objects. Recalling from Section 4.4, ADD(-S) is the combination of these two metrics, depending on whether or not the object is considered symmetric. When accounting for symmetries with the ADD-S metric, DeepRM outperforms PoseCNN [2] and DeepIM [1] on all 21 object classes. Neglecting symmetries, PoseCNN achieves superior results on the "bowl" and "foam_brick" objects. This is likely due to the fact that these objects are textureless, and have high levels of symmetry. The "bowl"

Method	P.E.	Ref.	AUC of ADD(-S)
PoseCNN [2] ⋆	1		61.31
PVNet [6]	1		73.4
RePose [19]	M	✓	77.2
GDR-Net [12]	1		80.2
DeepIM [1]	1	✓	81.9
RNNPose [58]	M	✓	83.1
Trabelsi [16]	1	✓	83.1
SO-Pose [55]	1		83.9
GDR-Net [12]	M		84.4
CosyPose [7]	1	✓	84.5
DeepRM (Ours)	1	✓	**86.6**

Table 5.1: Comparison to state-of-the-art on the YCB-Video dataset. Ref. indicates that the network includes refinement. ⋆ indicates the method that is used to provide the initial coarse estimates to our network. P.E. indicates whether a unique model is trained per object, versus a single model for all objects. M represents a unique model per object, and 1 represents a single model for all objects.

object has a continuous symmetry around its vertical axis, and the "foam_brick" has several discrete axis of symmetry. PoseCNN is trained with a symmetry aware loss function, whereas DeepRM is not. This decision was made to reduce computational requirements during training. Therefore, it is likely that the performance of DeepRM could be improved with a symmetry aware loss function. Successes of other recent works such as [7] and [16] further support this idea. After objects with symmetries, the next category of classes that DeepRM slightly struggles with are small objects, such as "scissors" and "large_marker". Although performance on these classes is still near or above 90%, which is acceptable for practical use cases. For textureless objects such as "banana" and "pitcher_base", DeepRM performs exceedingly well, achieving 95% accuracy. This demonstrates that DeepRM is able to operate on textureless objects, addressing one of the main limitations of feature based methods. Figure 5.1 illustrates successful refinement sequences on both occluded and texture-less objects.

As the intention of this work is to further close the gap between RGB and RGB-D based methods, Table 5.3 presents a comparison of DeepRM to the current state of the art RGB-D methods on the YCB-Video dataset. We show that DeepRM only

lags behind RGB-D based methods by 1.5 to 5%.

Method	PoseCNN [2]		DeepIM[1]		DeepRM (Ours)	
Evaluation Metric	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S
master_chef_can	50.2	83.9	71.2	93.1	**77.0**	**94.8**
cracker_box	53.1	76.9	83.6	91.0	**87.7**	**92.5**
sugar_box	68.4	84.3	94.1	96.2	**95.8**	**97.1**
tomato_soup_can	66.2	81.0	86.1	92.4	**89.5**	**93.7**
mustard_bottle	81.0	90.4	91.5	95.1	**93.4**	**96.1**
tuna_fish_can	70.7	88.1	87.7	96.1	**90.9**	**97.2**
pudding_box	62.7	79.1	82.7	90.7	**89.1**	**93.1**
gelatin_box	75.2	87.2	91.9	94.3	**93.4**	**95.4**
potted_meat_can	59.5	78.5	76.2	86.4	**79.8**	**91.3**
banana	72.3	86.0	81.2	91.3	**90.7**	**95.2**
pitcher_base	53.3	77.0	90.1	94.6	**93.6**	**95.9**
bleach_cleanser	50.3	71.6	81.2	90.3	**83.1**	**91.6**
bowl*	**30.0**	70.0	8.6	81.4	5.4	**83.9**
mug	58.5	78.2	81.4	91.3	**88.1**	**94.1**
power_drill	55.3	72.7	85.5	92.3	**92.0**	**95.2**
wood_block*	26.6	64.3	60.0	81.9	**79.9**	**90.0**
scissors	35.8	56.9	60.9	75.4	**79.5**	**89.3**
large_marker	58.3	71.7	75.6	86.2	**79.3**	**90.2**
large_clamp*	24.6	50.2	48.4	74.3	**51.7**	**76.7**
extra_large_clamp*	16.1	44.1	31.0	73.3	**39.5**	**78.8**
foam_brick*	**72.9**	88.2	35.9	81.9	42.5	**90.7**
Mean	53.4	74.6	71.7	88.1	**77.2**	**91.6**

