Czech Technical University in Prague
Faculty of Electrical Engineering

6D Pose Estimation of Textureless Objects from a Single Camera

Odhadování rotace a translace netexturovaného objektu z jedné kamery

BACHELOR THESIS

Author:	Michal Lukeš
Supervisor:	prof. Ing. Jiří Matas, Ph.D.
Department:	Department of Cybernetics
Study programme:	Open Informatics
Branch of study:	Computer and Information Science
Year:	2021

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Lukeš Michal**	Personal ID number:	**474572**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Branch of study:	**Computer and Information Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

6D Pose Estimation of Textureless Objects from a Single Camera

Bachelor’s thesis title in Czech:

Odhadování rotace a translace netexturovaného objektu z jedné kamery

Guidelines:

1. Familiarize yourself with state-of-the-art approaches to 6D pose estimation [1].
2. Select a method suitable for textureless, shiny objects, such as EPOS [2]. A method with available implementation is recommended, otherwise, implement it.
3. Create a small database of a few objects, obtain their 3D model and generate, if necessary, training images required by the selected method.
4. Create a dataset of test views with known 6D ground truth information. Consider using a robotic arm, calibrated w.r.t. the camera.
5. Evaluate the performance of the selected method, and analyse its failure modes.
6. (optionally) Propose improvements to the select method, and evaluate them.
7. (optionally) Implement a demo with a robotic system grasping and moving an object in the field of view of a calibrated camera.

Bibliography / sources:

[1] BOP: Benchmark for 6D Object Pose Estimation - BOP Challenge 2020 on 6D Object Localization. ECCV Workshops (2) 2020 : 577-594 (also available at http://cmp.felk.cvut.cz/home/)
[2] Tomáš Hodaň, Dániel Baráth, Jiří Matas - EPOS: Estimating 6D Pose of Objects with Symmetries C VPR 2020 : 11700-11709, 2020 (also available at http://cmp.felk.cvut.cz/epos/)

Name and workplace of bachelor’s thesis supervisor:

prof. Ing. Jiří Matas, Ph.D., Visual Recognition Group, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **17.01.2021** Deadline for bachelor thesis submission: **13.08.2021**

Assignment valid until: **30.09.2022**

prof. Ing. Jiří Matas, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Author statement for undergraduate thesis:

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 13.8.2021 ....

Michal Lukeš

Abstrakt: V této práci se věnuji vyhledávání objektů v prostoru na základě jediného RGB snímku a to jak pozice na všech třech osách tak i rotace kolem každé z nich za pomocí 3D modelů daných objektů. Uplatnění těchto metod je zejména v robotickém uchopování, autonomním řízení, nebo augmentované realitě. Skvělým zdrojem pro hledání vhodné metody je BOP Challenge [
1] , ve kterém jsou porovnávány nejlepší nové algoritmy na množině datasetů. Vybraný algoritmus pak budu přizpůsobovat a naučím jej na svém vlastním datasetu. Současné nejlepší metody pro 6D detekci objektů používají kombinaci klasifikátorů - například Cosypose [2] používá 3 různé neuronové sítě a EPOS [3] používá k predikci 6 kroků včetně vlastní neuronové sítě. Oba algoritmy mají dostupnou implementaci a skvělé výsledky v BOP.
Pro ukázku funkčnosti si vyberu 4 objekty a jejich 3d modely a pomocí kamery se pokusím vytvořit základní dataset. Dále ale pokračuji technikou renderování fotorealistických obrázků, která je kvůli automatickému anotování objektů ve všech dimenzích mnohem rychlejší a praktičtější na velká množství dat nutná pro trénování neuronové sítě.
Klíčová slova: 6D, odhad polohy, netexturované a lesklé předměty

Abstract: This thesis focuses on estimating the pose of objects based on only one RGB image of the scene. This includes the position of the object on the three-axis as well as its rotation using 3D models of the objects. Deployment of such methods is mainly in robotic grasping, autonomous driving or augmented reality. An excellent source for comparing these methods is the BOP Challenge [1], which is a competition trying to find the best state of the art public method by comparing them on a list of datasets. I will then adapt the chosen algorithm and train it on my dataset.
The current state of the art methods use a combination of classifiers. For example, Cosypose [2] uses three neural networks, and EPOS [3] utilizes six steps for the prediction, including a neural network. Both motioned algorithms have publicly available implementation and excellent results in the BOP Challenge.
For proof of concept, I choose to use four objects with their respective 3D models, and I try to create a training dataset using an RGB camera. Then I switch to the photorealistic rendering of the training images, which is significantly faster and more practical for the amount of training data a neural network requires. This is mainly because it allows for automatic annotation of the objects in the 6D space.
Key words: 6D, Pose Estimation, textureless and reflective objects

Contents

List of Figures

Chapter 1

Introduction

6D pose estimation, particularly using just one RGB image, is a computer vision problem of estimating the pose of all known objects in an image with respect to the camera, namely its position and rotation. In industrial developments, higher demands make for new application scenarios. 6D pose estimation uses several kinds of information to this problem. It obtains texture information, 3D model information, and colour information to measure the 6D pose of objects. The approaches to the 6D pose estimation can be divided to two main categories: 1. Non-learning-based approach, and 2. Learning-based approach.

1.1 Non-learning-based approach

This more traditional approach to the problem dates back to 1963 to [4]. The first practical approaches relied on local image features or template matching and assumed a grayscale or RGB input image. A common approach to the problem is establishing a set of 2D-3D correspondences between the input image and the object model by selecting the correspondences using local image features, such as SIFT [5], and estimating the pose by the PnP-RANSAC algorithm. This approach has demonstrated robustness against occlusion and clutter in the case of objects with distinct and non-repeatable shapes or textures. However, these methods tend to struggle with symmetrical objects as the visible parts of such objects may have multiple fits to the object model. Additionally, methods relying on local image features have a poor performance on texture-less objects because the feature detectors often fail to provide a sufficient number of reliable locations and the descriptors are no longer discriminative enough.

1.2 Learning-based approaches

Learning-based approaches usually use convolutional neural networks, regression or other methods based on deep learning to train a model with training data to then estimate the 6D pose. Recent machine learning-based algorithms have been dominant over traditional ones. In BOP 2020, five methods have outperformed Vidal-Sensors18 [6], a point pair features-based winner of the challenge from 2017 and 2019. But, there are multiple approaches, each with its pros and cons. One of the first problems of CNN based problems was the performance on symmetrical objects. For example, if the method were trained using the squared loss between the ground truth poses and the predicted poses, it would predict the average of the possible poses for an input image, which is often not a valid solution (for example a pipe with the symmetry axis in the middle, would have two correct poses, but the estimate would be perpendicular to them).

1.2.1 Keypoint Approaches

Keypoint-based approaches use two steps: 1. extract the 2D feature points in the input image, and 2. regress the 6D pose results using a PnP algorithm. An example of such an approach would be algorithm BB8 [7], which solved the problem presented by textureless symmetrical objects by restricting the rotation angle; however, it had a problem with occluded objects if it didn't obtain the correct 3D bounding box. PVNet [8] resolved this problem with occlusion by segmenting images into several patches and using those to predict which object they belonged to and where the 2D projections were. PrimA6D [9] further improved on this approach by replacing the 3D bounding boxes with learning orientation-induced primitives. Single-Stage 6D [10] revealed that the typical two steps this approach takes are a weakness because the loss function of the neural network cannot represent the accuracy of 6D pose estimation. Therefore it presented a single-stage 6D pose estimation method that could directly regress the 6D pose based on groups of 3D-to-2D correspondences associated with each 3D object keypoint.

1.2.2 Holistic Approaches

Compared to approaches based on key points, holistic approaches are usually an end-to-end architecture and faster. Examples are PoseNet [11], which utilises a 23 layer deep convolution neural network to estimate the 6D pose. SSD [12] which improved accuracy by combining bounding box priors with the feature maps of different spatial resolutions and SSD-6D [13], which extended it for 6D pose estimation and allowed for easy training and handling of symmetries. Deep-6DPose [14] contributes by decoupling pose parameters into translation and rotation so that the rotation can be regressed via a Lie algebra representation.

