UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

On 3D Pose Estimation for XR. Classic Approaches vs Neural Networks

Author:
Vitalii VOROBIOV

Supervisor:
Dr. Taras KHAPKO

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2021

Declaration of Authorship

I, Vitalii VOROBIOV, declare that this thesis titled, "On 3D Pose Estimation for XR. Classic Approaches vs Neural Networks" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

On 3D Pose Estimation for XR. Classic Approaches vs Neural Networks

by Vitalii VOROBIOV

Abstract

The primary purpose of this paper is to investigate different approaches for 3D object pose estimation, which uses neural networks, and for model-based tracking - an innovative solution that builds upon a combination of known matching and pose estimation algorithms and to propose the one which will be more suitable for our problem. Object tracking is one of the critical problems for many applications on AR/MR devices that use object pose estimation to create an immersive experience by combining the physical world with virtually generated objects. The main limitation of our application is that it must work in real-time and be efficient enough to run on devices with weak computing power (*e.g.*, RealWear HMT-1).

Acknowledgements

During the writing of this work, I gained much valuable experience for myself, but all this would have been impossible without my supervisor Taras Khapko. So first, I want to thank him for being my mentor, for always helping and inspiring me during this challenging journey. I am very grateful to the Ukrainian Catholic University for the opportunity to spend the best four years of my studies here, for the knowledge and experience gained, for the opportunity to realize myself professionally, as well as for new friends who supported me during challenging moments. Also, I want to thank my parents, who have always believed in me and supported my every choice.

Contents

List of Figures

List of Abbreviations

XR	**E**xtended **R**eality
VR	**V**irtual **R**eality
AR	**A**ugmented **R**eality
MR	**M**ixed **R**eality
SIFT	**S**cale **I**nvariant **F**eature **T**ransform
SURF	**S**peed **U**p **R**obust **F**eature
SDK	**S**oftware **D**evelopment **K**it
ADD	**A**verage **D**istance of **M**odel **P**oints
KLT	**K**anade **L**ucas **T**omasi **T**racker
PnP	**P**erspective a**n**d **P**oint

Chapter 1

Introduction

In the modern world, technologies are evolving rapidly. While the last decade saw a shift from bulkier personal computers to laptops and later on mobile phones, the next logical step in our personal devices evolution seems to be connected with "Extended Reality." Extended Reality **[21]** or shortened XR refers to all real-and-virtual environments generated by computer graphics and wearables. The "X" in XR is simply a variable that can stand for any letter from the list of technologies that XR can offer. XR is the umbrella term that encompasses various forms of computer-altered reality, including Augmented Reality (AR), Mixed Reality (MR), and Virtual Reality (VR). So, we already know what XR means and that we have to deal here with three different technologies. But what exactly the difference between them? There are two key factors that help distinguish these concepts. The first is how a user interacts with this technology and the second one is how immersive this technology is (or, in other words, how much virtuality it brings to us).

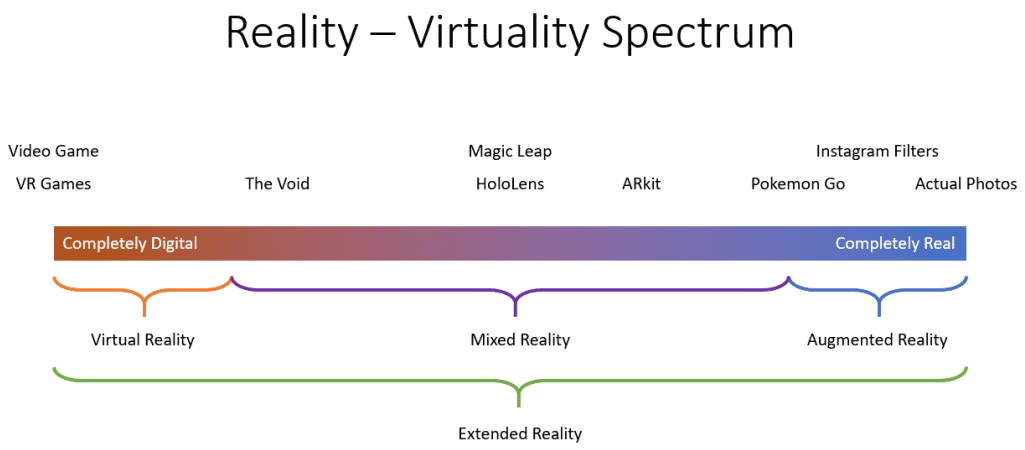


FIGURE 1.1: Reality - Virtuality Spectrum **[20]**

From the image above, we can see that the closest to reality and most familiar is Augmented Reality. Next is Mixed Reality and then finally Virtual Reality. Let us take a closer look at the three of these separately.

Augmented reality does precisely what it sounds like: reality, enhanced with digital components. The most commonly used AR applications these days rely on smartphones to showcase the digitally augmented world: users can activate a smartphone's camera, see the world around them on the screen, and rely on AR application to enhance that view in any number of ways: Superimposing images, Adding

real-time directions, Inserting labels, Changing colors.

Mixed Reality (MR) is when natural and virtual worlds merge to produce new environments and visualizations, where physical and digital objects co-exist and interact in real-time. It is based on advancements in CV, ML, graphics, display technology, and novel input systems. At first, it may seem that AR and MR are the same, but the key difference is that in AR, you can not interact with virtual objects, which appear only as an overlay, but in Mixed reality, they persist in the user's environment, becoming a part of it and boosting overall immersion.

Virtual reality **[22]** (VR) refers to a fully computer-generated simulation in which the user can interact within an artificial three-dimensional environment using wearable devices, such as special VR Headsets fitted with sensors. Unlike Augmented Reality and Mixed Reality, VR does not interact with the natural world at all (unless you hit your table with a controller while playing Beat Saber). This technology is most suitable for games because it allows you to immerse yourself in the gameplay entirely.

1.1 Problem

Nowadays, XR technologies develop very quickly. Many ordinary users and large companies are starting to integrate AR/VR/MR technologies into their business. This trend has been primarily seen in such domains as Manufacturing, Healthcare, Automotive, Marketing, Oil & Gas. Many manufacturing companies (BMW, JABIL, Airbus) already use AR/MR technologies in their manufacturing processes for staff training and increased production speed. Moreover, many inspection companies (Lufthansa Technik, SGS SA, Bureau Veritas S.A) provide inspection and repair of some machines or devices, and they also want to use AR/MR technologies to increase employee productivity and improve the quality of work performed. The problem they most often encounter and which is critical for them to solve is live object detection and tracking on AR/MR devices. In many cases, it is necessary to create different types of virtual training that help employees better understand and perform their work, while also there are many use cases for it in the day-to-day work with different complicated machinery. The ability to see and work with virtual overlay aligned with this machinery helps prevent mistakes and provides additional context in the form of virtual tooltips, instructions, or even remote assistance.