Table 5.2: Detailed results on the YCB-Video dataset for RGB based methods. ADD and ADD-S represent AUC ADD and AUC ADD-S metrics, respectively. (*) denotes symmetric objects.

	Methods				
	RGB-D				RGB
Object	DeepIM	DenseFusion	PVN3D	FFB6D	DeepRM (Ours)
002 master chef can	96.3	**96.4**	96.0	96.3	94.8
003 cracker box	95.3	95.5	96.1	**96.3**	92.5
004 sugar box	**98.2**	97.5	97.4	97.6	97.1
005 tomato soup can	94.8	94.6	**96.2**	95.6	93.7
006 mustard bottle	**98.0**	97.2	97.5	97.8	96.1
007 tuna fish can	**98.0**	96.6	96.0	96.8	97.2
008 pudding box	90.6	96.5	**97.1**	**97.1**	93.1
009 gelatin box	**98.5**	98.1	97.7	98.1	95.4
010 potted meat can	90.3	91.3	93.3	**94.7**	91.3
011 banana	**97.6**	96.6	96.6	97.2	95.2
019 pitcher base	**97.9**	97.1	97.4	97.6	95.9
021 bleach cleanser	**96.9**	95.8	96.0	96.8	91.6
024 bowl*	87.0	88.2	90.2	**96.3**	83.9
025 mug	**97.6**	97.1	**97.6**	97.3	94.1
035 power drill	**97.9**	96.0	96.7	97.2	95.2
036 wood block*	91.5	89.7	90.4	**92.6**	90.0
037 scissors	96.0	95.2	96.7	**97.7**	89.3
040 large marker	**98.2**	97.5	96.7	96.6	90.2
051 large clamp*	77.9	72.9	93.6	**96.8**	76.7
052 extra large clamp*	77.8	69.8	88.4	**96.0**	78.8
061 foam brick*	**97.6**	92.5	96.8	97.3	90.7
ALL	94.0	93.1	95.5	**96.6**	91.6

Table 5.3: Detailed comparison against RGB-D based methods for the YCB-Video dataset. The ADD-S metric is used for evaluation. (*) denotes symmetric objects. Both DeepIM [1] and our method are initialized with predictions from PoseCNN [2].

