Real-Time 6DoF Tracking of Rigid 3D Objects Using a Monocular RGB Camera

Graduação em Ciência da Computação
Centro de Informática
Universidade Federal de Pernambuco

Bachelor's Dissertation presented to the Center of Informatics of the Federal University of Pernambuco in partial fulfillment of the requirements for the degree of
Bachelor of Computer Science
by

Lucas Valença Rocha Martins de Albuquerque
(lvrma@cin.ufpe.br)

Advisor: Veronica Teichrieb *(vt@cin.ufpe.br)*
Co-Advisor: Francisco Paulo Magalhães Simões *(fpms@cin.ufpe.br)*

Subject Areas: Computer Vision, Pattern Recognition, 6DoF Tracking, Augmented Reality

Recife
2020

FICHA

BANCA

Dedicatória

Eu gostaria de, primeiro de tudo, dedicar este trabalho à minha família. Tem gente que tem uma mãe dedicada, mas eu tenho a sorte de ter três, todas 24h por dia todos os dias dispostas a fazer de tudo pra que qualquer loucura que eu sonhasse parecesse totalmente tangível. Minha mãe Sonia, avó Vilma e tia Nana, esse trabalho, assim como qualquer outro que eu fizer, é fruto do esforço de vocês. Também quero agradecer ao meu tio e figura paterna, Zaldo, o primeiro PhD. que eu conheci na vida, e à minha irmã Lila. Esses dois são fontes de admiração e inspiração que me fazem sempre querer correr atrás de algo melhor sabendo que, se eu tenho a quem puxar, vai dar certo.

Quando eu entrei nessa universidade, com um supletivo e um certificado de Java debaixo do braço, não esperava que 5 anos depois fosse ter passado por tanta coisa. Na primeira aula com o Prof. Silvio, eu achava que matriz era uma igreja bem grande. Hoje, depois de ter pago absolutamente todas as cadeiras que ele disponibilizou, só tenho a agradecer por ser um professor tão bom e ter me introduzido a este novo mundo. Logo em seguida, o Voxar Labs e seus líderes Chico, Lucas, Jonga, VT e Maria me botaram debaixo das asas, mesmo com toda a minha doideira. Foi a extrema paciência e tolerância deles que me guiou e formou como pesquisador. Sou aluno do curso de ciência, mas sem o Voxar eu jamais saberia o que realmente significa ser um cientista.

Não poderia deixar de mencionar meus três terapeutas: Pedro, o oficial, Manu, a namorada, e Alfredo, o cachorro. Vocês me tiraram de momentos de grande depressão, me curaram e me ensinaram uma nova forma de viver melhor. Nunca vou ter como expressar a minha gratidão em palavras. Certamente eu não estaria terminando esse curso sem cada um de vocês. Quero deixar um agradecimento especial a Alfredão por nunca ter hesitado em me acompanhar em cada uma das inúmeras noites em claro que geraram este trabalho aqui.

Por fim, quero deixar uma nota aos amigos. Passei por muitos perrengues brabos ao longo dos últimos anos e esse foi o pessoal que largou tudo e imediatamente correu pra me ajudar, sem pensar duas vezes, não importa o quão insignificante fosse a necessidade. Faço questão de eternizar aqui seus nomes de guerra: Lucas Chuchu (a.k.a. Cunhado), Edjan Popojan, Robson Diamante Negro, Felipe Cavalão, MM, Pepeu Santos, Rodrigo Boldrigo, Lucana Ananias, Coach André, Pedro Barbudo, Arlindo Arsimples, André Lobo Mau, Richarles Barioni, Xilef Rotieh, Thiago Chefinho, Inusi Sergio, Danilove e o seu lobo, Hamilton. Nas palavras do grande Mano Brown, vocês são imortal nos meus versos.

Acknowledgements

This research was partially funded by Conselho Nacional de Desenvolvimento Cientíifico e Tecnológico (CNPq) (process 425401/2018-9). This work was developed at Voxar Labs and was both supervised and reviewed by professors Francisco Simões, João Paulo Lima, Lucas Figueiredo, and Veronica Teichrieb. The augmented reality add-on was implemented by Luca Ananias, who also helped with debugging the multi-object variation. Thiago Chaves helped with running the public implementations of GOS and RBOT on all datasets, generating part of the final results used for comparisons in this work. He also helped with reviewing the manuscript. Arlindo Gomes helped with dealing with difficult 3D models and designed the Glass Bottle model of the 3DPO dataset. All aforementioned names are to be included as co-authors in eventual publications of this work and are listed here for transparency.

Additionaly, over the course of this research, Voxar Labs members Lucas Maggi, Rafael Roberto, Gutenberg Barros, Ricardo Barioni, Kelvin Cunha, Heitor Felix, and Thiago Souza took part in helpful discussions and are thanked.

The manuscript of this dissertation was further reviewed by professor Tsang Ing Ren and Ricardo Barioni, both of which have made very pertinent suggestions and are also thanked.

I would like to thank Po-Chen Wu (OPT's author) as well as Bogdan Bugaev for their very helpful tips on the usage and implementation of the OPT dataset and the AUC metric, respectively. Lastly, I also want to thank RBOT's author Henning Tjaden for his clarifications on running the public implementation of RBOT on the OPT dataset.

Abstract

Estimating the translation and rotation of a 3D object in the real world is one of the quintessential problems in computer vision. If one is to teach a computer how to understand what it sees in an image, as is the goal of computer vision, more often than not it is also necessary to know where and how each thing that is seen is positioned, as simply knowing that the thing is “there somewhere" is usually not sufficient. As humans, we tend to take this task for granted, as we are naturally equipped with a very high-resolution stereo camera (our eyes) and a huge neural network trained for tens of thousands of hours (our brain). Even so, we are not capable of exactly telling the position and rotation of an object by looking at it, which makes the recent state-of-the-art advancements in 6DoF tracking all the more impressive.