1.2 Goal

The main goal of this work is to analyze existing and develop a new system for detecting objects provided their 3D models and then tracking them in time. Here detection means estimating their pose (6DoF, Translation over X, Y and Z, Rotation over X, Y, Z) in some reference world coordinate system at a given point in time. The solution needs to be cross-platform to support both mobile devices and tablets (Android, IOS), Mixed Reality headsets (Magic Leap, Microsoft HoloLens), Augmented Reality Headsets (Realwear HMT-1). Some of those devices have limited computing power, so the solution needs to be optimized for their needs.

1.3 Constraints

There are several limitations imposed on our application. The first one is that the application should be able to run in real-time to provide a smooth experience to the user. The excellent result for this step is to track the object with a frame rate of at least 15 frames per second. The application should be optimized to run not only on powerful devices such as Magic Leap or Hololens but also on devices with limited computing power such as RealWear HMT-1. The solution needs to be object-invariant, which means that the whole system must work correctly for any type of object, provided its 3D model is available. The new object adaptation process should not take a long time or somehow change the system architecture.

Chapter 2

Related Works and Background Information

There are two main types of optical tracking methods [**1**]: marker-based tracking and marker-less tracking. Marker-based tracking relies on different types of artificial patterns present on the scene to establish a good object or camera tracking. Marker-less tracking, in turn, uses naturally generated features from the scene. By directly comparing these two approaches, the first one is easier to implement and, in general, is faster and more robust. However, it is not always possible to use pre-prepared markers, which is the main drawback of this type of approach. The second one is more adaptive to the new environment but is more complicated in development and computationally expensive.

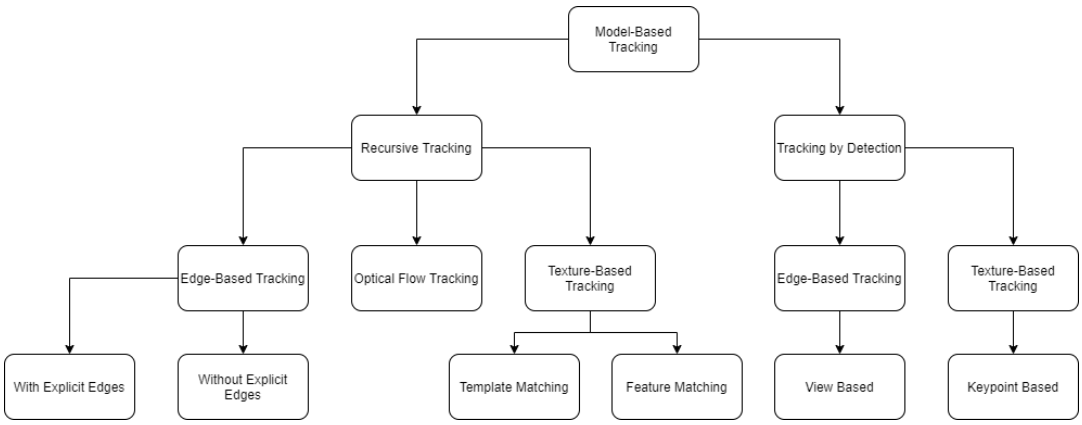


FIGURE 2.1: Model-Based Tracking Types

When it comes to object tracking based on its 3D model, what is referred to as model-based tracking, it can be classified into two categories: recursive tracking and tracking by detection. In recursive tracking, we use data (either camera pose or position of scene objects) from previously processed frames to estimate the new values for the current frame. This approach is not computationally expensive and requires less processing power. Tracking by detection does not rely on previously computer information and so is more computationally expensive, but it can lead to more accurate results.

Recursive tracking itself can be classified into three categories: edge-based, optical flow, and texture-based tracking. In edge-based tracking, a wireframe from the 3D model is used to match natural world objects' edges. Optical flow tracking uses relative information from the movement of the object's projection in the image plane. Texture-based tracking uses different features (e.g., SIFT, SURF, ORB, etc...) to find

correspondences between consecutive frames and matching key points to estimate the object (or camera) position.

In this work, we focus on marker-less recursive tracking, where both edge-based and texture-based detection methods are used to find initial object pose before relying on texture-based camera tracking.

2.1 The Camera Model

A camera is an essential tool in computer vision. Using a camera, we can record visual information, which is then processed by computer vision algorithms. Different models are used to represent the camera imaging process, but the most widely used in computer vision, and the one we used in this work is the pinhole camera model.

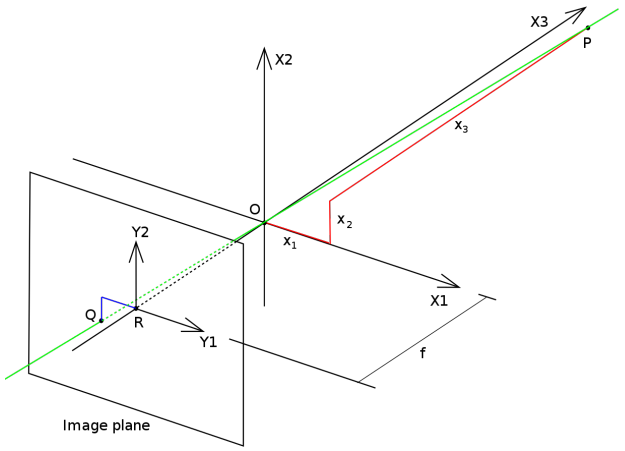


FIGURE 2.2: Pinhole Camera Model

In this camera model, the image plane is the projection of the world to the camera. The aperture of the camera is its center which is referred to as the pinhole O. The focal length f is the distance between the pinhole and the image plane. The work of this camera model is simple. Let us take, for example, world point p. It emits a light ray through the aperture (pinhole) of our camera, which passing through it, is reflected on the image plane Q. In this case, the output image will be rotated from left to right and from top to bottom relative to the original image. This work will use a slightly simplified version of this camera model, namely where the image will be already correctly rotated and located in front of our camera at a focal length distance.

From the image above, we can see that this camera model has several coordinate systems:

1. World coordinate system X_w Y_w Z_w. This the coordinate system of the real world.

2. Camera coordinate system X_c Y_c Z_c. The center of the system is the pinhole of the camera.

3. Image plane coordinate system C_x C_y. The origin of the system is defined on the center of the image plane.

4. Pixel coordinate system x_i y_i. Unlike the image plane coordinate system, its center is at the top left corner of the image plane. This coordinate system is most commonly used for image processing.