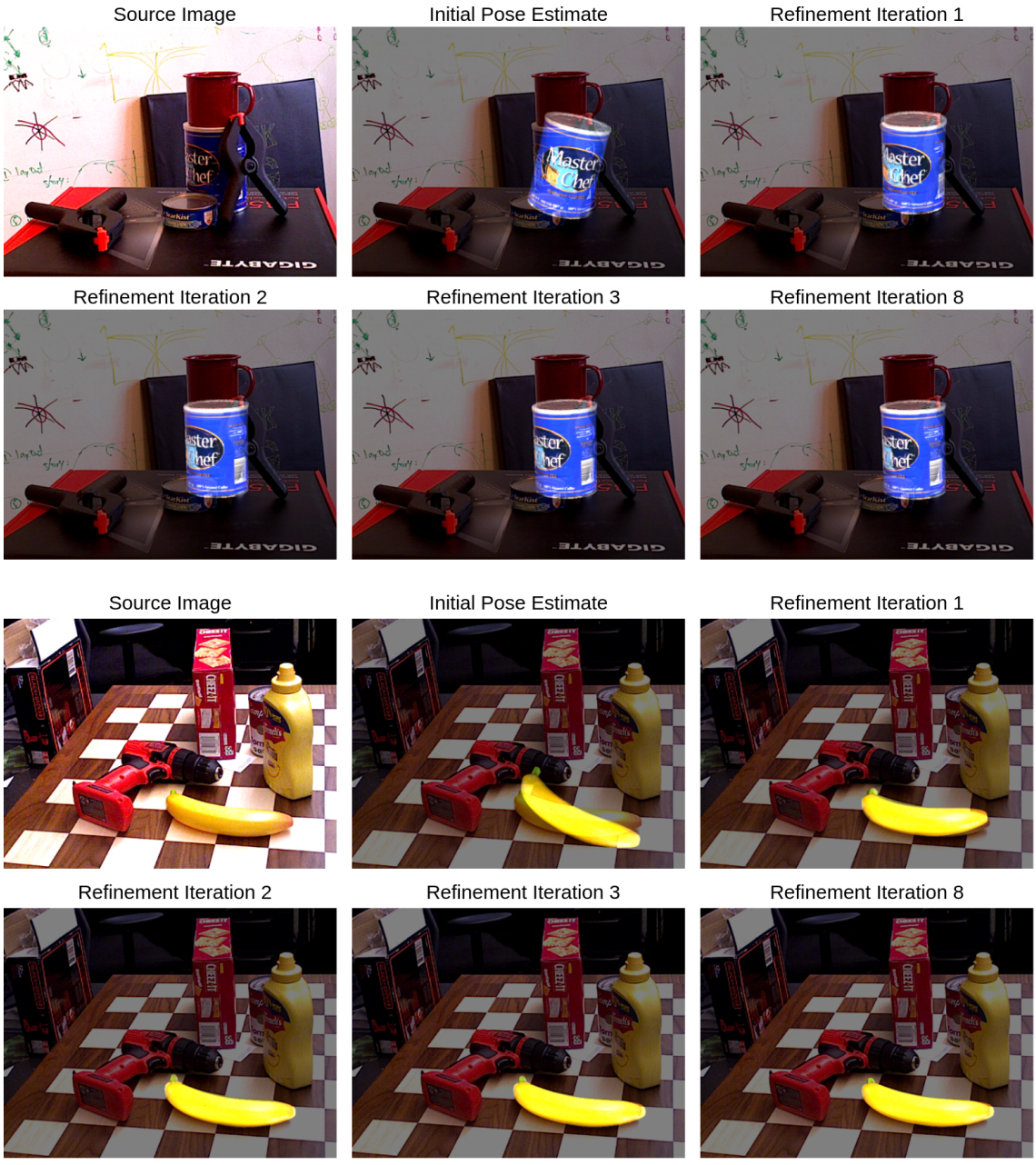

Figure 5.1: Example refinement sequences.

Method	P.E.	Ref.	ADD(-S) 10%
PoseCNN [2]	1		24.9
PVNet [6] ⋆	1		40.8
RePose [19]	M	✓	51.6
PPC [64]	1	✓	55.3
DeepIM [1]	1	✓	55.5
GDR-Net [12]	1		56.1
Trabelsi [16]	1	✓	58.4
RNNPose [58]	M	✓	60.7
GDR-Net [12]	M		62.2
SO-Pose [55]	1		62.3
DeepRM (Ours)	1	✓	**65.0**

Table 5.4: Comparison to state-of-the-art on the LM-O dataset. Ref. indicates that the network includes refinement. ⋆ indicates the method that is used to provide the initial coarse estimates to our network. P.E. indicates whether a unique model is trained per object, versus a single model for all objects. M represents a unique model per object, and 1 represents a single model for all objects.