This work introduces a novel real-time edge-based 6DoF tracking approach for 3D rigid objects requiring just a monocular RGB camera and a CAD model with material information. The technique is aimed at low-texture or textureless, pigmented objects. It works even when under strong illumination, motion, and occlusion challenges. It is shown how preprocessing the model’s texture can improve tracking and how applying region-based ideas like localized segmentation improve the edge-based pipeline. This way, the technique is able to find model edges even under fast motion and in front of high-gradient backgrounds. The implementation runs on desktop and mobile. It only requires one CPU thread per object tracked simultaneously and requires no GPU. It showcases a memory footprint that is orders of magnitude smaller when compared to the state of the art. To show how the technique contributes to the state of the art, comparisons were made using two publicly available benchmarks.

Keywords: 6DoF Tracking, Edge-Based, Monocular, Real-Time, Model-Based, Mobile, Multi-Object, Augmented Reality, Textureless, Segmentation, RGB, HSV, Single-Thread.

Resumo

Estimar a translação e rotação de um objeto 3D no mundo real é um dos problemas fundamentais da visão computacional. Se formos ensinar um computador a entender o que ele vê em uma imagem, como é o objetivo da visão computacional, na maioria das vezes também é necessário saber onde e como cada coisa está posicionada, já que simplesmente saber que a coisa está "ali em algum lugar" geralmente não é suficiente. Como humanos, não nos preocupamos muito com isso, pois estamos naturalmente equipados com uma câmera estereoscópica de altíssima resolução (nossos olhos) e uma rede neural gigantesca treinada por dezenas de milhares de horas (nosso cérebro). Ainda assim, nós não conseguimos estimar a pose 6DoF de um objeto apenas olhando pra ele, o que faz os avanços recentes no estado da arte de rastreamento 6DoF ainda mais impressionantes.

Este trabalho introduz uma nova técnica de rastreamento 6DoF em tempo real baseada em arestas para objetos 3D rígidos que requer somente uma câmera RGB monocular e um modelo CAD com informação de material. A técnica é focada em objetos pigmentados e com pouca ou nenhuma texturização. Funciona mesmo sob os desafios de iluminação, movimento, e oclusões fortes. É mostrado como pre-processar a textura do modelo pode melhorar o rastreamento e como aplicar ideias baseadas em região, como segmentação local, pode melhorar a *pipeline* baseada em arestas. Assim, a técnica é capaz de encontrar arestas do modelo mesmo sob movimentos rápidos e diante de fundos com muitos gradientes. A implementação roda em aparelhos *desktop* e móveis. Só é necessário uma *thread* da CPU por objeto rastreado simultaneamente e GPU não é necessária. A técnica possui um consumo de memória menor em ordens de magnitude quando comparado com o estado da arte. Para mostrar como a técnica contribui para o estado da arte, comparações foram feitas utilizando dois *benchmarks* publicamente disponíveis.

Palavras-chave: Rastreamento 6DoF, Baseado em Arestas, Monocular, Tempo Real, Baseado em Modelo, Móvel, Multi-Objeto, Realidade Aumentada, Sem Textura, Segmentação, RGB, HSV, Única Thread.

List of Figures

List of Tables

List of Acronyms

3DPO	3D Printed Object
6DOF	6 Degrees of Freedom
AR	Augmented Reality
AUC	Area Under Curve
CAD	Computer-Aided Design
CPU	Central Processing Unit
CV	Computer Vision
DL	Deep Learning
FPS	Frames per Second
GL	Graphics Library
GOS	Global Optimal Searching
GPU	Graphics Processing Unit
LM	Levenberg–Marquardt
ML	Machine Learning
MR	Mixed Reality
NDK	Native Development Kit
OPT	Object Pose Tracking
ORB	Oriented FAST and Rotated BRIEF
PNP	Perspective-n-Point
RAM	Random-Acess Memory
RBOT	Region-Based Object Tracking
SLAM	Simulatenous Localization and Mapping
VGA	Video Graphics Array
VR	Virtual Reality
XR	Cross Reality

List of Symbols

A	Function that returns the set of all pixels inside a closed contour.
C	8-bit RGB diffuse color vector of a face's material.
E	Extrinsic camera parameters as a 4×4 matrix $\in \mathbb{SE}(3)$.
H	The smoothed unit step function.
K	Intrinsic camera parameters as a 3×3 matrix $\in \mathbb{R}^{3\times3}$.
O	Occlusion detection set.
R	3D rotation expressed as a 3×3 matrix $\in \mathbb{SO}(3)$.
S	Parametric pixel linear search function.
W	Subset of pixels in $A(\zeta)$ not within the search range.
Φ	Binary segmentation mask.
Θ	Function to obtain an orthogonal vector from another 2D vector.
$\tilde{\Theta}$	Normalized output of Θ.
Υ	Set of 2D normal vectors for every element of ζ.
β	Function that projects a point from $\mathbb{P}^3$ to $\mathbb{R}^2$.
χ	Set of 2D-3D match candidate tuples.
$\varnothing$	The empty set.
η	Saturation tolerance range function.
ι	Region containing all pixels visited during circular search.
λ_A	Resolution scale factor.
λ_S	Shortest frame side scale factor.
μ	Unprojected 3D equivalent of ζ.
ν	2D dimensions of the base VGA resolution (640×480).
ϕ	Parametric pixel search line kernel.
π	Ratio of a circle's circumference to its diameter.
σ	Uncertainty coefficient.
ξ	Preprocessed set of model edges.
ζ	Set of pixels describing the model's 2D silhouette.
c	Principal point of the camera.
d	Variable determining the linear search direction.
d_θ	Shortest (cyclic) distance function between hue values.
f	Focal length of the camera.
k_H	Sharpness coefficient of the smoothed unit step function.
k_m	Largest contour percentage threshold for mask simplification.
k_A	Maximum percentage of frame pixels a search circle can occupy.
k_η	Vertical shift of the saturation tolerance range function.

k_θ	Angular coefficient of radius growth.
k_l	Elasticity coefficient of the saturation tolerance range function.
k_{p1}	Accumulated pre-processed probability limit for a face.
k_{p2}	Limit amount of different colors preprocessed for a face.
k_s	Desaturation threshold.
r	Radius of the segmentation search circles.
r_0	Minimum radius value in pixels.
r_1	Maximum allowed radius value in pixels.
t	3D translation expressed as a 3×1 vector $\in \mathbb{R}^3$.