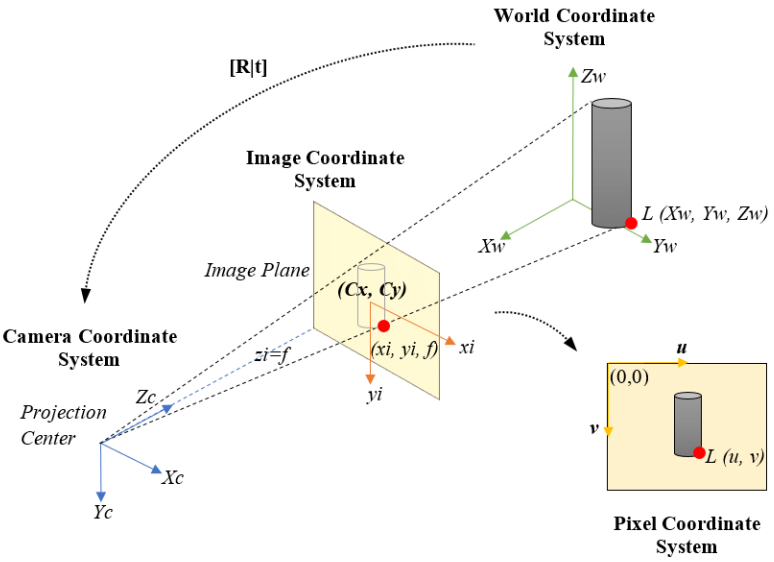


FIGURE 2.3: Pinhole Camera Model

The conversion of 3D point in the world space to the 2D point on the image plane is determined by this formula:

$$P_c = K \times [R|t] \times P_w \tag{2.1}$$

Where K is a camera matrix with its intrinsic parameters. Focal length f_x, f_y, principal point offset x_0, y_0 and shear s. In our case, $f_x = f_y$, because we use a pinhole camera model, however in practice, these values can differ for several reasons **[12]**:

- Flaws in the digital camera sensor
- Image non-uniform scale in post-processing
- Camera's lens distortion
- Anamorphic camera format, which shrinks widescreen scene into the standard.
- Errors in camera calibration

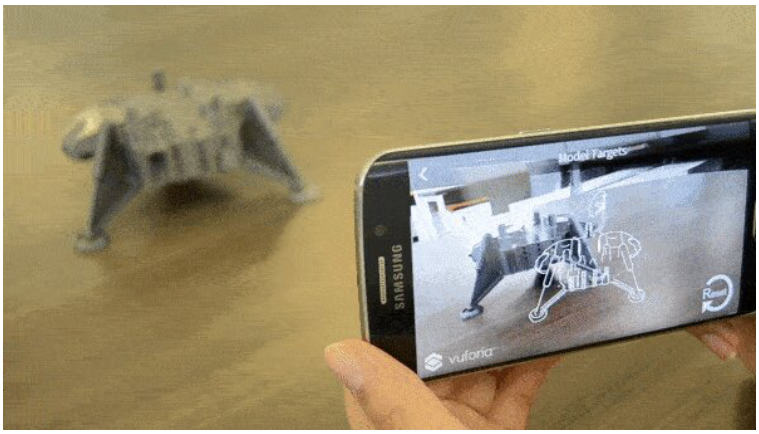
$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Translation}} \times \underbrace{\begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Scaling}} \times \underbrace{\begin{pmatrix} 1 & s/f_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Shear}} \tag{2.2}$$

R is a 3×3 rotation matrix, and t is a 3D translation vector representing the relative rotation and translation between the world coordinate system and the camera coordinate system. $[R|t]$ are the so-called external parameters of the camera. So, the key to fast and robust pose estimation and 3D tracking is correct solving of $[R|t]$.

2.2 Commercial Solutions

2.2.1 Vuforia

Vuforia is a company that PTC acquired in 2016. It develops a platform to create cutting-edge augmented reality experiences for both handheld devices and digital eyewear. Vuforia has its own product named "Vuforia Engine" an Augmented Reality SDK that provides much functionality, but the most interesting for us is the "Model Targets" functionality. It enables apps built with Vuforia Engine to recognize and track particular objects in the real world based on the object's shape. This solution can work with different types of objects, from home appliances and toys to vehicles and large-scale industrial equipment.



(a)

(b)

FIGURE 2.4: (a) Object detection process (b) Object tracking with overlay

We can not tell for sure what algorithms does "Vuforia" use, but we can make certain assumptions based on their model generation process. There are several steps in a model generation:

- Choosing the correct object orientation.
- Selecting the measuring units (*e.g.* millimeters, centimeters, meters).
- Object coloring.

- Test for enough amount of vertices.
- Model surface type selection.
- Object motion type selection.
- View angle selection.

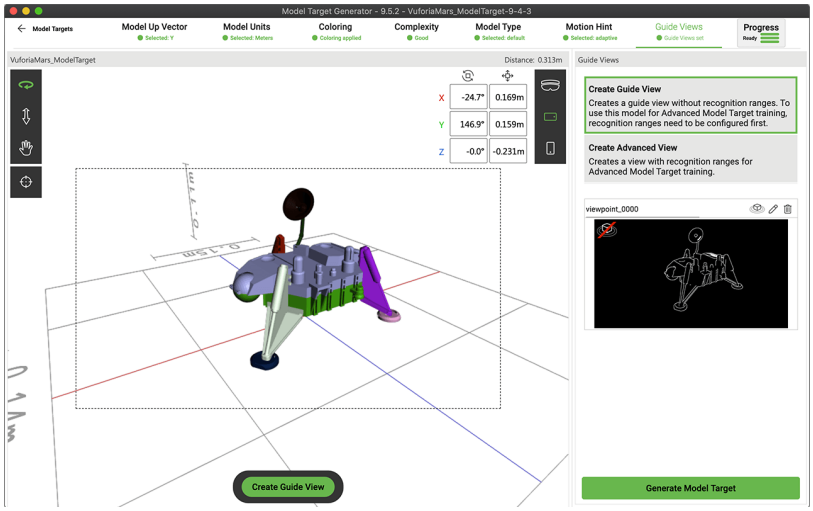


FIGURE 2.5: Model generation process

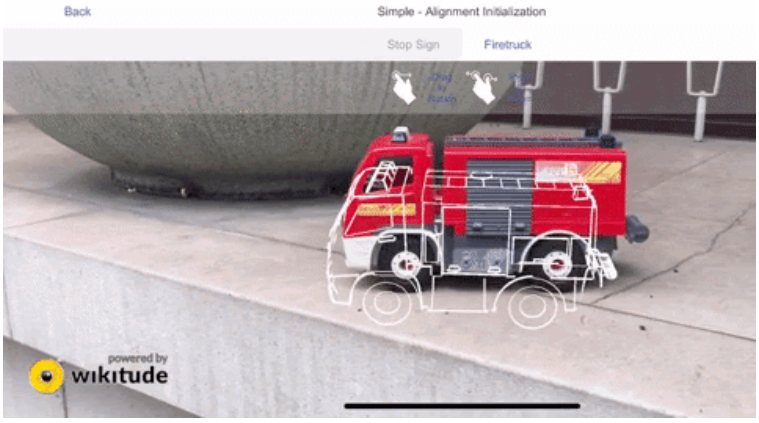
Knowledge of the correct object orientation and measuring units helps with point conversion from the 3D model to the real-world object. The object coloring step and the selection of the surface type tell us that the texture of the 3D model is necessary for the detection or tracking part of their algorithm. So, they can use some feature matching to get correspondences between 3D model points and points on the real-world object. Regarding the object surface type, different features (e.g., SIFT, SURF) could be used to make an algorithm work in the same way as with matte and metal surfaces. Also, the 3D model should have a significant amount of vertices to generate enough tracking features. The motion type can have two values, either a moving or a static object. The selection of a static object can simplify the tracking process of the object by using camera pose estimation algorithms (pose estimation accuracy can be improved by using additional sensors such as a gyroscope and accelerometer). The view angle selection step is used to generate a projection of a 3D model wireframe onto the camera image plane, which is then used in the application to make initial object pose detection. From this step, we can assume that the object detection process in Vuforia is using some edge-based method.