Results on LM-O dataset. Table 5.4 presents the results of DeepRM compared to the current state-of-the-art on the Occlusion LINEMOD dataset for the ADD(-S) 10% metric. Initial predictions are obtained from PVNet [6], where DeepRM outperforms the leading state-of-the-art method SO-Pose [55] by 2.7%. As with YCB-Video, GDR-Net [12] is also very comparable, but only when a unique model is trained for each target object. For a shared model, DeepRM outperforms GDR-Net by a large margin of 8.9%. Furthermore, because DeepRM is based on the DeepIM [1] framework, we consider DeepIM's performance to be a baseline, and show a 9.5% improvement over this.

Table 5.5 reports the class-wise performance of DeepRM compared to other RGB based refinement methods on the Occlusion LINEMOD dataset. With the exception of "glue" and "driller", all objects in Occlusion LINEMOD are textureless. Additionally, objects are also relatively small compared to YCB-Video. Combining the small and textureless qualities of objects with high occlusion makes this dataset very challenging. Nonetheless, DeepRM outperforms the current state-of-the-art RGB based refinement method, RNNPose, [58] by 4.3%. Similarly to YCB-Video, DeepRM

Method	DeepIM[1]	RePOSE [19]	RNNPose [58]	DeepRM (Ours)
ape	**59.2**	31.1	37.2	50.3
can	63.5	80.0	88.1	**93.4**
cat	26.2	25.6	29.2	**47.1**
driller	55.6	73.1	**88.1**	85.8
duck	52.4	43.0	49.2	**56.0**
eggbox*	63.0	51.7	**67.0**	62.8
glue*	**71.7**	54.3	63.8	59.5
hole.	52.5	53.6	62.8	**75.0**
MEAN	55.5	51.6	60.7	**65.0**

Table 5.5: Detailed results on the Occlusion LINEMOD dataset for refinement based methods. The ADD(-S) 10% metric is used for evaluation. (*) denotes symmetric objects.

struggles with symmetric objects, being outperformed by other methods on the two symmetric classes: "eggbox" and "glue". This further reinforces the theory that a symmetry aware loss function would be beneficial to DeepRM.

5.2 Ablation Studies

5.2.1 Ablation Study on Backbone Architectures

Table 5.6 displays network performance as a function of various backbone architectures, resolution, and the number of trainable parameters per architecture for the YCB-Video dataset. Resolution in this case is regarding the input to the neural network, not the original input image resolution. Unlike the original convention of EfficientNet [25], we choose to fix the input resolution of our network to 320x240 when using any of the EfficientNet backbones. This ensures we have a consistent spatial dimension on our final feature map. We instead demonstrate how adjusting the dimensions of the fully connected LSTM layers affects the performance and size of the model. Best results are achieved with the EfficientNet-B3 backbone, and a fully-connected configuration of 512→256→128. However, the EfficientNet-B0 256→256→256 configuration still achieves results superior to all state-of-the-art methods, using 2x less

Backbone	Resolution	FC Layer Dims	# Params	AUC of ADD(-S)
DeepIM [1] (baseline)	640x480	256→256	60M	81.9
DeepRM FlowNetS	640x480	512→256→128	208 M	83.3
DeepRM EfficientNet-B0	320x240	128→128→128	20 M	84.5
DeepRM EfficientNet-B0	320x240	256→256→256	34 M	85.7
DeepRM EfficientNet-B2	320x240	384→256→256	55 M	85.5
DeepRM EfficientNet-B3	320x240	512→256→128	79 M	**86.6**

Table 5.6: Ablation Study on Various Backbones Architectures for YCB-Video.

parameters than the larger version.

5.2.2 Ablation Study on LSTMs vs GRUs

As discussed in Section 2.2.1, Gated Recurrent Units (GRUs) [21] are a simplified version of LSTM modules. Since the task of pose estimation can have real-time processing constraints for certain applications [19], we evaluate the performance of LSTMs vs GRUs in Table 5.7. For the smaller backbone network, EfficientNet-B0, LSTMs require 22% more parameters, and provide a negligible accuracy improvement of 0.2%. Therefore, we conclude that for real-time processing tasks, GRUs are a preferable alternative to LSTMs for our architecture. However, for the EfficientNet-B3 backbone, LSTMs show a more significant accuracy improvement of 1.2% over GRUs. Consequently, when maximum accuracy is required, the improved performance of LSTMs may be worth the increased complexity.