This method, however, had poor accuracy when measuring the 6D pose of small or symmetrical objects. PoseCNN [15] introduced a novel loss function that enables it to handle symmetric objects. The method handled well occlusion and symmetric objects in cluttered scenes with RGB or RGB-D images as input.

1.3 Note on depth-based approaches

Many non-learning based methods are focused on just depth information. This is the result of RGB based methods requiring complex algorithms to obtain precise results. At the same time, the development of hardware allows for cheap and straightforward 3D scene information, such as depth cameras and 3D scanners. Compared with 2D information, 3D information preserves the object's original appearance, which is more helpful in measuring the 6D pose [16]. Learning-based methods are usually only RGB based or optionally use depth channel to increase accuracy [2, 17, 18].

1.4 Challenges of 6D pose estimation

As stated before, many methods have difficulty classifying symmetrical and textureless objects, but some have solved this.

The occlusion of a object can cause a method to miss it. Even more so if the occlusion covers some texture which the algorithm may rely on. A common workaround is to separate objects into multiple fragments using the information of visible parts to predict the object's pose, but usually, the more parts of an object that could be seen, the more reliable the results obtained.

Another problem present very reflective objects, which are the most troublesome for pure RGB methods as well approaches measuring depth information using sensors that rely on light since the reflective property of such an object messes up the image.

Background clutter is also pose a challenge to 6D pose estimation methods. Because the target is surrounded by so much useless information, it is difficult to measure the 6D pose directly. But almost in any practical scenario, there will always be clutter in the background. While attempts using masks to cover background exist, like Mask R-CNN [19], the more robust solution when using CNNs is adapting the training dataset to cover for this. This is especially simple when rendering the images, eg. using BlenderProc4BOP [20, 21].

The 6D pose estimation of deformable objects (for example clothes or plants) is a huge challenge in the field because the posture of the objects is unpredictable, and there are many ways for the objects to deform. Thus, many conditions need to be taken into consideration, resulting in pressure on the algorithm and calculation.

To create a dataset images of the objects are captured in different poses in different views. The best matching in the dataset is found when measuring the pose of objects [22].

1.5 6D localization or 6D detection

The difference lies in prior information about the object present in the input image distinguishes two 6D object pose estimation tasks: 6D localization, where the identifiers of present object instances are provided for each image, and 6D detection, where no prior information is provided. 6D detection has computationally more expensive evaluation as many more hypotheses regarding the type of the object need to be evaluated, whereas 6D localization needs only to output the top N pose estimates for an object class.

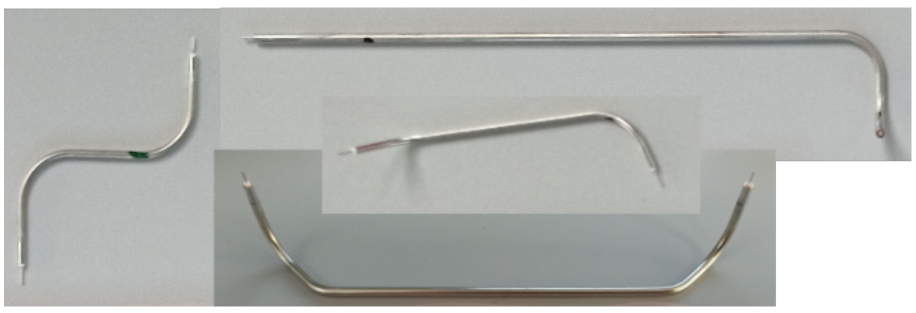
1.6 The task

The given task by the Rohde Schwarz závod Vimperk, s.r.o. is to examine the possible application of such a 6D pose estimation algorithm in an automated machine. A robotic arm would assist this application to grab and move Semirigid RF cables. These cables are small in size, textureless and often symmetrical. The company provided me with four examples and with their corresponding 3D models. However, the final application would have to accommodate about 200 other similar objects. These examples have modified dimensions from the ones used in manufacturing to preserve the confidentiality of the company. They contain a wire in the middle, which is encased by a white insulator further covered by a silver, metallic and somewhat reflective cover, representing most of the objects' surface. These will be delivered to the machine in a black container containing a known amount of objects. The robot will also know the kind of object in prior to the detection. The idea is to use only an RGB camera to tell the robotic arm how to pick up one of the objects and then use it. Since this operation would not be instant, the algorithm can re-evaluate the contents of the box for the next pick up to accommodate for possible misplacement of the remaining objects by the movement of the arm. The selected method should run and learn (if it requires learning) on consumer-level hardware as the company policy forbids sending required (training) data to a 3rd party cloud computing service without a Non-disclosure agreement.

1.7 Conclusions

An ideal method would be accurate, fast, robust, scalable and easy-to-train. However, not all deployments demand all of the above at the same time. For example, a deployment in autonomous driving might not need a precise 6D position of some background objects but will prefer speed. In comparison, a robotic grasping project will need a high degree of accuracy to be able to hit the target but might not require an instantaneous result if the robot needs to wait for another part anyway.

Figure 1.1: Photos of the provided objects



Chapter 2

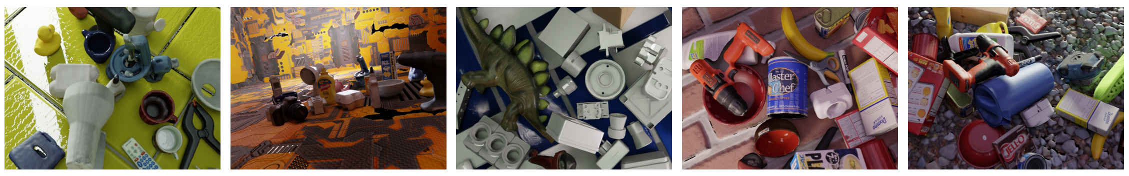
BOP Challenge

The BOP Challenge 2020 is the third in a series of public challenges in the BOP project. The goal of BOP is to continuously report state of the art in 6D object pose estimation. The first challenge was organized in 2017 [23]. The second one from 2019 [24] and the third from 2020 [1] share the same evaluation methodology and leaderboard.

For the BOP Challenge 2020, 50K photorealistic training images for each of the seven core datasets are provided. The images were rendered by BlenderProc4BOP [21], an open-source and lightweight physically-based renderer (PBR) prepared for the BOP Challenge 2020. The objects were rendered inside a cube with randomized surface materials from the CC0 Textures library. This makes the rendering noticeably faster than rendering a complete indoor scene and still allows for great generalization of the background by the neural networks.

BOP allows for an unbiased comparison of publicly available methods. The table at `https://bop.felk.cvut.cz/leaderboards/` provides sorting by performance on each of the datasets as well as per-dataset average called the Core, the average time the method took to estimate poses for all objects in an image, the kind of test image - either RGB, RGB-D or D, and the method's name and date of evaluation. The average time mentioned should not be used to sort the methods alone, as the hardware was not standardized, and every contestant had access to different levels of hardware performance. Even though the challenge has already been concluded, the submission form is still open and should one choose to publicize their results, the system will add them to the leaderboard.

Figure 2.1: Examples of the rendered images



The BOP 2020 was focused on training the methods on only synthetic images because capturing and annotating real training images requires a significant effort. Therefore, the challenge focuses primarily on the more practical scenario where only the object models, that can be used to render synthetic training images, are available at training time. These 3D object models are often available or can be generated at a low cost using KinectFusion-like systems for 3D surface reconstruction.

2.1 Error functions

To evaluate performance, BOP uses three pose-error functions - MSSD (Maximum Symmetry-Aware Surface Distance), MSPD (Maximum Symmetry-Aware Projection Distance) and VSD (Visible Surface Discrepancy). This is because each has its disadvantages and its preferred use case. Both the performance according to each pose-error function and their average are published so everyone can find the best-suited method for a given use case. MSSD is suitable for robotic grasping and MSPD for augmented reality applications. However, because both are calculated over the entire model surface, misalignments of invisible parts are penalized. This may not be desirable for applications such as robotic manipulation with suction cups where only the alignment of the visible part is relevant. VSD is calculated only over the visible object part. It evaluates the alignment of the object shape but not of its colour.