2.2.2 Wikitude

Wikitude is another company that has its own platform for the creation of augmented reality experiences. They provide Wikitude Augmented Reality SDK that has a similar feature as Vuforia has. Its name is "3D Model Object Targets". Unlike Vuforia, this solution can not directly work with 3D models. To do this, you need to send the desired 3D model to the developers of this solution, and they will manually generate the so-called point cloud map, which you can then use in your application. However, this solution can automatically generate a point cloud map

from 2D images of the 3D object. So, for this step, some implementation of structure from motion algorithm is used. Then this point cloud map is used for object tracking.

(a)



(b)

FIGURE 2.6: (a) Wikitude edge-based detection (b) Object tracking step with 3D model overlay

For the initial detection step, the process is similar to Vuforia's one. The object's wireframe is generated from some point of view, and then the edge-based method is used to estimate the initial object position.

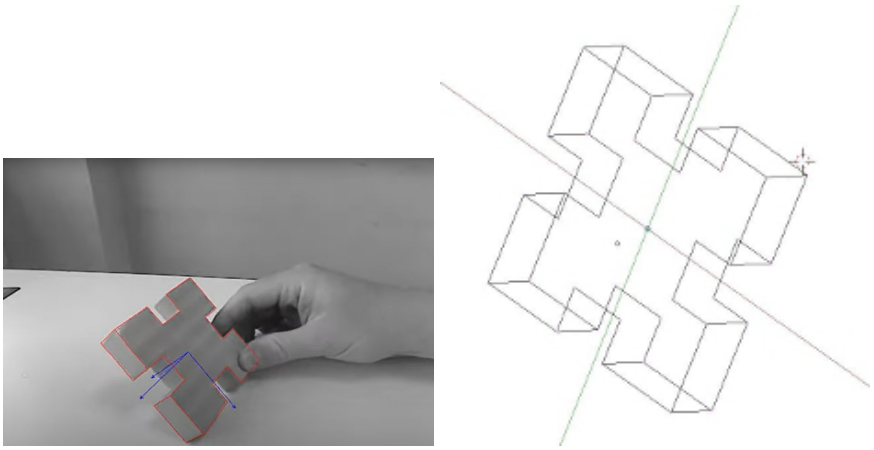
2.2.3 ViSP

ViSP (Visual Servoing Platform) is an open-source, cross-platform library developed by a team of researchers from Inria university that allows to develop applications using visual tracking methods. The library is divided into several modules, and one of them is the "Trackers" module, which has the implementation of model-based tracking for 3D objects. The algorithm is divided into two steps. The first one is the detection of some features. The features can be of two types:

- Local features of the image (e.g., SIFT, SURF)
- Moving edges along the normal to the projection of the 3D model

The second step is tracking of previously detected features. The edge-based tracking method (similar to the RAPiD algorithm) or KLT keypoints tracker is used based on

features. The last step of the approach is the estimation of object pose by matching the tracked features with the projection of the 3D model.



(a) (b)

FIGURE 2.7: (a) Example of object tracking using ViSP (b) 3D Model example used for tracking

2.2.4 Conclusion

So from the analysis of commercial solutions, we can conclude that the vast majority of companies solve the problem of object pose estimation by using classic computer vision algorithms. This decision can be explained by the fact that the classic approaches are more robust, require less computational power than the neural networks, and are easily generalized. However, in order to be able to confirm this statement, in the next section, we will analyze the existing classical approaches and neural networks.

2.3 Classic Computer Vision

2.3.1 RAPiD

RAPiD [**3**] (Real-time Attitude and Position Determination) is an object tracker (if you include it in our classification, it will take the place of a recursive edge-based tracker), which Harris and Stennett proposed. The main advantage of this approach is that it can run in real-time, which is crucial for our application. This algorithm can estimate the relative pose of the object by analyzing the displacement of image edges between two consecutive frames. The main idea is not to compare the raw edges of the object but to use points that lie on them. This technique is "points sampling". Firstly, all edges should are extracted from the 3D model, then a series of points (also called control points) is sampled on each edge. Since this is a recursive tracking algorithm, we need to utilize the estimated pose from the previous frame to estimate the object position in the new frame. Assume that we have an estimated pose for the previous frame that can be expressed by two variables: the rotation matrix R and translation vector t. Then the control point P_w (which is in the world coordinate system) for the previous frame will have camera coordinate such as

$$P_c = RP_w + t \tag{2.3}$$

and its projection point p on the image plane could be obtained from previous equation 2.1. Now for current frame after a relative rotation ΔR and translation Δt our point P_c could be represented as $P_c^{'} = \Delta R \cdot RP + t + \Delta t$. ΔR could be approximated as $\Delta R \approx I + \Omega$, where Ω is a skew-symmetric matrix that includes three parameters. So, $P_c^{'}$ and $p^{'}$ could be represented as a function with six parameters $\delta p = (\Omega_x, \Omega_y, \Omega_z, \Delta t_x, \Delta t_y, \Delta t_z)$. However, it is difficult to locate $p^{'}$ directly. That's why RAPiD searches for strong gradients in orthogonal direction $\vec{n}$. The distance l is written as

$$l = \vec{n}^T(m' - m). \tag{2.4}$$

For each control point P_i, we have

$$l_i = \vec{n}_i W_i \delta p \tag{2.5}$$

where W_i is a 2×6 matrix involving R, P, and t. When enough control points are provided, δp can be calculated as the least-squares solution of equations:

$$\delta p = \underset{\delta p}{\operatorname{argmin}} \sum_i (\vec{n}_i W_i \delta p - l_i)^2 \tag{2.6}$$

Therefor, ΔR and Δt can be obtained respectively as

$$\begin{cases} \Delta R \approx I + \Omega \\ \Delta t = (\Delta t_x, \Delta t_y, \Delta t_z) \end{cases} \tag{2.7}$$

Finally, the pose for the current frame is estimated as $[\Delta R \cdot R | t \Delta t]$. This new pose will be used to predict the pose for the next frame and, in this way, the camera/object pose will be updated for each frame continuously.

2.3.2 RAPiD Improved

RAPiD [**3**] is a very powerful algorithm that can work in real-time, but it has one main drawback, it is not robust. The first problem with this approach is that strong gradients searched near control points could be generated by wrong edges due to cluttered environments, partial occlusions, or adjacent ambiguous edges. RAPiD cannot filter out those false-positive edges because it simply picks the nearest one, and this can cause the problem of getting the wrong 3d to 2D correspondences. Another problem with the RAPiD approach is that it assumes that the relative pose rotation ΔR between two consecutive frames is small. That means that the tracking process can fail if the camera or object is moving very fast.