Backbone	Type	FC Layer Dims	# Params	AUC of ADD(-S)
EfficientNet-B0	GRU	256→256→128	27 M	84.5
	LSTM	256→256→128	33 M	**84.7**
EfficientNet-B3	GRU	512→256→128	63 M	85.4
	LSTM	512→256→128	79 M	**86.6**

Table 5.7: Ablation Study on LSTMs vs GRUs for YCB-Video.

5.2.3 Ablation Study on Optical Flow

As discussed in Sections 2.4 and 3.1.4, optical flow is a powerful technique that can be used as a prior to a variety processing tasks. We hypothesize in this work that 6D object pose estimation is one such task. To test this hypothesis, we perform an ablation study on our architecture, where we train and test with and without the auxiliary optical flow head. Table 5.8 displays the results of this study. We find that the auxiliary optical flow head provides an accuracy improvement of 1.8% on the EfficientNet-B3 backbone configuration of our network, clearly demonstrating its benefit. Furthermore, this improvement only costs a 5% increase in parameters during training. At inference, this portion of the network is removed.

Method	# Params	AUC of ADD(-S)
No Flow	75 M	84.8
Flow	79 M	**86.6**

Table 5.8: Ablation Study on the impact of the auxiliary optical flow head.

5.2.4 Ablation Study on Refinement Iterations

The process of iterative refinement is heavily dependent on the number of iterations performed. As such, we investigate the impact of training and testing on a variety of refinement iterations. All tests were performed with the EfficientNet-B3 backbone on the YCB-Video dataset. AUC ADD(-S) results are reported in Table 5.9. Best performance is achieved when training with 6 iterations, and testing with 8 iterations. We note that too many iterations during testing actually starts to hurt performance, as opposed to converging to a stable value. This indicates that there may be a need for a more sophisticated stopping criteria, as opposed to stopping after a fixed number of iterations. Manhardt et al. [18] for example stop if the last update was less than 1.5°, and 7.5mm. We theorize that we may be able to re-introduce the optical flow head into testing, and set a stopping condition based on a normalized magnitude of the

train iters	init	2			4				6			
test iters		2	4	6	2	4	6	8	2	4	6	8
ADD(-S)	60.0	82.3	82.5	82.2	84.2	85.5	85.5	85.4	83.9	86.0	86.4	**86.6**

Table 5.9: Ablation Study on refinement iterations. ADD(-S) represents AUC ADD(-S).

optical flow field, however, this is a topic for future work. We also note that we are able to refine for up to 8 iterations, while similar works such as [1, 7, 16] use a fixed number of 2-4 iterations. We believe that this is enabled by the recurrent features of our network, which effectively propagate information through each iteration to the next.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we introduce DeepRM, a novel method for precise 6D pose estimation of rigid objects from RGB only data. DeepRM improves upon existing render-and-compare approaches by leveraging several unique elements, such as an optical flow enforced learning process, an efficient and scalable framework, and a recurrent refinement technique. We demonstrate that gated memory mechanisms such as LSTMs and GRUs can be used to improve the performance of this task by propagating additional information though each refinement step. We also demonstrate the benefit of using optical flow as an auxiliary task to reinforce the learning of features applicable to our problem. Along with these unique elements, DeepRM provides the first scalable framework for 6D pose estimation to ensure it can be used across a variety of practical applications. By improving both the accuracy and reliability of this task on RGB only data, we are able to continue expanding the areas where it can be applied.

6.2 Future Work

While it achieved state-of-the-art results, DeepRM slightly struggled on symmetric objects when compared to non-symmetric ones. This is likely due to our direct use of the point matching loss function for pose, which does not account for object symme-

tries during training. Meanwhile, related works such as [7] and [16] utilize techniques for addressing this limitation. Labb et al. [7] for example discretizes ground truth poses over continuous axis of symmetry, and only selects the closest pose to the current prediction when calculating the loss. Due to the additional computational complexity it would require during training, we did not implement this feature, but reserve it as a topic for future work.