2.2 Datasets

For the training to be effective, the training set needs to exhibit similar characteristics to the target application while maintaining enough variation to prevent overfitting. BOP 2020 features eleven datasets, but only seven of them were selected as core datasets. Each is available in a unified format and includes 3D object models and training and test RGB-D images annotated with ground-truth 6D object poses. The seven core datasets further include photorealistic training images. The test images were captured in scenes with graded complexity, often with clutter and occlusion. All test images are real captured scenes. It is also required that at least 10% of the projected surface area of each object is visible.

Linemod [25] or **LM** features 15 texture-less 3d objects in heavily cluttered scenes, but with little occlusion.

Linemod-Occluded [26] or **LM-O** provides additional ground-truth annotations for all modelled objects in one of the test sets from LM with various levels of occlusion. Because this dataset features the same objects as LM, the PBR-BlenderProc4BOP training images are identical.

T-LESS [27] features 30 industrial objects with no significant texture or discriminative color. The objects exhibit symmetries and mutual similarities in shape and/or size and a couple of objects are a composition of other objects.

ITODD [28] 28 objects captured in realistic industrial setups with only a Gray-D sensor. If we were ranking the datasets instead of the methods in BOP, this would be the worst-performing one, thus the hardest for the methods. It has the lowest average score across the board. Methods using depth channel tend to perform significantly better on this dataset as its missing RGB information.

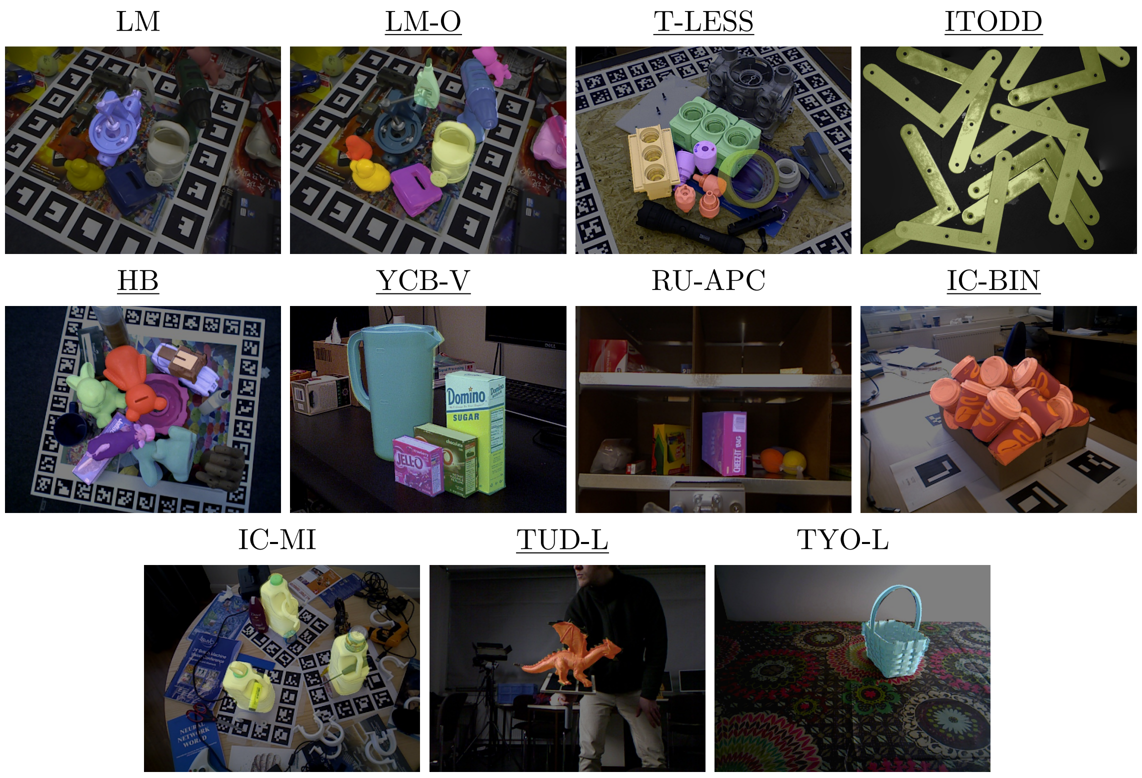
HomebrewedDB [29] or **HB** has 33 objects (toys, household items and industrial objects) in scenes with varying complexity.

YCB-Video [30] or **YCB-V** includes 21 items such as food boxes and other household items. They vary in shapes, sizes, textures, weight and rigidity.

RU-APC [31] is a dataset focused on warehouse pick-and-place applications. It features 14 textured products in images on a cluttered warehouse shelf.

IC-BIN [32] only includes 2 objects, but they appear in multiple locations with heavy occlusion. The objects appear many times in a scene. This dataset is focused on a bin-picking scenario.

Figure 2.2: Examples from each of the eleven BOP datasets. The names of the seven core datasets are underlined. Shown are RGB channels of sample test images which were darkened and overlaid with colored 3D object models in the ground-truth 6D poses.



IC-MI [33] features 6 household objects - Two texture-less and four textured.

TUD-L [24] shows three objects under different lightning conditions.

TYO-L [24] has 21 objects captured on a table with four different table cloths and five different lighting conditions.

2.3 BOP Toolkit

The BOP Challenge also provides BOP Toolkit [34], an open-source Python library for 6D object pose estimation. It implements the evaluation of the pose estimates, rendering of the images, visualization of 6D object poses, i/o operations for the datasets, error functions and calculation of parameters required for the BOP format.

Table 2.1: The leaderboard from BOP Challenge 2020, only RGB methods

Position	**Method**	**Test image**	**ARCore**
3	CosyPose-ECCV20-SYNT+REAL-1VIEW	RGB	0.637
5	CosyPose-ECCV20-PBR-1VIEW	RGB	0.570
10	CDPNv2_BOP20 (RGB-only)	RGB	0.529
13	CDPN_BOP19 (RGB-only)	RGB	0.479
14	CDPNv2_BOP20 (PBR-only & RGB-only)	RGB	0.472
15	leaping from 2D to 6D	RGB	0.471
16	EPOS-BOP20-PBR	RGB	0.457
20	Zhigang-CDPN-ICCV19	RGB	0.353
22	Pix2Pose-BOP20-ICCV19	RGB	0.342
23	Sundermeyer-IJCV19	RGB	0.270
24	SingleMultiPathEncoder-CVPR20	RGB	0.241
25	DPOD (synthetic)	RGB	0.161

Chapter 3

EPOS

EPOS [3] is the method I selected for my task. Upon its release on April 2020 it has outperformed all RGB methods in BOP 2019 [24] on all three datasets used at the time (T-LESS [27], LM-O [26] and YCB-V [30]). Later, in September 2020 in BOP 2020 [1] it was outperformed by the CosyPose [2], CDPN [18] and leaping from 2D to 6D [35] methods. I have chosen EPOS primarily because of the offer of consultations from Tomáš Hodaň, who proposed the method and assisted me in this work. Compared CosyPose, EPOS is also easier to train, as CosyPose used 32 NVIDIA V100 GPUs to train their model for the competition, and I have nowhere close to that amount of performance available. CDPNs model has been trained on an NVIDIA GTX 1070 GPU, but this method teaches one model for each object. This is an approach I would prefer not to use as it would mean having to train and store a large number of models, though it might yield better performance since my in my case, I know for which object the method is looking. Leaping from 2D to 6D uses both the approach of training one model for each object and powerful processing power in the form of the Tesla P40 GPU.

3.1 How EPOS works

Understanding how a method works can reveal its weak and strong points and help understand various problems that might come up during its deployment.

The 3D object models are the only necessary training input of this method. However, to improve the performance beyond this very basic level, training images are required.