So, the first improvement proposed to this algorithm was replacing the rotation linearization stage with a minimization approach of the distance between the extracted features and the projections of the 3D model.

$$[R|t] = \underset{[R|t]}{argmin} \sum_i dist(Proj(M_i, [R|t]), m_i') \tag{2.8}$$

Here M_i represents feature points of our 3D model, $Proj(M_i, [R|t])$ denotes the projection of M_i under the pose $[R|t]$. m_i are the computer features. This equation can be solved using Levenberg-Marquardt nonlinear optimization algorithm.

Another improvement that Simon and Berger proposed is tracking stabilization. They introduced robust estimators to reduce the impact of outlier edges during the

optimization. Particularly, M-estimators, which are maximum likelihood-type estimators and special cases of extremum estimators, have been widely used and could be simply written as

$$p(r) = \begin{cases} |r| & |r| < r_{max} \\ r_{max} & |r| \geq r_{max} \end{cases} \tag{2.9}$$

Another two M-estimators were proposed by Tukey and Huber

$$p_{Huber}(r) = \begin{cases} \frac{c^2}{6}[1 - (1 - (\frac{r}{c})^2)^3] & |r| \leq c \\ \frac{c^2}{6} & |r| > c \end{cases} \tag{2.10}$$

$$p_{Tukey}(r) = \begin{cases} \frac{r^2}{2} & |r| \leq c \\ c(|r| - \frac{c}{2}) & |r| > c \end{cases} \tag{2.11}$$

The next improvement was proposed by Vacchetti et al. It was a modification to the Tukey estimator. Now it can consider multiple hypotheses when searching for strong gradients near the projection control points. With this improvement, not only one strong edge gradient is taken, but several of them within predefined distance are considered. The selection of the correct one happens implicitly during the minimization process.

$$p *_{Tukey} (r_1, \ldots, r_n) = \min_i p_{Tukey}(r_i) \tag{2.12}$$

So the equation 2.8 can be modified as follow

$$[R|t] = \underset{[R|t]}{argmin} \sum_{i,j} p *_{Tukey} (dist(Proj(M_i, [R|t]), m_{ij}')). \tag{2.13}$$

Here j is the index of the hypothesis.

2.3.3 Explicit Edges Tracking

Explicit edges tracking is another classic approach, but it uses higher-level edge features. Those can be straight lines, contours, and corners. This approach aims to establish correspondences between the extracted features and similar features from the 3D model. This solution is more robust because we have fewer outliers, but the computational effort of the algorithm will increase. Koller et al. proposed to use Mahalanobis distance between the line segments to establish the correct correspondences. The line segment can be represented as

$$X = (c_x, c_y, \Theta, l) \tag{2.14}$$

Here c_x and c_y represent the center of the line, Θ is its rotation and l is the distance. Let's take for example line segment of our model X_m and the extracted line from the projection X_p. The Mahalanobis distance between them can be defined as

$$d = (X_m - X_p)^T (A_m + A_p)^{-1} (X_m - X_p) \tag{2.15}$$

A_m and A_p are the covariance matrices of X_m and X_p respectively. The pose estimation equation can be solved using the same Levenberg-Marquardt algorithm.

$$[R|t] = \underset{[R|t]}{argmin} \sum_i (X_p^i - X_m^i([R|t]))^T A_d^i (X_p^i - X_m^i([R|t])) \tag{2.16}$$

2.3.4 Texture Based Tracking

Another popular method for 3D model pose estimation is texture-based tracking. This approach requires keyframes, which will be used as a reference image and a 3D model. The keyframes should contain a realistic appearance of our object because they are used to generate features from our 3D model. Practically it is impossible to get the same images for the object due to variability in illumination, scale orientation, and viewpoint. That is why this method is using SURF key points, which are invariant to scale and rotation. Across multiple keyframes, the SURF features are calculated. After this, the 3D location of each feature is calculated by back-projecting each 2D point to the 3D model. During the inference, the features from the input frame are compared with those previously calculated from the 3D model. Then we need to find correspondences between the features and match them. To eliminate possible outliers, the RANSAC is used during the matching phase.

2.4 Neural Networks

2.4.1 PoseCNN + DeppIM

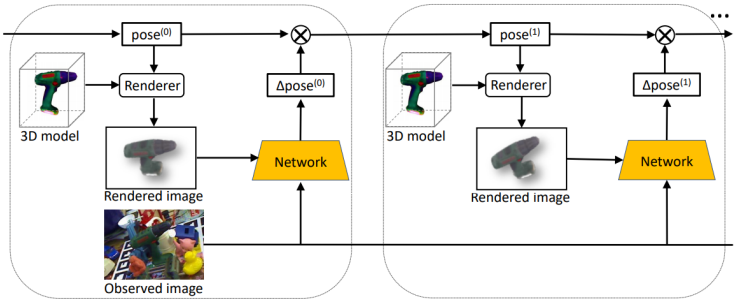


FIGURE 2.8: DeepIM iterative matching

This approach [16] [17] consists of two parts and is the modification of the plain PoseCNN network. The first step of initial pose estimation is done using the PoseCNN neural network. In order to decrease processing time, a Deep Iterative Matching was proposed to replace PoseCNN on the pose refinement step. DeepIM is trained to predict relative rotation and translation of the object, getting the initial position and 3D model as input. Firstly, the 3D model is rendered to the image with respect to the initial pose. Then network uses the input image from the camera and previously rendered image to refine the input pose. The newly estimated pose is then used in the following steps to render a new image and estimate the object's pose.

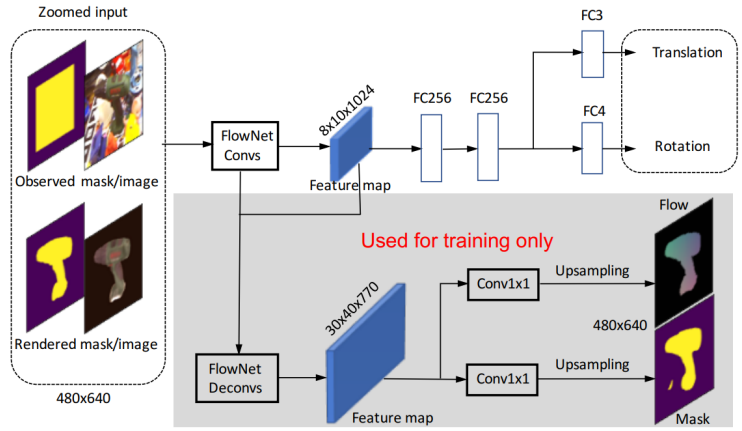


FIGURE 2.9: DeepIM Architecture

DeepIM operates on zoomed-in patches (480x640) from the original image. For relative pose prediction, DeepIM is using the FlowNetSimple backbone. The corresponding mask between the input frame and the rendered image is taken as an input to the convolution layers, which produce a feature map. Then the feature map is forwarded through several fully connected layers to predict the translation and rotation.