Bibliography

[1] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "DeepIM: Deep Iterative Matching for 6D Pose Estimation," *International Journal of Computer Vision*, vol. 128, no. 3, pp. 657–678, oct 2020. [Online]. Available: http://arxiv.org/abs/1804.00175http://dx.doi.org/10.1007/s11263-019-01250-9

[2] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes," *ArXiv*, may 2017. [Online]. Available: http://arxiv.org/abs/1711.00199

[3] M. Billinghurst, A. Clark, and G. Lee, "A survey of augmented reality," *Foundations and Trends in Human-Computer Interaction*, vol. 8, no. 2-3, pp. 73–272, 2014.

[4] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, pp. 1530–1538, 2017. [Online]. Available: https://wadimkehl.github.io/

[5] B. Tekin, S. N. Sinha, and P. Fua, "Real-Time Seamless Single Shot 6D Object Pose Prediction," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 292–301, 2018.

[6] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao, "PVNET: Pixel-wise voting network for 6dof pose estimation," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, pp. 4556–4565, dec 2019. [Online]. Available: http://arxiv.org/abs/1812.11788

[7] Y. Labbé, J. Carpentier, M. Aubry, and J. Sivic, "CosyPose : Consistent Multi-view Multi-object 6D Pose Estimation," in *European Conference on Computer Vision*, vol. 2, 2020, pp. 574–591.

[8] G. Wang, F. Manhardt, F. Tombari, and X. Ji, "GDR-Net: Geometry-Guided Direct Regression Network for Monocular 6D Object Pose Estimation," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 16 611–16 621, 2021. [Online]. Available: http://arxiv.org/abs/2102.12145

[9] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7724 LNCS, no. PART 1, pp. 548–562, 2013.

[10] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, "Learning 6D object pose estimation using 3D object coordinates," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8690 LNCS, no. PART 2, 2014, pp. 536–551. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-319-10605-2

[11] P. Wohlhart and V. Lepetit, "Learning descriptors for object recognition and 3D pose estimation," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, no. 1, pp. 3109–3118, 2015.

[12] C. Wang, D. Xu, Y. Zhu, R. Martin-Martin, C. Lu, L. Fei-Fei, and S. Savarese, "DenseFusion: 6D object pose estimation by iterative dense fusion," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, jan 2019, pp. 3338–3347. [Online]. Available: http://arxiv.org/abs/1901.04780

[13] Y. Wu, M. Zand, A. Etemad, and M. Greenspan, "Vote from the Center: 6 DoF Pose Estimation in RGB-D Images by Radial Keypoint Voting," *ArXiv*, 2021. [Online]. Available: http://arxiv.org/abs/2104.02527

[14] Y. He, H. Huang, H. Fan, Q. Chen, and J. Sun, "FFB6D: A Full Flow Bidirectional Fusion Network for 6D Pose Estimation," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3003–3013, 2021. [Online]. Available: http://arxiv.org/abs/2103.02242

[15] T. Hodaň, M. Sundermeyer, B. Drost, Y. Labbé, E. Brachmann, F. Michel, C. Rother, and J. Matas, "BOP Challenge 2020 on 6D Object Localization," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12536 LNCS, pp. 577–594, 2020.

[16] A. Trabelsi, M. Chaabane, N. Blanchard, and R. Beveridge, "A Pose Proposal and Refinement Network for Better 6D Object Pose Estimation," in *IEEE Winter Conference on Applications of Computer Vision*, 2021, pp. 2381–2390.

[17] E. Brachmann, F. Michel, A. Krull, M. Y. Yang, S. Gumhold, and C. Rother, "Uncertainty-Driven 6D Pose Estimation of Objects and Scenes from a Single RGB Image," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 3364–3372, 2016.