3.1.1 Surface Fragments

Since EPOS represents all objects in surface fragments (which is how it handles occluded objects, as explained in 1.4), the very first step is to split every object into N fragments, where N is the number of fragments. The furthest point sampling algorithm finds the fragment centres by iteratively selecting a vertex from the set of all the vertices given by the 3D model most distant from the already chosen vertices. The algorithm starts with the centroid of the object model, which is then discarded from the final set of centres. The team behind EPOS performed an experiment to find out the optimal amount of fragments. They tried 1, 4, 16, 64 and 256 fragments. The performance increased with the number of fragments. Unfortunately, the experiment didn't include tries with more than 256 fragments to see whether the performance improved. The reason for this might be that more fragments require significantly more memory to be available. Also, on T-LESS, the accuracy drops when the number of fragments increases from 64 to 256. So the ideal number of fragments can vary for every dataset. The theory is that because T-LESS includes small objects, the higher amount of fragments causes them to be too small. The network then doesn't have enough training examples for every fragment, and the accuracy decreases. This theory, unfortunately, didn't get tested as it would require creating whole new datasets with tiny and big objects.

3.1.2 Prediction of 2D-3D Correspondences

To correctly predict position in 3D space from a 2D RGB image, the method first needs a probability of an object's surface fragment being visible at a given pixel. But the method actually predicts two probabilities - one for the chance that an object is visible in the given pixel and the second for every fragment of that object being visible. This approach overcomes both the problem of occlusion and symmetry, both of which would cause the combined chance to be close to zero as the following applies:

$$\Pr(fragment, object|pixel) = \Pr(fragment|object, pixel)\Pr(object|pixel)$$

With this approach, a pixel containing an occluded object will keep its probability of containing a specific fragment high. In the case of symmetry, the pixel retains a high chance of containing the object, but the network should split the chance of seeing a specific fragment between all the symmetrical ones.

Next, each surface fragment has a regressor, which predicts the fragments 3D location at the given pixel. This, combined with the prior probabilities, shows the position of an object in 3D space. For all of these predictions, DeepLabv3+ [36] is used, a deep convolutional neural network with an encoder-decoder structure. For m objects, each represented by n surface fragments, the network has 3mn+mn+m+1 output channels (3mn represents the three 3D coordinates of m times n fragments, one mn for their probabilities and the m+1 for the probability of m objects and the one background class).

This network is trained by minimising a loss function designed to increase accuracy on images annotated with ground-truth 6D object poses. Pixels outside the visibility masks of the objects are considered to be the background. The network is provided with only a single corresponding fragment per pixel during training to learn the object symmetries. This replaces the need to identify the visible object parts in each training image and find their fits to the object models.

The pixels the network thinks represent an object are linked with a 3D location on its fragments if the chance exceeds a given threshold. The threshold is relative to the maximum sum of collected locations from all indistinguishable fragments expected to have a similarly high probability. For example, if the object is a sphere, all fragments will be the same and have the same chance (1/number of fragments). This creates a set of correspondences for each instance of an object containing values determining the pixel, its 3D location, and the prediction's confidence. That means every pixel will have zero or more possible matches resulting in a many-to-many relationship between the 2D image locations and the predicted 3D locations.

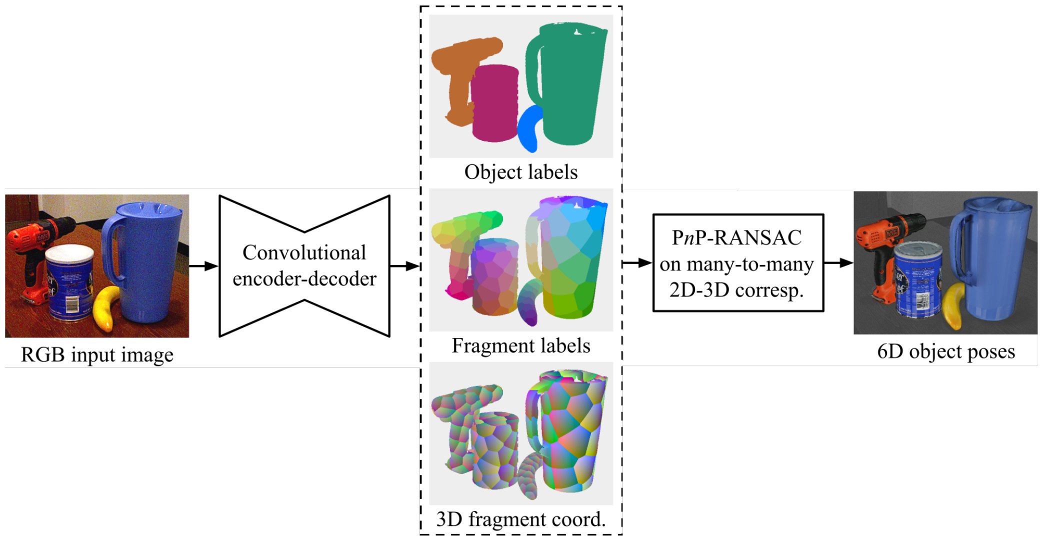
3.1.3 6D Pose Fitting

Now the method has sets of possibly multiple object locations for each object instance, but not more than one can be correct. An efficient variant of PnP-RANSAC [37] algorithm is used to reduce their number. Individual pose hypotheses are proposed sequentially and added to a set of maintained hypotheses. Next the quality of the proposition is calculated. The pose is estimated from a sampled triplet of correspondences by the P3P solver [38], and refined from all inliers by the EPnP solver [39] followed by the Levenberg-Marquardt optimization [40]. If the triplets have collinear 3D locations, they are rejected. The triplets are sampled by PROSAC algorithm [41], which first focuses on correspondences with high confidence and progressively blends to uniform sampling. At each pixel, only the highest quality proposition is considered as only up to one correspondence may be compatible with the hypothesis and the rest only provides alternative explanations. If the pose hypotheses are behind the camera or if the determinant of the rotation matrix is equal to -1, they are discarded. This process repeats until the quality of the new proposition reaches a set threshold, or the number of iterations does. Then the hypothesis with the highest quality is integrated into the set of maintained hypotheses and the process repeats again for the next hypothesis, until there are none left.

3.2 Installation

The entire source code in Python is available from github with instructions on how to install it. Compared to other methods like CosyPose, this process is relatively complicated as the method requires numerous packages and other external repositories to function.

Figure 3.1: Visualisation of the EPOS pipeline



The installation manual mentions a list of libraries and how to install them using the Conda package manager. However, these do not suffice. At least in my case with the latest default installer it did not include several other required packages. Some of them are mentioned as missing in the error codes when running the scripts, but others were quite tricky to find out without previous knowledge about the code. It would be great for anyone in the future who tries to deploy EPOS to have these listed in the manual. I have completed this list of missing dependencies and how to install them and consider it one of the improvements to EPOS in 4.3.3 Next, the manual requires creating environmental variables for paths in the file system by creating a env_vars.sh file in the conda directory (this creates one of the many configuration files EPOS uses). Next, the external dependencies need to be installed - the BOP renderer [42] and Progressive-X [43]. This is where many of the dependencies mentioned before are missing. The other two external dependencies don't require installation. The first is an older version of the TensorFlow-Slim image classification model library [44] and second is the BOP Toolkit [34].

Then it was only a matter of downloading one of the BOP datasets to verify I completed the installation successfully. Creating a list of images using *create_example_list.py* from the *scripts* folder and then using *create_tfrecord.py* to prepare the listed images into a format the method will understand. Next, either download one of the available pre-trained models or train one, and if it performs as expected, the process was successful. This straightforward process is allowed by having parameters for these datasets prepared and hardcoded in the files.

Chapter 4

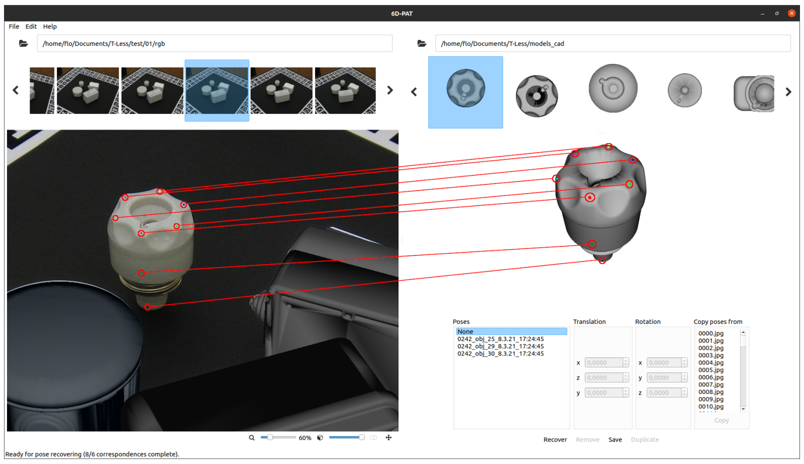
Experiments

The next step would be to create a dataset for my objects and test the performance of the selected method. I started the way that seemed the most straightforward - I built my own primitive rotating platform to capture the objects from multiple angles. I then used a 6D Pose Annotation Tool from `https://github.com/florianblume/6d-pat` to annotate taken images. I have created a minimal dataset of forty real images, which means I took a photo approximately at every nine degrees of rotation around the whole circle at uniform elevation. Then I quickly started searching for a way to use rendering, given how long such a small dataset took to make.