Contents

1

INTRODUCTION

Tracking the 6DoF pose of a 3D object through monocular videos in real-time has been a long-time computer vision challenge. Multiple tasks in thriving technological fields (e.g., extended reality, robotics, autonomous vehicles, medical imaging) require information regarding the objects populating the environment. Real-time tracking requires high precision with temporal consistency at every snapshot of the scene. In addition, dynamic conditions of the real world may impose numerous difficulties, such as unexpected occlusions, foreground or background clutter, varying illumination, and motion blur. Hardware challenges are also present (e.g., lack of processing power or sufficiently reliable scene input from available sensors).

It is also important to briefly mention detection techniques, which are often seen as complementary to tracking works. The main difference between them lies in the temporal consistency and computational cost. Usually, the detection step (which can be automated or even manually aligned) provides an initial estimation for trackers to use. Unlike detectors, trackers achieve higher efficiency by not needing to search the entire frame every time, as the object is expected to be in the vicinity of where it was in the last frame. Additionally, trackers try to make the current frame's estimation coherent with the previous frames, generating smoother results.

Objects being tracked can vary in appearance, being considered textured if there is a significant amount of visual information such as internal gradients and color variation. The lack of such information in textureless objects configures a relevant category considering how widely common these objects are (e.g., 3D prints, industrial parts), and how suitable their textureless surfaces are for applications such as augmented reality.

This dissertation presents a novel 6DoF object tracking technique (see Fig. 1) based on edges and photometric properties. The technique is mainly aimed at rigid, textureless or low-texture objects with pigmentation (see Section 4.4). It uses previously known material information from the input 3D model to generate photometrically-robust local color data. This data undergoes statistical simplification in order to generalize it, enabling matching to real-world objects. During tracking, the color matching process then generates a binary segmentation mask in real time. The approach is shown to be capable of matching object edges even under motion blur or in front of high-gradient backgrounds, helping to overcome what is arguably the largest pitfall of edge-based tracking. A novel, more computationally efficient way to match object's 2D

Figure 1: The proposed work in different scenes. Left to right: flashing lights, occlusion, clipping, motion blur, multi-object, and augmented reality.

and 3D silhouette points and a new search range concept are also introduced.

In terms of hardware, the only required input sensor is a monocular RGB camera, such as a usual mid-range commodity webcam, and a single CPU core per object. The proposed technique is also, to the best of the author's knowledge, the first monocular real-time 6DoF object tracker among state-of-the-art contenders to have been shown to work in mobile devices. For this reason, it can be argued that the proposed work is more suitable for lightweight application scenarios than currently existing methods. Additionally, most fields that make use of object tracking commonly utilize it as part of a larger pipeline (running simultaneously with other algorithms). Thus, sparing resources by just using one core of the (usually multi-core) modern CPU can be considered a desirable feat.

1.1 Objectives and Contributions

This dissertation's main objective and contribution is to introduce a novel multi-platform edge-based approach aimed at textureless, pigmented objects which provides comparable precision to the state of the art with less hardware requirements, lower runtime, and a memory footprint orders of magnitude smaller. The technique excels in handling illumination changes and fast motion. It showcases a new way for edge-based tracking to isolate object edges under blur and high-gradient backgrounds. The technique requires less initialization precision and can recovery from partial losses without needing to be reset. Quantitative evaluations to validate the claims are performed using 2 different publicly available benchmarking datasets. Comparisons to the current state of the art are also performed. Secondary contributions are listed as follows:

- A brief study on real-time monocular 6DoF tracking and how CAD object materials can be used to improve it.
- A more efficient way to encode and unproject 3D models on 2D frames.
- A novel dynamic approach to determining the tracking search range.
- An additional segmentation step to the RAPID-like edge-based pipeline, inspired by the region-based state of the art. This step aids with handling motion blur and high-gradient backgrounds, which are challenging aspects of edge-based tracking.
- A real-time approach to HSV-based segmentation to lessen the *reality gap*.

- A new, more challenging dataset focused on textureless objects using high-quality 3D prints.
- An AR rendering tool for visualizing the results of 6DoF detectors and trackers.

Finally, the proposed work has been accepted for publication as a full paper on the 16^{th} International Conference on Computer Vision Theory and Applications (VISAPP 2021). It is the author's hope that the contents of this dissertation can gain more international visibility through this publication and therefore have a greater impact on future works in the area.

1.2 Dissertation Outline

This work is organized as follows: Chapter 2 provides a brief overview of projective geometry. It also provides some insight on core computer vision and graphics concepts necessary for the comprehension of the dissertation; Chapter 3 provides an overview of the 6DoF monocular RGB tracking field as well as the current state of the art; Chapter 4 describes the proposed approach; Chapter 5 details the implementation and the additional tools the accompany this work; Chapter 6 provides quantitative evaluations of the proposed method; and Chapter 7 contains the conclusion and future work discussion. Supplemental information can be found in the Appendices and will be introduced throughout the work accordingly.

2

BACKGROUND THEORY

The work described in this Bachelor's dissertation covers a very simple approach to a very nontrivial problem. In fact, publicly-available state-of-the-art 6DoF trackers mentioned for comparison have also been published as PhD. theses, each of which can be highly informational (Tjaden, 2019; Prisacariu, 2012). The works described in this dissertation make use of knowledge from the areas of linear algebra, projective geometry, computer vision, digital image processing, and computer graphics. The in-depth explanation of these topics is beyond the scope of this dissertation, which will provide a brief overview and links to in-depth references instead.

2.1 Projective Geometry

This section will review some core concepts of projective geometry behind real-time pose estimation using a 3D model. General linear algebra knowledge is assumed. For a more in-depth look at the math, please refer to the Multiple View Geometry in Computer Vision book (Hartley & Zisserman, 2003). For detailed information specifically targeted at the 6DoF tracking problem, please refer to the Survey on Monocular Model-Based Tracking of Rigid Objects (Lepetit *et al.*, 2005).

2.1.1 The Pinhole Camera Model

This model, very common in computer vision, describes an ideal *pinhole camera* (see Fig. 2). It is used to express the mathematical relationship between a point in $\mathbb{P}^3$ and its 2D projection on $\mathbb{R}^2$, over the so-called *image plane*. This mapping is expressed by a 3×4 *camera projection matrix*, which, for simplicity, will be used in its homogeneous 4×4 form in this work.