2.4.2 EfficientPose

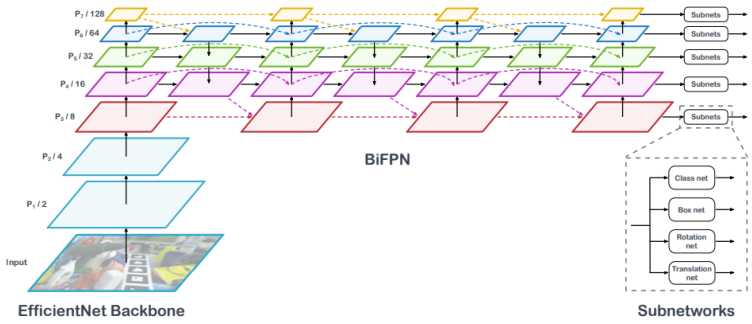


FIGURE 2.10: EfficientPose Architecture

EfficientPose [**15**] solution is a neural network approach based on EfficientNet neural network. Its architecture consists of EfficientNet Backbone, the bidirectional feature pyramid network (BiFPN), and prediction subnetworks. Each subnetwork is divided into four networks which of each is used to predict some parameters of the object. They are the class of the object, its bounding box, rotation, and translation.

2.4.3 HybridPose

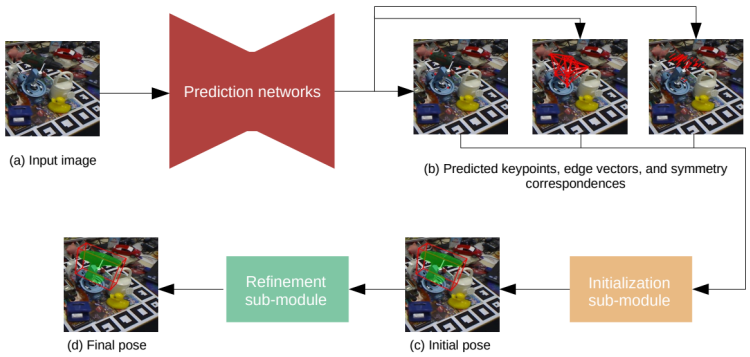


FIGURE 2.11: HybridPose Architecture

The HybridPose [**14**] network consists of two parts, intermediate representation prediction networks, and a pose regression module. The network takes as an input RGB image from the camera, and after processing by prediction networks, it outputs predicted keypoints, edge vectors, and symmetry correspondences. The next step is pose estimation with the help of pose regression module. It also consists of two parts, pose initialization sub-module and pose refinement sub-module. First, the pose initialization sub-module solves system of linear equations with previously predicted key points and gives an estimated pose to the refinement sub-module. For outliers elimination from the initial pose, the refinement step uses the Gauss-Newton optimization algorithm to obtain the final refined pose.

Chapter 3

Implementation

3.1 Approach

For our implementation, we decided to choose the combination of classic approaches. The whole solution consists of three main parts:

1. Model Generation
2. Initial Detection
3. Tracking

We needed to choose some software to work with 3D models for a model generation step because writing our own would take much time. The choice fell on Blender, developed by Blender Foundation. We chose it because it is an open-source project, and it allows you to integrate your Python scripts into its workflow. So, all code for a model generation phase was written in Python.

We decided to choose an edge-based approach with a wireframe overlay to adhere to the already familiar user experience made in commercial solutions in the initial detection step. For the tracking phase, a texture-based method was chosen, but with a camera tracking improvement. Since this solution must be cross-platform and run on devices such as Magic Leap, Hololens, mobile phones, etc., we decided to use Unity Game Engine. It allows not only to create cross-platform applications but is the industry standard for XR development. In Unity C# is the primary language of the development, so the basis of the application and the wrapper computer vision algorithms were written in C#. Also, we used the ARCore library for Android-based devices to get access to video stream from the camera and use the camera tracking function to improve our tracking step. The main logic was written in C++ to preserve the application's speed and make the development of computer vision algorithms easier and simpler.

3.2 Model Generation

To generate a model for our solution, we need to choose some 3D object. The object is imported into Blender, where the python script renders the model from different angles, saving the camera position, rotation, and intrinsic parameters on each frame. Also, from a predefined angle of view, the object wireframe is generated using Canny Edge Detector, which then will be used in the phase of initial detection.

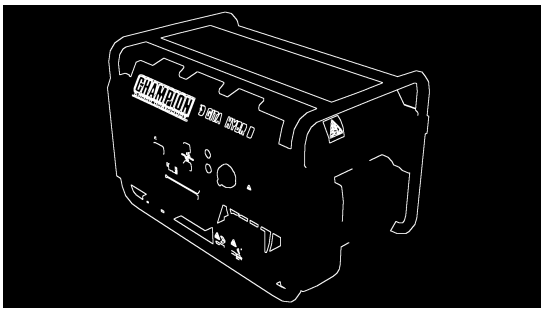
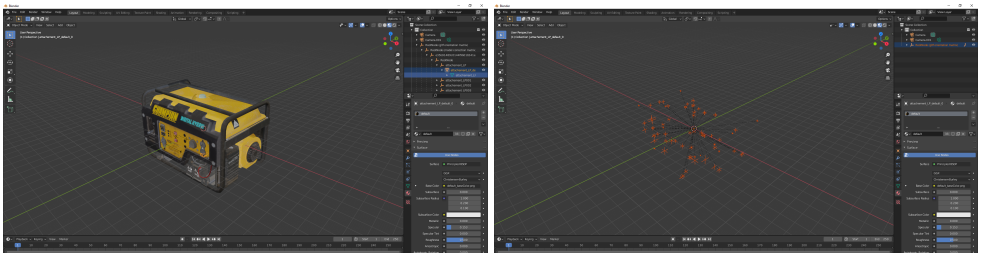


FIGURE 3.1: 3D model wireframe example

For the tracking phase we generate keypoint information from our 3D model. For each pair of images, we detect ORB features and match them. Outliers are removed using RANSAC algorithm. After the matches filtration, for each image, we project a ray from the camera origin (position from which the corresponding image was rendered) through the image plane (position of keypoint) to find the intersection of lines. Those intersection points will be the 3D positions of our keypoints from the model. The keypoints with their 3D positions and descriptors are saved into file, which then will be used in tracking phase.



(a) (b)

FIGURE 3.2: (a) Sample object on Blender scene (b) Extracted keypoints 3D positions

3.3 Initial Detection

For the initial object pose estimation, we used a similar approach, which is described in the paper "Occlusion, Clutter, and Illumination Invariant Object Recognition" by Carsten Steger **[3]**. Firstly, we need to calculate direction vectors for both input image and previously saved template image from the model generation step (with predefined rotation and translation). Direction vectors are calculated using Sobel operator over X and Y axis. For example, template image consists of set of points $p_i = (x_i, y_i)^T$, the direction vectors for this template will be $d_i = (t_i, u_i)^T$, where $i = 1, \ldots, n$. For the input image direction will have representation of $e_{x,y} = (v_{x,y}, w_{x,y})^T$ for each image point (x, y). Then we need to calculate similarity measure between template and input image direction vectors. The similarity measure which is robust to occlusion, clutter and illumination changes is defined as follow

$$s = \left| \frac{1}{n} \sum_{i=1}^{n} \frac{\langle d'_i, e_{q+p'} \rangle}{\|d'_i\| \cdot \left\| e_{q+p'} \right\|} \right| = \left| \frac{1}{n} \sum_{i=1}^{n} \frac{t'_i v_{x+x'_i, y+y'_i} + u'_i w_{x+x'_i, y+y'_i}}{\sqrt{t'^2_i + u'^2_i} \cdot \sqrt{v^2_{x+x'_i, y+y'i} + w^2_{x+x'_i, y+y'_i}}} \right| \tag{3.1}$$

This similarity measure is also invariant to arbitrary illumination changes because all vectors are scaled to a length of 1. What makes this measure robust against occlusion and clutter is the fact that if a feature is missing, either in the model or in the image, the noise will lead to random direction vectors, which, on average, will contribute nothing to the sum. The final step is comparing the calculated measure with a threshold; if the value exceeds it, that means that we successfully detected our object with already known translation and rotation.