While I had an industrial robotic arm at my disposal, I chose not to use it to capture the photos. While it would have been theoretically possible to calibrate the arm and the camera to annotate the images, I did not find any existing implementation of this approach. I experimented with implementing one myself, but this has proven to be far too ambitious as it would require knowledge of both the robotic arm programming and the camera software. This said, if such a method existed and were publicly available, it would allow for creating great datasets for training and testing 6D pose methods. It would be possible to annotate all photos taken from many various angles and distances while only annotating one scene by hand. This level of automation in creating datasets could be bigger for the industry than the PBR as it would be able to create real data using a bit more human effort and resources compared to PBR (as an industrial robotic arm is more expensive than a computer powerful enough to run Blender).

In the beginning, I was using a Dell workstation with an NVIDIA Quadro P2200 with 5 GB of VRAM and a fresh Ubuntu 20.04.2.0 LTS installation. Because of the lack of VRAM on this card for training, I had to switch to a dual processor Dell PowerEdge with four Quadro P4000 cards with 8 GB of VRAM. These cards provided enough VRAM for the training to run, but the TensorFlow model still didn't fit all the optional variables into it, but it fit the necessary ones so the training worked. If more VRAM were available, it might have computed faster.

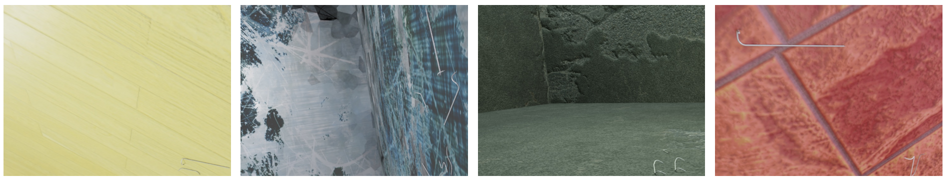
Figure 4.1: Screenshot from the 6D-PAT software showing the annotation process on an example from the T-LESS dataset. While it allows for copying the pose from a previous image, where there is usually only a minor difference, the process is still very time consuming, even for tiny datasets


6D-PAT
File Edit Help
/home/flo/Documents/T-Less/test/01/rgb
/home/flo/Documents/T-Less/models_cad
Poses
None
0242_obj_25_8.3.21_17:24:45
0242_obj_29_8.3.21_17:24:45
0242_obj_30_8.3.21_17:24:45
Translation
Rotation
Copy poses from
Recover
Remove
Save
Duplicate
Copy
Ready for pose recovering (6/6 correspondences complete).

4.1 Creating synthetic training images

At first, I used the provided tools in the BOP Toolkit [34] that installed together with EPOS [3] using the script *render_train_imgs.py*. Images generated by this script didn't look nearly as realistic as the images from BOP, as this script is probably just a leftover from before implementing BlenderProc4BOP [21]. BlenderProc4BOP is an example of using BlenderProc [20], which itself is a procedural Blender pipeline for photorealistic training image generation. Blender is a free and open-source 3D computer graphics software tool set used for creating animated films, visual effects, art, 3D printed models, motion graphics, interactive 3D applications, virtual reality, and computer games. For the rendering, I used the default parameters used for the T-LESS dataset configured by the BOP team.

Figure 4.2: Examples from my first batch of renders from the BlenderProc4BOP using my 3D models but the T-LESS parameters. Notice how little of the area my objects take as they are sometimes barely visible



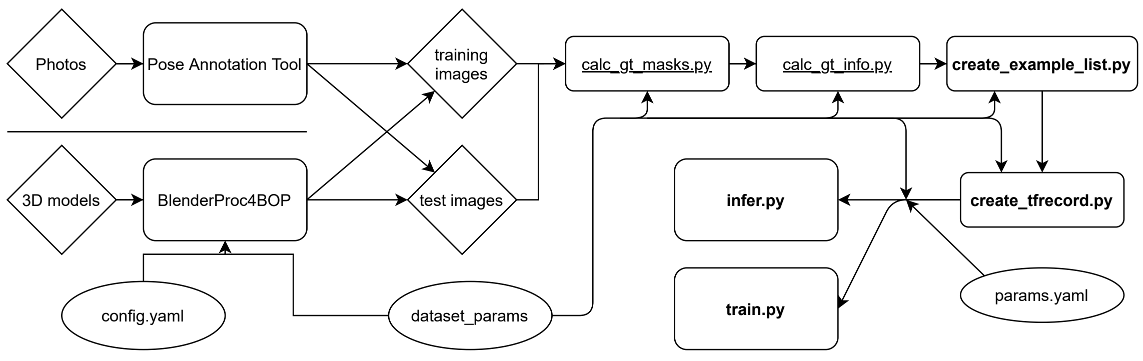
4.2 The First experiment

The scene was set up to create a box with each side long 2 metres, and give all the walls a randomly sampled texture from the CC0 Textures library for generalisation. Then add a light source and drop one of each object on the ground with a physics simulation. As the process seemed to work well, I had more renders made and decided that 50 thousand, like in the BOP Challenge, should suffice. Then the files generated needed to be processed with *calc_gt_masks.py* and *calc_gt_info.py* from the BOP Toolkit to have the correct BOP format that EPOS understands. Next, I needed to create a list from them using the *create_example_list.py* and finally convert them to TFRecord with *create_tfrecord.py.*

The training script wasn't working for me at first. It was returning errors whenever I tried running it - I managed to resolve this by deleting the old checkpoint file from the pre-trained model folder. When the training eventually ran, it soon reached the point where the loss function reported Not a Number (NaN). I managed to create a workaround solution to this by reducing the value of parameter *base_learning_rate* to a tenth of its original value. While that meant that the model might take longer to learn to the same amount as with the original value, it did not get stuck at NaN anymore.

After about one million training steps (a number recommended by Tomáš Hodaň), I tried running the inference script. This script computes the estimated 6D pose from another dataset for testing to evaluate a trained model. This one is able to even visualise the results. I used it with a thousand other rendered test images to get the first results, but the method didn't recognise a single object. This was a major obstacle as I didn't know what could have caused such a result.

Figure 4.3: A diagram showing the entire pipeline from taking the images to finish. It starts with creating training and test datasets, either by synthesising or taking real photos, then processing the datasets into BOP format and finally using these to train or test the model. Diamonds represent source files, rectangles are scripts, and circles are configuration files. If the script's name is underlined, it means it has additional parameters hardcoded in the source file and those that are bold require additional parameters as arguments. I created this diagram to help me keep track of all the possible configuration options



Tomáš Hodaň, fortunately, knew a possible reason - the ground truth maps the network uses for learning are represented only in one-fourth of the original resolution to the model. Ground truth maps are images that only show the shape of the objects, one map for every object in the image they represent. They are created by rendering the 6D position of objects from the training dataset. Given how small my objects appear in the images and their resolution of 720x540 meant that the neural network might not have seen any objects at all. Possible solutions for this problem: Render the images in higher resolution or get the BlenderProc4BOP to render the images from less distance from the objects.

4.3 Improvements

Since the method still did not work, I needed to improve the rendering process by configuring the *config.yaml* file. However, this has proven to be very difficult as the way it stores information is not very user-friendly to configure. It also lacks good documentation, instead recommending trying to use working examples from the creators and other users.