Linear projection parameters are expressed in this work as the 3×3 intrinsic parameter matrix

$$K = \begin{bmatrix} f_x & 0 & cx \\ 0 & f_y & cy \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times3}, \tag{2.1}$$

where (f_x, f_y) represents the focal length in terms of pixels, and (c_x, c_y) represents the principal

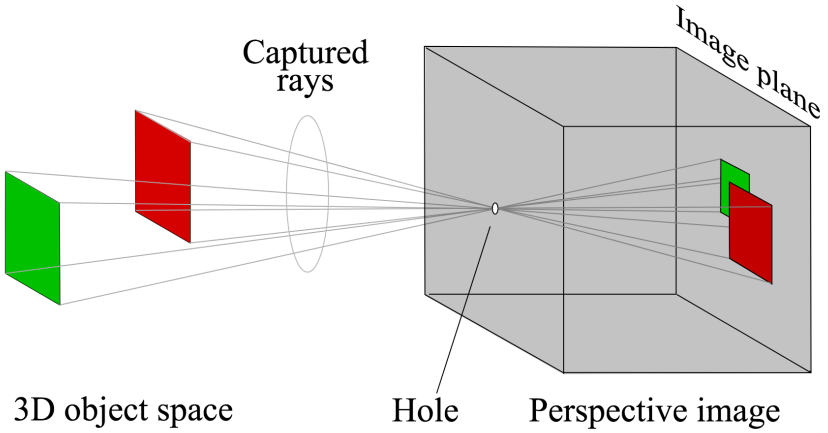


Figure 2: An illustration of the pinhole camera model.*

*Image publicly available by the CC0 (Creative Commons - No Rights Reserved) license, courtesy of Wikimedia Commons user *Trassiorf*.

point, as per the pinhole camera model (Hartley & Zisserman, 2003).

This way, the image plane projection $p \in \mathbb{R}^2$ of an object point $P \in \mathbb{R}^3$ can be described by the standard camera model as

$$p = \beta(K \cdot \beta(E \cdot \tilde{P})), \tag{2.2}$$

where matrix E will be introduced in the following subsection and the tilde notation over $\tilde{P}$ represents the point P in homogeneous coordinates so that if $P = [x,y,z]^T \in \mathbb{R}^3$, then $\tilde{P} = [x,y,z,1]^T \in \mathbb{P}^3$ (3D projective space). Additionally, β represents the conversion function from projective to real domains $\mathbb{P}^n \to \mathbb{R}^n$ for every $m \times 1$ vector $X \in \mathbb{P}^n$, where $m = n+1$, as follows

$$\beta(X) = \left[\frac{X_{(0,1)}}{X_{(m,1)}}, \cdots, \frac{X_{(n,1)}}{X_{(m,1)}} \right]^T \in \mathbb{R}^n \tag{2.3}$$

.

There are multiple other very interesting properties beyond what's being explained here. Those can be found in more detailed explanations on projective spaces and homogeneous coordinates (Hartley & Zisserman, 2003).

2.1.2 Rotation and Translation Representations

Rotations can be defined as 3×3 matrices members of the 3D rotation group often denoted as $\mathbb{SO}(3)$ so that

$$\mathbb{SO}(3) = \left\{ R \mid R \in \mathbb{R}^{3\times 3},\ R^T R = R R^T = I,\ det(R) = 1 \right\}, \tag{2.4}$$

where I represents the identity matrix.

For translation, these can usually be expressed as simply a 3×1 vector in $\mathbb{R}^3$, which

represents 3DoF motion without rotation.

For optimization purposes, the rotation matrix R might sometimes be used as r_R, corresponding to the twist-coordinate 3×1 vector v_{rot} obtained through Rodrigues' rotation formula as follows

$$v_{rot} = v \cdot cos(\theta) + (k \times v) \cdot sin(\theta) + k \cdot (k \cdot v) \cdot (1 - cos(\theta)), \tag{2.5}$$

where θ represents the amount of angles for which vector $v \in \mathbb{R}^3$ will be rotated around the axis represented by vector $k \in \mathbb{R}^3$

For more information regarding Rodrigues' formula and 3D rotations, please refer to the exponential mapping section of the Monocular Tracking Survey (Lepetit *et al.*, 2005).

2.1.3 The Homogeneous Camera Projection Matrix

The extrinsic parameter matrix which describes the 6DoF rigid body transform from 3D model space to camera coordinate space is represented by the 4×4 homogeneous camera projection matrix

$$E = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in \mathbb{SE}(3), \tag{2.6}$$

with $R \in \mathbb{SO}(3)$ representing the 3×3 rotation matrix, $t \in \mathbb{R}^3$ representing the 3×1 translation vector, and $\mathbb{SE}(3)$ being the Lie group defined as

$$\mathbb{SE}(3) = \left\{ E \mid E = \begin{bmatrix} R & t \\ 0_{1\times 3} & 1 \end{bmatrix}, R \in \mathbb{SO}(3), t \in \mathbb{R}^3 \right\}, \tag{2.7}$$

For more details, a very intuitive derivation and explanation of these calculations can be found in the PhD. thesis of Henning Tjaden (Tjaden, 2019).

If we use the shortened version of R, defined previously as r_R, the 4×4 matrix E in (2.6) can also be referred as a 6×1 vector as follows

$$e_E = \begin{bmatrix} r_R \\ t \end{bmatrix} \in \mathbb{R}^6. \tag{2.8}$$

2.1.4 Lens Distortion and Calibration

In the real world, parts of the camera such as the lens present distortions that make it so light does not behave as expected. One of the most famous cases is perhaps the one of *Fisheye lenses*, commonly used in photography, which present strong *barrel distortion*. Distortion is not covered by the traditional pinhole camera model described previously. There are two main types of distortion that must be attended for: *radial* (e.g., barrel, pincushion, and mustache distortions)

and *tangential* (caused by the elements of the lens not being perfectly aligned). Please note that chromatic aberrations, which are a real problem for vision models, are not included here as distortions.