3.4 Tracking

The last step of our approach is tracking. Here we decided to combine two methods, texture-based tracking, and camera tracking. As we are using devices with a gyroscope and accelerometer, we integrated a camera tracking function from the ARCore library. After the initial pose detection, we use the object's 3D position to set up the camera tracking. Object pose is updated by subtracting the difference between the initial rotation and translation and the predicted from the camera tracker. To refine our position, we added a texture-based tracking method. We decided not to use it for each frame because it is computationally expensive. So, after N frames from the initial detection, we applied feature matching between previously calculated features (from the model generation step) and generated from the input frame. After the matching process and outliers filtration, we compute the homography matrix, which helps us refine our object's position. The final step is to estimate the updated camera pose by solving the PnP problem with 3D positions of the features from the generated model and their 2D projections on the image plane.

Chapter 4

Evaluation

4.1 Metrics

The ADD metric was used to calculate metrics of Neural Networks approaches for the object pose estimation task. ADD stands for an Average Distance of Model Points. It is an average distance between all corresponding points in our model and the prediction. Having the ground truth rotation R and translation T and the estimated rotation $\hat{R}$ and translation $\hat{T}$, we can compute the average distance between each point x of our ground truth Model M and corresponding to them points from our prediction:

$$ADD = \underset{x \in M}{avg} \left\| (Rx + T) - (\hat{R}x + \hat{T}) \right\| \tag{4.1}$$

The above equation is a good score function for not occluded objects. However, what to do if we have some occlusions? Then we need to use a little bit modified version of our score function. We need to calculate the average distance between our prediction and its nearest neighbor in 3D space, which may not necessarily be the corresponding vertex.

$$ADD = \underset{x_1 \in M}{avg} \underset{x_2 \in M}{min} \left\| (Rx_1 + T) - (\hat{R}x_2 + \hat{T}) \right\| \tag{4.2}$$

An estimated pose is considered correct if the average ADD is less than 10% of the object diameter.

To evaluate the classic computer vision approach that we implemented, we decided to take a different approach in metrics evaluation. We compare two metrics, the first one the distance between the ground truth position T of the object and the predicted one $\hat{T}$, and the second one is the difference between the rotation angle of the object R and the prediction $\hat{R}$. Then we take the mean value of all the observations N (one observation per frame).

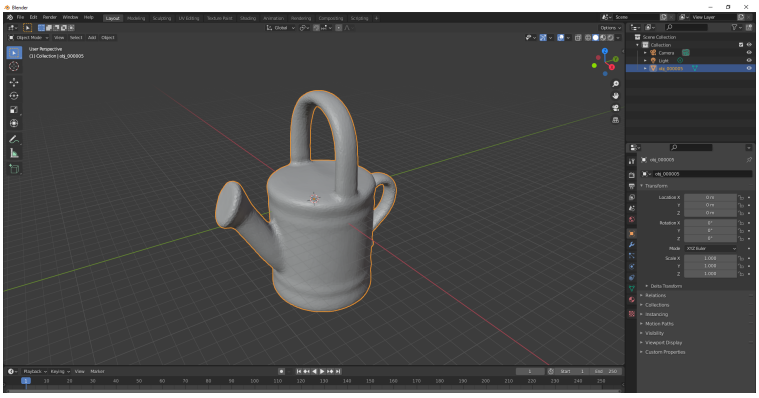
$$\overline{\Delta T} = \frac{1}{N} \sum_{n=0}^{N} T_i - \hat{T}_i \tag{4.3}$$

$$\overline{\Delta R} = \frac{1}{N} \sum_{n=0}^{N} R_i - \hat{R}_i \tag{4.4}$$

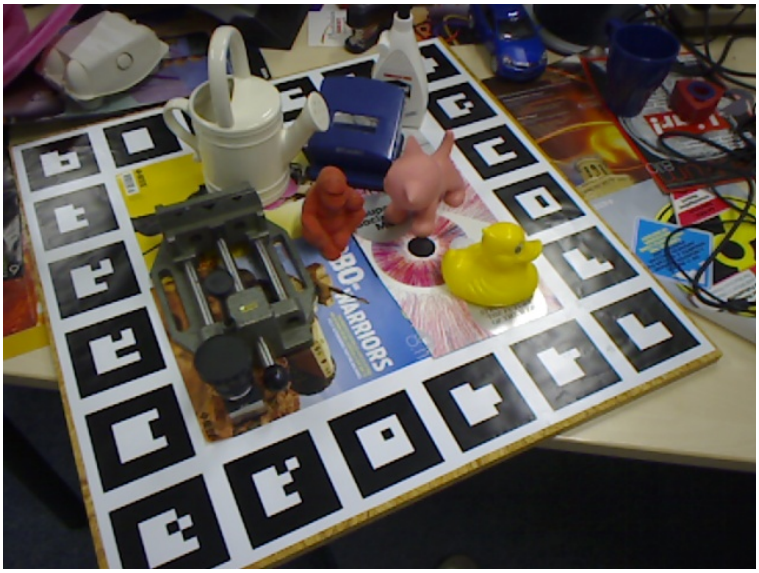
ΔT is measured using the measuring units of the 3D model (*e.g.*, Millimeters, Centimeters, etc.). ΔR is calculated in degrees, where R and $\hat{R}$ are rotations in three axes X, Y, and Z.

4.2 Datasets

For the neural network testing, the LineMOD dataset was used. The dataset includes 3D models and training and test RGBD images with ground truth rotation and translation values. Also, intrinsic camera parameters are included in the dataset.



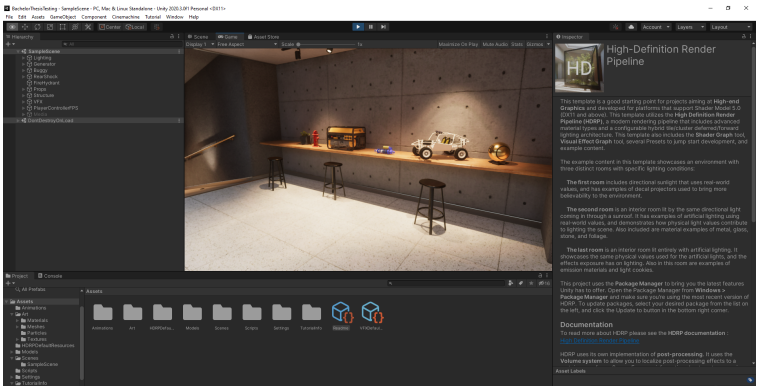
(a)



(b)

FIGURE 4.1: (a) 3D Model example from LineMOD Dataset (b) Sample image from the LineMOD Dataset

For the evaluation of our implementation, the current dataset cannot be used. It is because our approach uses the edge-based detection of the initial pose. That means that we cannot identify the model on the whole image, but the user needs to overlay the projected wireframe with the actual object contours. So, for the testing, we created the virtual environment using the Unity game engine.