[18] F. Manhardt, W. Kehl, N. Navab, and F. Tombari, "Deep model-based 6d pose refinement in rgb," *CoRR*, vol. abs/1810.0, pp. 833–849, 2018. [Online]. Available: http://campar.in.tum.

[19] S. Iwase, X. Liu, R. Khirodkar, R. Yokota, and K. M. Kitani, "RePOSE: Real-Time Iterative Rendering and Refinement for 6D Object Pose Estimation,"

in *IEEE/CVF International Conference on Computer Vision*, 2021. [Online]. Available: http://arxiv.org/abs/2104.00633

[20] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[21] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," *Proceedings of SSST 2014 - 8th Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111, 2014.

[22] T. N. W. D. H. HUBEL, "Receptive fields of single neurones in the cat's striate cortex," *International System on Chip Conference*, pp. 247–250, 1959.

[23] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–14, 2015.

[24] G. E. H. Alex Krizhevsky, Ilya Sutskever, "ImageNet Classification with Deep Convolutional Neural Networks," *Handbook of Approximation Algorithms and Metaheuristics*, pp. 1–1432, 2012.

[25] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 10 691–10 700, 2019.

[26] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[27] Y. Hu, J. Hugonot, P. Fua, and M. Salzmann, "Segmentation-driven 6D object pose estimation," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, pp. 3380–3389, apr 2019. [Online]. Available: http://arxiv.org/abs/1812.02541

[28] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, vol. 24, no. 9, 2018, pp. 583–593.

[29] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, pp. 2815–2823, 2019.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, 2016, pp. 770–778. [Online]. Available: http://image-net.org/challenges/LSVRC/2015/

[31] Y. H. Tsai, M. H. Yang, and M. J. Black, "Video Segmentation via Object Flow," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 3899–3908, 2016.

[32] S. Wang, R. Clark, H. Wen, and N. Trigoni, "DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2043–2050, 2017.

[33] J. L. S. S. Beauchemin, Barron, "The computation of optical flow," *DWI Reports*, vol. 117, no. 3, pp. 1–672, 1996.

[34] G. Wang, F. Manhardt, J. Shao, X. Ji, N. Navab, and F. Tombari, "Self6D: Self-supervised Monocular 6D Object Pose Estimation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12346 LNCS, pp. 108–125, 2020.

[35] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 248–255.

[36] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Computer Vision – ECCV 2014*, vol. 8693 LNCS, no. PART 5, 2014, pp. 740–755.

[37] F. Sadeghi and Sergey Levine, "CAD 2 RL : Real Single-Image Flight Without a Single Real Image," *Science and Systems Conference*, 2017.

[38] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects," *ArXiv*, no. CoRL, pp. 1–11, 2018. [Online]. Available: http://arxiv.org/abs/1809.10790

[39] A. Collet and M. Martinez, "MOPED: Object Recognition and Pose Estimation for Manipulation," *The International Journal of Robotics Research*, vol. 30, pp. 1284–1306, 2011. [Online]. Available: https://personalrobotics.ri.cmu.edu/projects/moped.php

[40] D. G. Lowe, "Object recognition from local scale-invariant features," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2, pp. 1150–1157, 1999.

[41] H. Bay, T. Tuytelaars, and L. V. Gool, "SURF: Speeded Up Robust Features," *Computer Vision–ECCV 2006*, pp. 404–417, 2006. [Online]. Available: http://link.springer.com/chapter/10.1007/11744023_32

[42] M. A. Fischler and R. C. Bolles, "RANSAC: Random Sample Paradigm for Model Consensus: A Apphcatlons to Image Fitting with Analysis and Automated Cartography," *Graphics and Image Processing*, vol. 24, no. 6, pp. 381–395, 1981.

[43] M. Rad and V. Lepetit, "BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, 2017, pp. 3848–3856.