4.3.1 Increasing the resolution

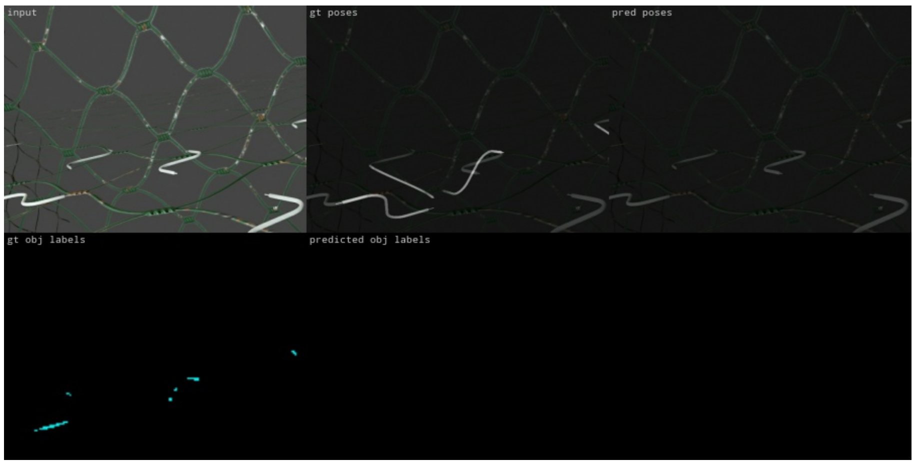
First, I tried to increase the resolution - the renderer actually takes this setting from the *dataset_params* file from the BOP Toolkit, so I did not have to use the *config.yaml*. The first resolution I tried was 1920x1080, understandably the rendering took longer, but now the object took a lot more pixels in the image. The training, however, did not work. At first, I thought the images might now be too large for the model and my available VRAM. I tried gradually lowering the resolution, but it still did not work even when I eventually got to a resolution lower than the original 720x540. The problem must have been somewhere else. I then went through all the configuration settings I was able to find, but I just could not fix the issue.

4.3.2 Rendering from a closer distance

The other way to increase the number of pixels my objects take in a picture, without changing the image's resolution, would be to put the point in space from which the BlenderProc4BOP renders the image (a virtual camera) closer to the objects, or vice versa. This setting is unfortunately located in the mentioned *config.yaml* file. BlenderProc4BOP positions these virtual cameras by randomly sampling them around a central point using the *CameraSampler* module. I set it up to create 10 random poses for the cameras around the centre in diameter of 0,1 to 0,25 units. The camera's rotation should be to look directly at one of the objects with enabled physics (that is only the objects).

Even though it does not seem to aim the camera at any object directly, it manages to capture some of them well enough. I also allowed for some tilt (random rotation around the axis aimed at the object). In the end, the inference script showed that the model saw the objects even at one-fourth of the resolution. However, because my models are really small in diameter, it still only represented them as a few scattered groups of pixels where the line of pixels was the thickest.

Figure 4.4: A grid put together by the inference script. It shows the input picture, the position of objects in the picture according to the labels, the predicted poses, visualisation of the ground truth maps and the predicted labels. Before, the bottom left picture did not show any labels. Thanks to the modifications, it now does, but still not well enough as the objects are still too small. The methods still can not locate a single object in the picture.



4.3.3 Other improvements

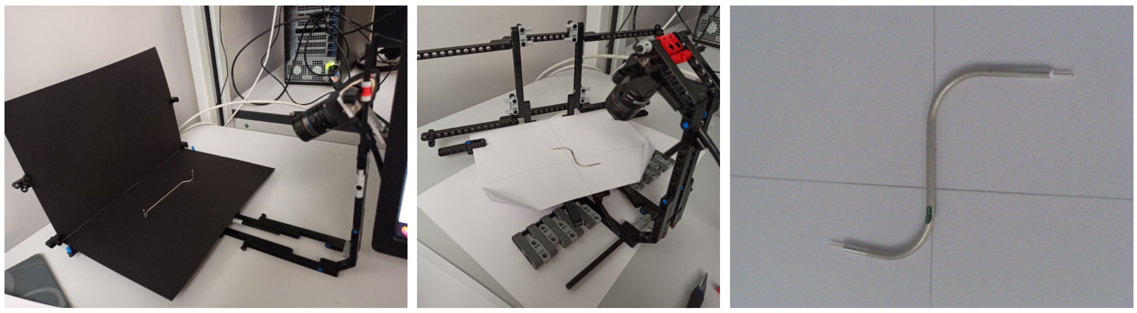
During the time it took to evaluate the previous improvements and thanks to the many examples of the *config.yaml* file I went through, I managed to put together some other improvements to the whole process.

In the beginning, my hardware suffered from random unexpected restarts (I later diagnosed this to be because of a faulty ram stick). This was especially unfortunate to happen while running one of the more time-demanding scripts over the weekend, and those were mainly the *calc_gt_masks.py* and the *calc_gt_info.py* scripts. These two mentioned scripts took around six days to process a larger dataset, and when they restarted, they would start again from the beginning of the dataset, rewriting all the previous progress. This motivated me to implement a simple logic to detect and skip the already processed parts of the dataset.

While monitoring the performance, I noticed that these two scripts used about half of the systems CPU threads at once each. Because their execution order did not matter, I managed to completely parallelize them, running them both simultaneously using all the systems resources. This improvement meant that I could have both parts computed in about 60% of the time they took when performed serially. Even more performance could be gained if they would use the GPUs.

Since I was executing the entire circle of testing different datasets regularly, I thought of automating it. Piece by piece, I put together a bash script that would execute all the scripts independently on me and allow me to save the time wasted between completing one part and manually starting the next. It also allowed me to manage the configuration options of the process better. I also used this script to change which objects and how many of them were rendered. I modified the BlenderProc4BOP run file to accept the kind and amount of objects it put in the scene. This meant I could choose to render an image containing just one kind the object and how many times it appeared. This would allow the neural network to adapt to my scenario where there is always a known amount of one known object. I chose not to use black walls instead of the randomized surface material, which would also fit the deployment, but the generalisation of the background was something I wanted to keep for use on my existing annotated real dataset. Instead, I made the dimensions of the box smaller, only one unit, which is as little as I was able to since the configuration does not accept floats as parameters. I also implemented logging using discord.sh to keep track of the training while I was not connected to the server using bash, webhooks and Discord.

Figure 4.5: Photos of my rotating platform with 2 different backgrounds and a photo taken using it. Later I have added markers from the ARToolKitPlus [45] library to get accurate position and intrinsic parameters of my camera and removed the background vertical plane, as my camera had a narrow field of view and it wasn't visible anyway. This allowed me to take full 360° photos



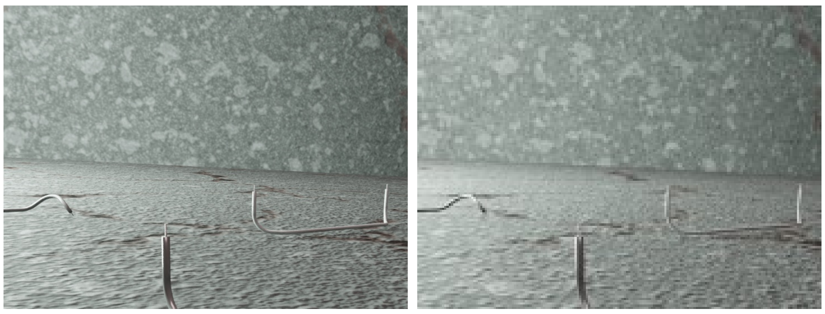
4.4 Conclusions

Even though I implemented numerous improvements, I was not able to achieve a result. Visualisation on other PBR images showed that the method did not even classify a single object, thus getting a 0.000 AR score on the test dataset. Even though the ground-truth maps eventually have shown the network parts of the objects, as 4.4 shows in its bottom-left part, it would only see the thickest parts of objects and learn to classify the rest as background. While my improvements to accuracy and usability were great for working on the method and possibly its eventual deployment, I think my approach failed because I could not increase the resolution. Given how small are my objects in diameter, it is tough to represent them in 180x135 resolution (one-fourth of 720x540, the resolution of the ground truth maps) in a representative shape using the BlenderProc4BOP without increasing the resolution.