Usually, these distortions can be mapped for each individual piece of hardware through a process known as *camera calibration*, which is also able to obtain the intrinsic camera parameters previously described in Eq. (2.1). This process can be done through multiple different methods, such as a direct linear transformation (Hartley & Zisserman, 2003), Zhang's method (Zhang, 2000), and Tsai's method (Tsai, 1987).

For software correction, distortion is usually mapped as a 1D vector of coefficients describing the disparities between the real-world camera and the ideal model, as well as expressing the individual distortion types. Using these, an image can be undistorted, as seen in Fig. 3.

(**a**) Original Image (**b**) Undistorted Image

Figure 3: An example of undistortion on a Fisheye lens.*

*Image publicly available by the CC BY 4.0 (Creative Commons - Attribution) license, courtesy of Sáez *et al.* (2019).

2.1.5 The Perspective-n-Point Problem

The P*n*P is one of the classical problems in computer vision. It is one of the ways of estimating the 6DoF pose of a camera by matching n 3D points to their equivalent 2D projections. In practical terms, for edge-based object tracking approaches like the proposed one, this means searching on the image for the 2D points and edges that most resemble the 3D ones found in the 3D CAD model and attempting to correlate them.

P*n*P has been solved by multiple methods, including the simpler P3P (Hartley & Zisserman, 2003) and the efficient EP*n*P (Lepetit *et al.*, 2009). The error can be minimized iteratively using approaches such as Random Sample Consensus (RANSAC) (Hartley & Zisserman, 2003) or a Levenberg–Marquardt optimization scheme (Levenberg, 1944).

2.2 Computer Graphics and Vision

This section will briefly go over some core concepts of computer graphics and computer vision used throughout the work. To understand these concepts in more detail, as well as other terms mentioned through the dissertation such as Bresenham's line and circle drawing algorithms

(very efficient, classic algorithms for drawing basic geometric shapes in 2D), depth buffering, and 3D mesh decimation algorithms (which is an active research field of its own), please refer to the Computer Graphics: Principles and Practice book (Foley *et al.*, 1996). For computer vision and image processing concepts mentioned in this work that go beyond what is explained below, including keypoints, descriptors, kernels, depth images, normalization, segmentation, thresholding, and gradient or contour extraction, please refer to the Digital Image Processing book (Gonzalez *et al.*, 2004).

2.2.1 Wavefront 3D Objects and Materials

The CAD models used in this work can be applied to any 3D manifold mesh. A manifold polygon mesh can be defined simplistically as any connected mesh that can exist in the real world. A connected mesh is manifold if every edge in the mesh is either a boundary edge or is part of exactly two other faces. A face in a Wavefront object can be defined as a convex polygon composed of usually 3 or 4 edges. For this dissertation 3 is assumed, so that the mesh is composed of triangular faces.

Wavefront objects, also known as OBJ files, usually specify the individual 3D vertices, 3D normal vectors for every vertex (not necessarily normalized), and texture coordinates (also known as UV coordinates) for every vertex, the latter expressed as 2D unit vectors. Faces are defined by the index of the vertices that compose them. The format supplies other less-common options, which are disregarded in this explanation.

These objects are usually accompanied by material (MTL) files, which provide texture and color details. These can be assigned to the whole object or to a subset of faces, so that a single OBJ can have multiple materials. These files usually contains ambient, diffuse, and specular RGB colors normalized as unit 3D vectors. For more complex distributions, a texture image can be assigned to each color, so that the vertices (and, by consequence, faces) are mapped to it by their UV coordinates. MTL files can provide multiple other details on how the object's material interacts with light, which are beyond the scope of this dissertation.

2.2.2 Color Spaces

In addition to RGB, there are multiple alternative color spaces to and from which an image can be converted. Among the ones most used for computer vision alone, there are HSV/HSL, CIELAB, YCbCr/YUV, and many others. Each different space has its own advantages and disadvantages. For this subsection, the focus will be on the RGB, HSV, and HSL spaces as can be seen in Fig. 4.

RGB is an *additive* color space defined by three main axes: red, green, and blue. This work limits itself to an 8-bit quantization, so that RGB values range from 0 to 255. Every color in this spectrum can be expressed in terms of a combination of the 3 additive primaries. This is very useful for things like displays that combine differently-colored light sources to color a pixel.

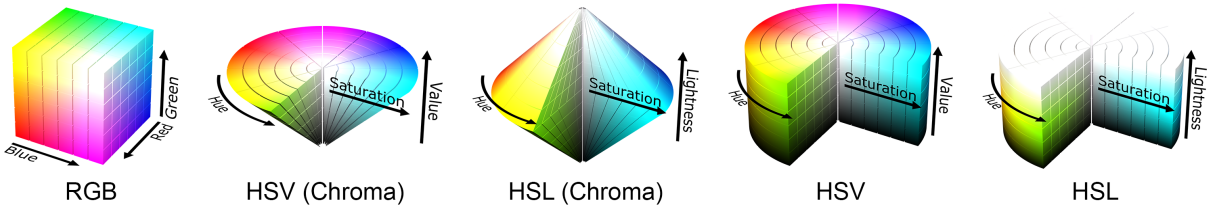


Figure 4: Representations of the RGB, HSV, and HSL color spaces.*

*Images publicly available by the CC BY-SA 3.0 (Creative Commons - Attribution) license, courtesy of Michael Horvath.

The problem with RGB's representation is that it is not very easy to work with in terms of human comprehension. For that end, the HSV and HSL spaces were created. These spaces are more closely aligned with the way humans perceive color, being more photometrically rooted. Photometry, in this sense, is defined as the science of measuring light in terms of its perceived brightness to the human eye. HSV and HSL spaces are so similar, in fact, that the technique proposed in this dissertation could work with either of them (though it has only been tested in HSV).

These two spaces have been widely used in computer graphics and vision for a very long time. The reason they are so useful is that they make the task of segmenting colors easier. For example, a red ball might get darker or lighter, might have specularities, and might suffer from slight color bleeding from other environmental sources. Still, it is fundamentally red, which is expressed in its *hue* channel. Some camera sensors might capture colors with a bit less intensity than others, or a real world object might get discolored over time, which might just alter the *saturation* channel. The same way, aforementioned illumination and shadows might, in some cases, just alter the *lightness* channel. These changes are sometimes very difficult to map and isolate in the sense of a subset of combinations of red, green, and blue when working with RGB. This is especially true when talking about colors that naturally mix the primary RGB colors.