(a)

FIGURE 4.2: Unity interface with the created sample scene

Unity provides the ability to select and configure different types of render pipelines, so we created a sample scene using the High Defenition Render Pipeline, which allows you to get a realistic-looking image. Several test objects were placed on the scene, and simple camera movement control was added to simulate the movements of mobile phone device.

(a)

(b)

FIGURE 4.3: (a) Testing scene example (b) The process of the initial detection

With the help of the Unity Game Engine, we can not only get a realistic picture that helps us simulate the real-world best, but also get the exact positions and rotations of both the 3D model on the scene and the camera itself for each frame. This information helps us to measure the accuracy of our implementation precisely.

4.3 Results

For the Neural Network part the results are shown in the table below.

Neural Networks Results	
Approach	Mean ADD
EfficientPose	97.35
HybridPose	91.3
PoseCNN + DeepIM	88.6

Chapter 5

Summary

Therefore, concluding this work, I can confidently say that both classic approaches and neural networks work very well to solve the task of pose estimation. However, it all depends on what we will use this solution for. The main drawback of neural networks is that they require a lot of GPU memory to run in real-time, and they cannot be easily generalized. They can work perfectly in robotics, where there is an ability to have more computing power. Regarding our task of using pose estimation for XR, classic approaches suit better here. They require less computational power, are easy to implement, and while running in real-time, even on mobile phones, they have less jitter and provide a more stable image.

Bibliography

[1] Michael Lowney, Abhilash Sunder Raj. *Model Based Tracking for Augmented Reality on Mobile Devices*. Stanford, CA 94305, 2016.

[2] Carsten Steger. *Occlusion, Clutter, and Illumination Invariant Object Recognition*. MVTec Software GmbH, Neherst. 1, 81675 Munchen, Germany, 2002.

[3] Pengfei Han, Gang Zhao. *A review of edge-based 3D tracking of rigid objects*. Beihang University, Beijing 100191, China, 2019.

[4] Harald Wuest, Didier Stricker. *Tracking of industrial objects by using CAD models*. Fraunhofer IGD, Darmstadt, Germany, 2007.

[5] Hamdi Ben Abdallah, Igor Jovancevic, Jean-Jose Orteu, Ludovic Brethes. *Automatic Inspection of Aeronautical Mechanical Assemblies by Matching the 3D CAD Model and Real 2D Images*. Universite de Toulouse, Albi, France, 2019.

[6] Changhyun Choi, Henrik I. Christensen. *Real-time 3D Model-based Tracking Using Edge and Keypoint Features for Robotic Manipulation*. College of Computing Georgia Institute of Technology, Atlanta, GA 30332, USA, 2010.

[7] Christian Wiedemann, Markus Ulrich, Carsten Steger. *Recognition and Tracking of 3D Objects*. MVTec Software GmbH, Neherst. 1, 81675 Munchen, Germany, 2008.

[8] Vincent Lepetit, Pascal Fua. *Monocular Model-Based 3D Tracking of Rigid Objects: A Survey*. Computer Vision Laboratory, CH-1015 Lausanne, Switzerland, 2005.

[9] Oliver Moolan-Feroze, Andrew Calway. *Predicting Out-of-View Feature Points for Model-Based Camera Pose Estimation*. 2018.

[10] Kenji Hata and Silvio Savarese. *CS231A Course Notes 1: Camera Models*. 2021.

[11] Luis Enrique Ortiz, Luiz Marcos G. Gonçalves, Elizabeth Viviana Cabrera. *A Generic Approach for Error Estimation of Depth Data from (Stereo and RGB-D) 3D Sensors*. 2017.

[12] Kyle Simek. *Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix*. 2013.

[13] Jack Liu. *The difference between AR, VR, MR, XR and how to tell them apart*. 2018.

[14] Chen Song, Jiaru Song, Qixing Huang. *HybridPose: 6D Object Pose Estimation under Hybrid Representations*. The University of Texas at Austin, 2020.

[15] Yannick Bukschat, Marcus Vetter. *EfficientPose: An efficient, accurate and scalable end-to-end 6D multi object pose estimation approach*. Steinbeis Transferzentrum an der Hochschule Mannheim, 2020.

[16] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, Dieter Fox. *DeepIM: Deep Iterative Matching for 6D Pose Estimation*. 2018.

[17] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, Dieter Fox. *PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes*. NVIDIA Research, University of Washington, Carnegie Mellon University, 2017.

[18] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, Nassir Navab. *Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes*. CAMP, Technische Universit at Munchen (TUM), Germany Industrial Perception, Palo Alto, CA, USA CV-Lab, Ecole Polytechnique Federale de Lausanne (EPFL), Switzerland, 2012.

[19] ViSP *Markerless 3D model-based tracker module overview* https://hackernoon.com/the-difference-between-ar-vr-mr-xr-and-how-to-tell-them-apart-45d76e7fd50 2021

[20] Jack Liu *The difference between AR, VR, MR, XR and how to tell them apart* https://library.vuforia.com/features/objects/model-targets.html 2018

[21] Kaitlyn Irvine *XR: VR, AR, MR — What's the Difference?* https://medium.com/viget-collection/xr-vr-ar-mr-whats-the-difference-ac7c3fcd590d 2017

[22] Cory Mitchell *Virtual Reality* https://www.investopedia.com/terms/v/virtual-reality.asp 2020

[23] Vuforia *How to Create a Model Target* https://library.vuforia.com/articles/Solution/how-to-create-model-target.html 2021

[24] Vuforia *Model Target Generator User Guide* https://library.vuforia.com/articles/Solution/model-target-generator-user-guide.html 2021

[25] Vuforia *Model Target Guide View* https://library.vuforia.com/articles/Solution/model-target-guide-view.html 2021

[26] Wikitude *3D Model-Based Object Targets* https://www.wikitude.com/3d-model-based-object-tracking/ 2021

[27] Wikitude *Target Management* https://www.wikitude.com/external/doc/documentation/studio/targetmanagement.html#object-targets 2021

[28] Wikitude *Best practice for Object Targets* https://www.wikitude.com/external/doc/documentation/latest/android/objecttargetguide.html#image-based-conversion-best-practice 2021

[29] Wikitude *3D Model Object Tracking* https://www.wikitude.com/blog-sdk-9-2-now-available-3d-model-object-tracking/ 2021

[30] ViSP *Markerless 3D model-based tracker module overview* https://visp.inria.fr/mbt/ 2021