Even if I managed to fix this issue, I do not expect great performance due to the nature of my objects. Given how similar they are to the objects used in the ITODD dataset, which is already the most difficult dataset to be included in the BOP Challenge with some of the methods achieving not even a 0.1 of AR score and even very challenging to the best of the overall RGB methods, like CosyPose, I would expect a poor performance even from them.

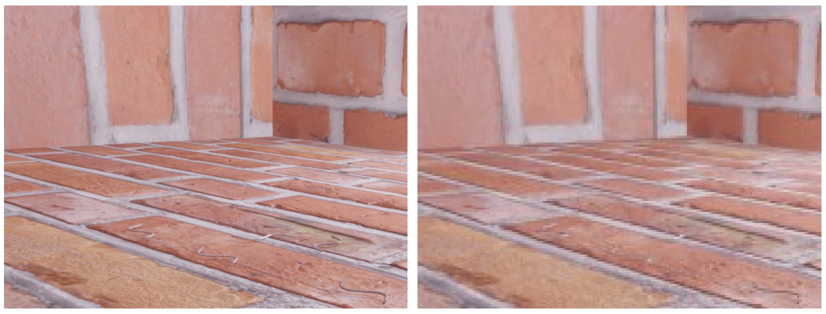
Unfortunately, my improvements have not been added to the public GitHub implementation of EPOS. The first reason is I have lost contact with Tomáš Hodaň, who proposed the method and implemented it, and because the method has already been beaten in performance by other methods. With the speed this field of research is moving forward, the improvements to EPOS are already not very relevant.

Figure 4.6: One of the used training PBR images and on the right is its copy but downscaled to 180x135 resolution to represent the methods 'vision' based on the ground truth maps that use this resolution. Some parts of the objects further from the camera are not even visible, even though this is one of the best images I found in terms of the distance of the objects to the virtual camera



.

Figure 4.7: Another example of the previous, this time with a background with better contrast to the objects, but the objects are further from the camera to better represent a more average picture. The objects take fewer pixels and are hardly even noticeable on the downscaled picture



Conclusions

In this thesis, I have taken on the problem of 6D pose estimation on a couple of very challenging objects. I have started by introducing the problem, along with historical approaches and their various implementations. Then I mentioned the numerous challenges these implementations have to face. I followed by introducing the BOP Challenge, where I compared the current state of the art methods and the datasets they use because I would have to create one myself. The next part was focused on the chosen EPOS method, why I chose to use it, and a brief explanation of how it works and of the installation process. When I did that, I explained my first experiment with the method on my objects. At first, I had to create a training dataset, real or PBR to train the model and evaluate the network's performance. I presented the problems I encountered as well as my solutions to them. While working on these solutions, I implemented various other improvements, some to the general usage and others to the expected performance of my deployment. In the end, I explained why my approach did not work.

4.5 Other solutions

Since my approach did not work, I would like to offer some other solutions using the information I collected while working on this thesis.

One of these possible solutions would be to use a better performing method. I would expect the best performance by the multi-view variant of the CosyPose method. While the single pose variant alone managed to beat EPOS, I explained why I chose not to use it in chapter 3. From my brief experiment with it, it was straightforward to install compared to EPOS as it only uses Python and Yann Labbé et al., who implemented it, managed to automate the process well. For the best performance of this method, certain restrictions would have to be elevated. First, I mentioned that this method could use multiple views of the same pose to increase accuracy. This would contradict the requirement only to use a single RGB camera. However, since said approach did not work, it should be considered as one of the possible options. Using RGB-D cameras would also be especially beneficial because the methods using the depth channel have had generally better performance on the ITODD dataset, which contains the most similar objects to mine, compared to pure RGB methods.

We could achieve even better results if we combined these two proposals. While these solutions would cost more than a single RGB camera, I would expect a great improvement in the performance.

An approach less in dispute with the original assignment would be to use the method introduced by Drost et al. [28]. The benefits of this approach would be a better performance as this is the best performing of all methods on the ITODD dataset and no required training as this method is actually based on point pair features instead of a neural network. This method is even implemented in the HALCON library, which is already commonly used in the company. The disadvantage and the reason why this method was not included in 2.1 is its requirement of RGB-D data, which is again a contradiction of the original assignment, but a reasonable one, due to the possible increase in performance.

Other possible improvements could be gained by collaborating with the someone from the team which proposed the method and modifying it to this specific use - for the BOP Challenge, participants were required to capture poses of all the objects, however since the robotic arm can only handle one object at a time, the method only needs to return the pose of the object it is most confident in. Performance could also be gained by applying texture to the objects. However, this would add a manufacturing step and might not be acceptable due to the application of the objects - for example, if they are visible on the finished product.

Bibliography

1. HODAN, Tomas; SUNDERMEYER, Martin; DROST, Bertram; LABBÉ, Yann; BRACHMANN, Eric; MICHEL, Frank; ROTHER, Carsten; MATAS, Jiri. BOP Challenge 2020 on 6D Object Localization. *CoRR*. 2020, roč. abs/2009.07378. Available from arXiv: `2009.07378`.
2. LABBÉ, Yann; CARPENTIER, Justin; AUBRY, Mathieu; SIVIC, Josef. CosyPose: Consistent multi-view multi-object 6D pose estimation. *CoRR*. 2020, roč. abs/2008.08465. Available from arXiv: `2008.08465`.
3. HODAN, Tomas; BARATH, Daniel; MATAS, Jiri. EPOS: Estimating 6D Pose of Objects with Symmetries. *CoRR*. 2020, roč. abs/2004.00605. Available from arXiv: `2004.00605`.
4. ROBERTS, Lawrence. *Machine Perception of Three-Dimensional Solids*. 1963. ISBN 0-8240-4427-4.
5. LOWE, D.G. Object recognition from local scale-invariant features. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. 1999, sv. 2, 1150–1157 vol.2. Available from DOI: `10.1109/ICCV.1999.790410`.
6. VIDAL, Joel; LIN, Chyi-Yeu; LLADÓ, Xavier; MARTÍ, Robert. A Method for 6D Pose Estimation of Free-Form Rigid Objects Using Point Pair Features on Range Data. *Sensors*. 2018, roč. 18, č. 8. ISSN 1424-8220. Available from DOI: `10.3390/s18082678`.
7. RAD, Mahdi; LEPETIT, Vincent. BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth. *CoRR*. 2017, roč. abs/1703.10896. Available from arXiv: `1703.10896`.
8. PENG, Sida; LIU, Yuan; HUANG, Qixing; BAO, Hujun; ZHOU, Xiaowei. PVNet: Pixel-wise Voting Network for 6DoF Pose Estimation. *CoRR*. 2018, roč. abs/1812.11788. Available from arXiv: `1812.11788`.
9. JEON, MyungHwan; KIM, Ayoung. PrimA6D: Rotational Primitive Reconstruction for Enhanced and Robust 6D Pose Estimation. *CoRR*. 2020, roč. abs/2006.07789. Available from arXiv: `2006.07789`.
10. HU, Yinlin; FUA, Pascal; WANG, Wei; SALZMANN, Mathieu. Single-Stage 6D Object Pose Estimation. *CoRR*. 2019, roč. abs/1911.08324. Available from arXiv: `1911.08324`.

11. KENDALL, Alex; GRIMES, Matthew; CIPOLLA, Roberto. Convolutional networks for real-time 6-DOF camera relocalization. *CoRR*. 2015, roč. abs/1505.07427. Available from arXiv: `1505.07427`.

12. LIU, Wei; ANGUELOV, Dragomir; ERHAN, Dumitru; SZEGEDY, Christian; REED, Scott E.; FU, Cheng-Yang; BERG, Alexander C. SSD: Single Shot MultiBox Detector. *CoRR*. 2015, roč. abs/1512.02325. Available from arXiv: `1512.02325`.

13. KEHL, Wadim; MANHARDT, Fabian; TOMBARI, Federico; ILIC, Slobodan; NAVAB, Nassir. SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again. *CoRR*. 2017, roč. abs/1711.10006. Available from arXiv: `1711.10006`.

14. DO, Thanh-Toan; CAI, Ming; PHAM, Trung; REID, Ian D. Deep-6DPose: Recovering 6D Object Pose from a Single RGB Image. *CoRR*. 2018, roč. abs/1802.10367. Available from arXiv: `1802.10367`.