Truthfully, there is enough interesting content about these colors spaces that many more pages could be spent on them. Yet, this dissertation will limit itself to this brief introduction. More information, such as details on how to convert values between these spaces, can be found in the previously-mentioned resources as well as in some academic works (Phung *et al.*, 2005; Saravanakumar *et al.*, 2011; Hidayatullah & Zuhdi, 2015). The usefulness of HSV and HSL will be further explored in the method of the proposed approach to deal with challenges like the reality gap and illumination.

3

RELATED WORK

As there are many possible tracking challenges regarding objects and scenes, different monocular real-time model-based 6DoF tracking approaches have been proposed over the years. Some of these are better suited for textured objects, such as feature-based ones, while others, like edge-based, excel in textureless objects (and even refractive ones). Different techniques also have limitations that suit them better to specific scenes.

3.1 Feature-Based and SLAM

So far, several well-known RGB real-time detection and tracking approaches make use of feature descriptors (Lowe, 1999; Bay *et al.*, 2006; Morel & Yu, 2009). These are extracted over multiple keyframes to estimate the pose of the object. These descriptors often arise from specific object characteristics such as gradients and color patterns. Consequently, these approaches tend to have difficulties when faced against objects with low texture, uniform color, or less-prominent geometric qualities. Additionally, robustness to certain challenges (e.g., changes in scene or object chrominance, illumination, scale, viewpoint, and motion blur) is generally anchored to the initial set of known features. Therefore, this set must be very carefully and reliably constructed for maximum invariance (sometimes a task which is delicate and time-consuming). Some of these approaches are coupled with Simultaneous Localization and Mapping (SLAM) techniques to match the entire scene where the object is currently located in. One such example is ORB-SLAM2 (Mur-Artal & Tardós, 2017), which has been adapted to track 3D rigid objects (Wu *et al.*, 2017) on a fixed scene. If the scene is altered or the object is moved, though, these methods tend to fail.

Non-model-based approaches (which use 3D reconstruction before tracking begins) will not be covered here. These techniques have their own advantages and limitations, working differently from what has been described so far. Industrial mobile approaches such as Apple's ARKit[1] usually work this way and make heavy use of other hardware such as neural engines and sensors (e.g., IMU). These approaches are beyond the scope of this work both in terms of not being monocular RGB and not being model-based.

[1] Apple ARKit

3.2 Region-Based

Recently, monocular region-based approaches have been proposed, mostly influenced by PWP3D (Prisacariu & Reid, 2012). These techniques extract a whole portion of the input frame and attempt to statistically increase the differences between the object and its surrounding background regions by the use of step functions. Discrepancies between the final segmented shapes are used to feed energy minimization functions which attempt to estimate the infinitesimal motion between frames. One such method, the *Region-Based Object Tracker*, best known as RBOT (Tjaden *et al.*, 2018), has so far yielded the most reliable results in versatile conditions. It is suitable to a wide range of objects and textures, being reliable even when under challenges in terms of motion and occlusion. RBOT also tracks multiple objects with linear scalability in terms of FPS. In terms of hardware, both RBOT and PWP3D require GPU usage (OpenGL[2] rendering in RBOT and CUDA[3] computations in PWP3D). RBOT's interaction between multiple objects as well as standard pose optimization both make use of GPU-rendered depth maps. Problems with these region-based techniques lie mostly in their need to perform heavy processing in large portions of each frame repeatedly, requiring high amounts of memory and parallelism to obtain real-time performance. This parallelism bottleneck comes to light especially when tracking multiple objects. In addition, these works suffer with object symmetries, as they rely mostly on the silhouette. A state-of-the-art work (Zhong & Zhang, 2019) has recently proposed a region-based hybrid approach which contains photometric and statistical constraints to aid in such failure cases, though it has not surpassed RBOT in the general scenario.

3.3 Edge-Based

Unlike real-time region-based, real-time edge-based approaches have been around for decades. The idea consists of taking the 3D model's edges and matching the overall expected shapes to the ones found in the frame, then adjusting the pose by minimizing the error. Some techniques attempt to match the entire model edges to the scene, those are called explicit-edge approaches and can yield robust results at a higher computational cost (Koller *et al.*, 1993; Han & Zhao, 2019). Most edge-based techniques, though, are based on point sampling heuristics (including the proposed method). These heuristics have arisen from the algorithm called RAPID (Harris & Stennett, 1990). It was the first available real-time 6DoF tracker and has served as a blueprint for a wide array of RAPID-inspired trackers and pipelines (Seo *et al.*, 2013b; Drummond & Cipolla, 2002; Han & Zhao, 2019; Lepetit *et al.*, 2005). Initially, these RAPID-based techniques have relied only on differences of color intensity, needing just a grayscale image. More recently, techniques like *Global Optimal Searching*, better known as GOS (Wang *et al.*, 2015), have achieved state-of-the-art edge-based tracking by applying

[2]OpenGL
[3]NVIDIA CUDA

color analysis to regions neighboring the edges. This is done to better match the object and avoid matching clutter and occlusion (Wang *et al.*, 2019; Seo *et al.*, 2013a). These edge-based techniques tend to struggle with highly textured objects, as the many internal gradients can cause confusion via erroneous internal matching. This limitation has been surpassed by the proposed approach, which has achieved state-of-the-art results for half of the OPT dataset objects, which have lots of internal edges (see Section 6.1). Additionally, these approaches are prone to fail in cases of heavy motion where edges become blurred. This limitation has been strongly diminished by the segmentation technique proposed in Section 4.4.

Edge-based trackers tend to be the most lightweight in terms of processing required and have been shown to work with refractive objects (Choi & Christensen, 2012). There are also edge-feature hybrid approaches that achieved promising results by matching the object's contours to feature descriptors. One such work, by Bugaev et al. (Bugaev *et al.*, 2018), far surpasses state-of-the-art results on the OPT dataset, although it is not real-time capable. Seo et al. (Seo *et al.*, 2013b) have merged RAPID's pipeline with SIFT features to track an object and its scene using phones, though with limited robustness to motion, clutter, and manipulation.