[44] M. Oberweger, M. Rad, and V. Lepetit, "Making deep heatmaps robust to partial occlusions for 3D object pose estimation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11219 LNCS, pp. 125–141, 2018.

[45] C. Song, J. Song, and Q. Huang, "HybridPose: 6D Object Pose Estimation under Hybrid Representations," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 428–437, 2020.

[46] B. Chen, Á. Parra, J. Cao, N. Li, and T. J. Chin, "End-to-end learnable geometric vision by backpropagating PNP optimization," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 8097–8106, 2020.

[47] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 779–788, 2016.

[48] G. Pavlakos, X. Zhou, A. Chan, K. G. Derpanis, and K. Daniilidis, "6-DoF object pose from semantic keypoints," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2011–2018, mar 2017. [Online]. Available: http://arxiv.org/abs/1703.04670

[49] Z. Zhao, G. Peng, H. Wang, H.-S. Fang, C. Li, and C. Lu, "Estimating 6D Pose From Localizing Designated Surface Keypoints," *ArXiv*, 2018. [Online]. Available: http://arxiv.org/abs/1812.01387

[50] S. Zakharov, I. Shugurov, and S. Ilic, "DPOD: 6D pose object detector and refiner," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2019-Octob, feb 2019, pp. 1941–1950. [Online]. Available: http://arxiv.org/abs/1902.11020

[51] K. Park, T. Patten, and M. Vincze, "Pix2pose: Pixel-wise coordinate regression of objects for 6D pose estimation," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2019-Octob, pp. 7667–7676, 2019.

[52] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," *Lecture Notes in Computer Science*

(including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9905 LNCS, pp. 21–37, 2016.

[53] Y. Bukschat and M. Vetter, "EfficientPose: An efficient, accurate and scalable end-to-end 6D multi object pose estimation approach," *arXiv*, 2020. [Online]. Available: http://arxiv.org/abs/2011.04307

[54] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 10 778–10 787, 2020.

[55] Y. Di, F. Manhardt, G. Wang, X. Ji, N. Navab, and F. Tombari, "SO-Pose: Exploiting Self-Occlusion for Direct 6D Pose Estimation," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 1, 2021. [Online]. Available: http://arxiv.org/abs/2108.08367

[56] A. Dosovitskiy, P. Fischery, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. V. D. Smagt, D. Cremers, and T. Brox, "FlowNet: Learning optical flow with convolutional networks," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, pp. 2758–2766, 2015.

[57] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, 2019, pp. 5738–5746.

[58] Y. Xu, K.-Y. Lin, G. Zhang, X. Wang, and H. Li, "RNNPose: Recurrent 6-DoF Object Pose Refinement with Robust Correspondence Field Estimation and Pose Optimization," *ArXiv*, vol. 1, 2022. [Online]. Available: http://arxiv.org/abs/2203.12870

[59] Z. Teed and J. Deng, "RAFT: Recurrent All-Pairs Field Transforms for Optical Flow," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12347 LNCS, pp. 402–419, 2020.

[60] Z. Deng, X. Hu, L. Zhu, X. Xu, J. Qin, G. Han, and P.-a. Heng, "R^{3}Net: Recurrent Residual Refinement Network for Saliency Detection," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 2018, pp. 684–690. [Online]. Available: https://doi.org/10.24963/ijcai.2018/95

[61] M. Ren and R. S. Zemel, "End-to-end instance segmentation with recurrent attention," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 293–301, 2017.

[62] A. Simonelli, S. R. Bulo, L. Porzi, M. Lopez-Antequera, and P. Kontschieder, "Disentangling monocular 3D object detection," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2019-Octob, pp. 1991–1999, 2019.

[63] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

[64] L. Brynte and F. Kahl, "Pose Proposal Critic: Robust Pose Refinement by Learning Reprojection Errors," in *Proceedings of the British Machine Vision Conference*, 2020, pp. 1–16. [Online]. Available: http://arxiv.org/abs/2005.06262