15. XIANG, Yu; SCHMIDT, Tanner; NARAYANAN, Venkatraman; FOX, Dieter. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. *CoRR*. 2017, roč. abs/1711.00199. Available from arXiv: `1711.00199`.

16. ZHANG, Zihao; HU, Lei; DENG, Xiaoming; XIA, Shihong. Weakly Supervised Adversarial Learning for 3D Human Pose Estimation from Point Clouds. *IEEE Transactions on Visualization and Computer Graphics*. 2020, roč. 26, č. 5, pp. 1851–1859. Available from DOI: `10.1109/TVCG.2020.2973076`.

17. PARK, Kiru; PATTEN, Timothy; VINCZE, Markus. Pix2Pose: Pixel-Wise Coordinate Regression of Objects for 6D Pose Estimation. *CoRR*. 2019, roč. abs/1908.07433. Available from arXiv: `1908.07433`.

18. LI, Zhigang; WANG, Gu; JI, Xiangyang. CDPN: Coordinates-Based Disentangled Pose Network for Real-Time RGB-Based 6-DoF Object Pose Estimation. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.

19. HE, Kaiming; GKIOXARI, Georgia; DOLLÁR, Piotr; GIRSHICK, Ross B. Mask R-CNN. *CoRR*. 2017, roč. abs/1703.06870. Available from arXiv: `1703.06870`.

20. DENNINGER, Maximilian; SUNDERMEYER, Martin; WINKELBAUER, Dominik; ZIDAN, Youssef; OLEFIR, Dmitry; ELBADRAWY, Mohamad; LODHI, Ahsan; KATAM, Harinandan. BlenderProc. *CoRR*. 2019, roč. abs/1911.01911. Available from arXiv: `1911.01911`.

21. HODAN, Tomas; VINEET, Vibhav; GAL, Ran; SHALEV, Emanuel; HANZELKA, Jon; CONNELL, Treb; URBINA, Pedro; SINHA, Sudipta N.; GUENTER, Brian. Photorealistic Image Synthesis for Object Instance Detection. *CoRR*. 2019, roč. abs/1902.03334. Available from arXiv: `1902.03334`.

22. CAPORALI, Alessio; PALLI, Gianluca. Pointcloud-based Identification of Optimal Grasping Poses for Cloth-like Deformable Objects. In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2020, sv. 1, pp. 581–586. Available from DOI: `10.1109/ETFA46521.2020.9211879`.

23. HODAN, Tomas; MICHEL, Frank; BRACHMANN, Eric; KEHL, Wadim; BUCH, Anders Glent; KRAFT, Dirk; DROST, Bertram; VIDAL, Joel; IHRKE, Stephan; ZABULIS, Xenophon; SAHIN, Caner; MANHARDT, Fabian; TOMBARI, Federico; KIM, Tae-Kyun; MATAS, Jiri; ROTHER, Carsten. BOP: Benchmark for 6D Object Pose Estimation. *CoRR*. 2018, roč. abs/1808.08319. Available from arXiv: `1808.08319`.

24. HODAŇ, T.; BRACHMANN, E.; DROST, B.; MICHEL, F.; SUNDERMEYER, M.; MATAS, J.; ROTHER, C. *BOP Challenge 2019*. 2019. `https://bop.felk.cvut.cz/media/bop_challenge_2019_results.pdf`.

25. HINTERSTOISSER, S.; LEPETIT, V.; ILIC, S.; HOLZER, S.; BRADSKI, G.; KONOLIGE, K.; NAVAB, N. Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes. *ACCV*. 2012.

26. BRACHMANN, Eric; KRULL, Alexander; MICHEL, Frank; GUMHOLD, Stefan; SHOTTON, Jamie; ROTHER, Carsten. Learning 6D object pose estimation using 3D object coordinates. *ECCV*. 2014.

27. HODAŇ, Tomáš; HALUZA, Pavel; OBDRŽÁLEK, Štěpán; MATAS, Jiři; LOURAKIS, Manolis; ZABULIS, Xenophon. T-LESS: An RGB-D Dataset for 6D Pose Estimation of Texture-less Objects. *IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2017.

28. DROST, Bertram; ULRICH, Markus; BERGMANN, Paul; HARTINGER, Philipp; STEGER, Carsten. Introducing MVTec ITODD – A dataset for 3D object recognition in industry. *ICCVW*. 2017.

29. KASKMAN, Roman; ZAKHAROV, Sergey; SHUGUROV, Ivan; ILIC, Slobodan. HomebrewedDB: RGB-D Dataset for 6D Pose Estimation of 3D Objects. *ICCVW*. 2019.

30. XIANG, Yu; SCHMIDT, Tanner; NARAYANAN, Venkatraman; FOX, Dieter. PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. *RSS*. 2018.

31. RENNIE, Colin; SHOME, Rahul; BEKRIS, Kostas E; DE SOUZA, Alberto F. A Dataset for Improved RGBD-Based Object Detection and Pose Estimation for Warehouse Pick-and-Place. *RA-L*. 2016.

32. DOUMANOGLOU, Andreas; KOUSKOURIDAS, Rigas; MALASSIOTIS, Sotiris; KIM, Tae-Kyun. Recovering 6D Object Pose and Predicting Next-Best-View in the Crowd. *CVPR*. 2016.

33. TEJANI, Alykhan; TANG, Danhang; KOUSKOURIDAS, Rigas; KIM, Tae-Kyun. Latent-class hough forests for 3D object detection and pose estimation. *ECCV*. 2014.

34. HODAŇ, T.; SUNDERMEYER, M. *BOP Toolkit*. 2020. `https://github.com/thodan/bop_toolkit`.

35. LIU, Jinhui; ZOU, Zhikang; YE, Xiaoqing; TAN, Xiao; DING, Errui; XU, Feng; YU, Xin. Leaping from 2D Detection to Efficient 6DoF Object Pose Estimation. *ECCVW*. 2020.

36. CHEN, Liang-Chieh; ZHU, Yukun; PAPANDREOU, George; SCHROFF, Florian; ADAM, Hartwig. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. *CoRR*. 2018, roč. abs/1802.02611. Available from arXiv: `1802.02611`.

37. BARATH, Daniel; MATAS, Jiri. Graph-Cut RANSAC. *CoRR*. 2017, roč. abs/1706.00984. Available from arXiv: `1706.00984`.

38. KNEIP, Laurent; SCARAMUZZA, Davide; SIEGWART, Roland. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In: *CVPR 2011*. 2011, pp. 2969–2976. Available from DOI: `10.1109/CVPR.2011.5995464`.

39. LEPETIT, Vincent; MORENO-NOGUER, Francesc; FUA, Pascal. EPnP: An accurate O(n) solution to the PnP problem. *International Journal of Computer Vision*. 2009, roč. 81. Available from DOI: `10.1007/s11263-008-0152-6`.

40. MORÉ, JorgeJ. The Levenberg-Marquardt algorithm: Implementation and theory. In: WATSON, G.A. (ed.). *Numerical Analysis*. Springer Berlin Heidelberg, 1978, sv. 630, pp. 105–116. Lecture Notes in Mathematics. ISBN 978-3-540-08538-6. Available from DOI: `10.1007/BFb0067700`.

41. CHUM, O.; MATAS, J. Matching with PROSAC - progressive sample consensus. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005, sv. 1, 220–226 vol. 1. Available from DOI: `10.1109/CVPR.2005.221`.

42. HODAN, Tomas; SUNDERMEYER, Martin. *BOP renderer* [`https://github.com/thodan/bop_renderer`]. GitHub, 2020.

43. BARATH, Daniel; MATAS, Jiri. Progressive-X: Efficient, Anytime, Multi-Model Fitting Algorithm. *CoRR*. 2019, roč. abs/1906.02290. Available from arXiv: `1906.02290`.

44. GUADARRAMA, Sergio. *BOP renderer* [`https://github.com/tensorflow/models/tree/master/research/slim`]. GitHub, 2021.

45. WAGNER, Daniel; SCHMALSTIEG, Dieter. ARToolKitPlus for Pose Tracking on Mobile Devices. In: 2007.