3.4 RGB-D and Machine Learning

For completeness sake, techniques which are not monocular but have relevant characteristics must also be mentioned. Some promising results have been obtained through machine learning approaches, which are very lightweight and work under extreme illumination conditions due to the depth data (Tan & Ilic, 2014). The downside of all these approaches lies in the higher hardware requirements when it comes to RGB-D cameras and high-end graphics cards. Works from Kehl et al. (Kehl *et al.*, 2017) (region-based) and Tan et al. (Tan *et al.*, 2017) (machine learning) have both been able to successfully use RGB-D information and, like the proposed work, require just a single CPU core. Tan et al., in particular, is even able to track multiple objects simultaneously. Deep learning has also been used, as seen in Garon and Lalonde's work (Garon & Lalonde, 2017). While it has not achieved the same speed of its machine learning counterparts, it was shown to be reliable in scenarios with strong occlusions or imprecise initialization.

4

METHOD

The proposed method uses the object's 3D model together with its material information. Models are defined as standard triangular meshes composed by a set of vertices, edges, and faces, without repetition of shared edges between faces. Materials can be assigned to the entire object or to a subset of faces. Diffuse texture images are assumed to be in RGB format with 8-bit quantization per intensity channel. The usage of 8-bits of color quantization per RGB pixel (i.e., each channel can have up to 256 different color intensity levels) can be thought of as a *common-ground* choice for computer graphics, used by similar approaches (Tjaden *et al.*, 2018) and widely supported both in terms of hardware and software. The color of faces with no diffuse texture specified in its material (as is the case with textureless objects such as solid-color 3D prints) is referred to as the RGB vector $C \in \mathbb{R}^3 \rightarrow \{0, \cdots, 255\}^3$. Such color is specified by the diffuse color coefficient of the face's material.

All frames are remapped to remove radial and tangential lens distortion. It is also assumed that the camera is precalibrated, with the focal length and principal point expressed in pixels as the 3×3 intrinsic parameter matrix K, as per the pinhole camera model.

The extrinsic parameter matrix E describes the 6DoF transform from world to camera space and is represented as a 3×4 matrix $[R \ t]$. The projection $p \in \mathbb{R}^2$ of an object point $P \in \mathbb{R}^3$ can be described in the specific usage case of this work as $p = \beta(K \cdot (E \cdot \tilde{P}))$, with $\tilde{P}$ being its homogeneous coordinate form. This way, β represents the conversion between $\mathbb{P}^3 \rightarrow \mathbb{R}^2$ so that $\beta(P) = [X/Z, Y/Z]^T \in \mathbb{R}^2$.

The pose E_i for the i^{th} frame can be estimated by approximating the object's infinitesimal rigid body motion between frames. This is done by using the i^{th} frame and the last known pose expressed by $E_{(i-1)}$ to estimate $\Delta t = t_i - t_{(i-1)}$ and ΔR, finally obtaining E_i.

An overview illustration of the proposed technique can be seen in Fig. 5.

4.1 Model Preprocessing

Before tracking begins, the 3D model is preprocessed and has its texture simplified. This is a one-time procedure. Every model face's diffuse texture image region (i.e., the diffuse image associated to the set of vertices, masked by the UV coordinates associated with each face

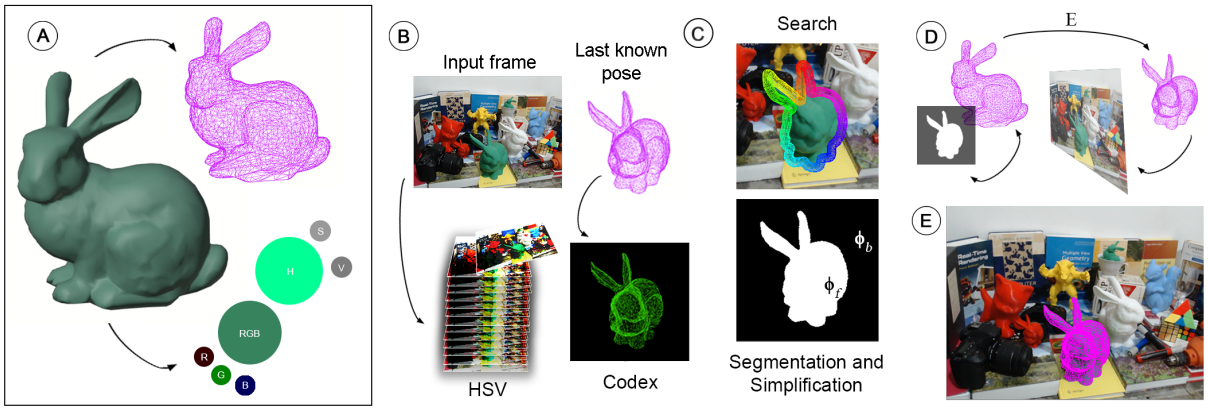


Figure 5: An illustrative overview of the proposed edge-based tracking pipeline for a single object. A) Preprocessing of the 3D model to extract color information and edges. B-1) Input frame processed into HSV and set for multiple iterations. B-2) Last known pose projected and encoded. C) Local circular color search to construct a segmentation mask. D) Establishing 2D-3D correspondences to perform pose estimation from world to camera coordinates. E) Tracking result for the frame.

vertex) is simplified by being quantized into a 32-bin RGB probability histogram. Each color's probability value corresponds to the percentage of pixels of said color in the face's histogram. Colors are then ranked according to said probability, highest first, and stored in HSV format. They are stored until either reaching a $k_{p1} = 95\%$ accumulated probability limit (i.e., all stored colors for that face represent over $k_{p1}\%$ of the face's surface) or reaching the $k_{p2} = 3$ color per face limit. For faces without diffuse texture, $C = 100\%$ probability. The k_{p1} limit was set to avoid storing unrepresentative colors and k_{p2} was set due to the textureless scope of this work, as an upper limit for the amount of colors and gradients per triangle.

Each face color is then assigned to every edge within the face. Thus, every edge has a list of all colors from all faces that contain it, without repetition. The list of edges, vertices that form them, and HSV colors they contain are then stored. The original object model and materials are no longer utilized.

4.2 Efficient Silhouette Encoding

The first step executed in real-time for every frame consists of encoding the 2D-3D correspondences for the current pose. All the model's 3D vertices are projected using the last known pose. Based on line projection properties, the 2D edges are drawn using the projected vertices and Bresenham's line-drawing algorithm. The canvas used consists of a single-channel 32-bit image. Each pixel of a projected edge has its intensity value corresponding to its own 3D edge index from the edge set ξ contained in the preprocessed model. This is similar to the Edge-ID algorithm (Lima *et al.*, 2009), but uses a single 32-bit channel directly for the ID instead of splitting it in three 8-bit channels. The resulting image is referred as the *codex*. It is important to mention that the implementation does not utilize depth testing (details in the Appendix A).

Next, a topological border following algorithm (Suzuki & Abe, 1985) is used on the codex to extract the external-most 2D contour ζ. The contour is continuous and returned in counter-clockwise order. This set of pixels corresponds to the object's 2D silhouette at the last known 6DoF pose.

Finally, the set of 2D normal vectors Υ is determined so that $\Upsilon_i = \tilde{\Theta}(\zeta_{(i+1)} - \zeta_{(i-1)})$, as ζ is circular and continuous. The vector $\tilde{\Theta}$ corresponds to the normalized output of the function $\Theta(v) = [-y,x]^T$, which returns a 2D vector orthogonal to $v = [x,y]^T \in \mathbb{R}^2$. The normal vector is always rotated 90^o clockwise. Given the counter-clockwise property of the contour's continuity, this means the 2D normals will automatically always point outwards from the silhouette (see Fig. 6).

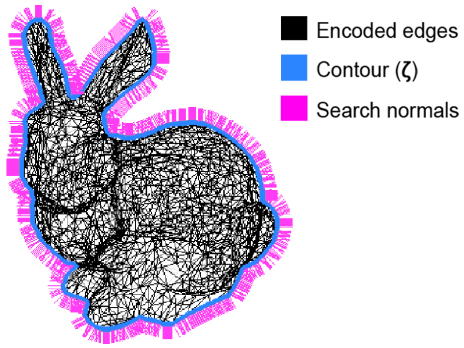


Figure 6: Illustration of the final encoding output containing projected mesh edges, outlined contour, and normal vectors.

To extract a 3D point P from the codex's 2D point p, because only the edge vertices were projected, it is first necessary to linearly interpolate those two endpoint Z values to obtain P_z. Then, $P = [(p_x - c_x)/(P_z \cdot f_x), (p_y - c_y)/(P_z \cdot f_y), P_z]^T$ (Lima, 2014), where c_x, c_y, f_x and f_y come from the intrinsic camera matrix K defined previously. To find those two endpoint z values, the edge ID is checked in the codex and verified in the edge array for which pair of 3D vertices compose that edge. This operation is only performed when a match is found for that silhouette pixel, which is more efficient than calculating all z values using a depth buffer.

4.3 Adaptive Circular Search

In order to segment the frame in real-time, a search range must be established. Instead of conducting 1D linear searches directly (as is traditional in edge-based approaches), the search is performed locally in dense circular areas with radius r, centered around contour points from ζ (drawing inspiration from RBOT). The goal of this search is to match the colors found in the frame with the ones expected from that region of the preprocessed 3D model (see Section 4.4). Additionally, instead of sampling 3D points at random, consecutive points are taken directly from ζ in 2D. This is done to maintain uniform contour coverage as the projected silhouette's length varies with z-axis translation.

Ideally, every point in ζ should be evaluated in its own circle, covering a dense strip around the object's curvature. Because that is too computationally expensive, every r^{th} consecutive contour point is sampled instead (see Fig. 7). This makes it so that every circle is always equally re-visited by each of its two immediate neighbors, increasing coverage. Neighbors checking the circle's area can be useful when there's larger motion and object dislocation between frames or when colors are blended due to blur.

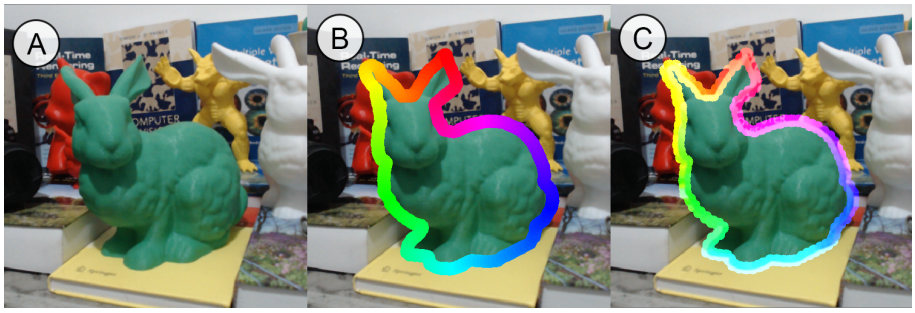


Figure 7: Segmentation search coverage. A) Input frame. B) Ideal case. C) r-spaced search.

Instead of using a fixed radius value (as is commonly done in tracking approaches), it is hereby suggested to change the radius according to the object's current silhouette area in relation to the frame resolution. In the proposed way, r is updated at every new pose, proportionally to how many pixels are being occupied by the object's projection. This projection region consists of all pixels inside (and including) ζ, referred to as $A(\zeta)$, the area of pixels covered by ζ. Thus, the number of pixels inside this region is represented by $|A(\zeta)|$. The radius r can be calculated as a linear function with limiting thresholds:

$$r(\zeta) = min\left(\left\lfloor\left(k_\theta \cdot \lambda_A^{-1} \cdot |A(\zeta)| + r_0\right) \cdot \lambda_S\right\rfloor, r_1\right), \tag{4.1}$$

where $r_0 = 7$ pixels is the minimum tolerable level of detail where shapes are still discernible for the PnP-based approach used, determined empirically. The angular coefficient $k_\theta = 0.002$ determines how fast the radius grows with proportion to the object size. The value of the maximum allowed radius is defined as $r_1 = \sqrt{k_A \cdot |A(F)|/\pi}$. This equation makes it so the area in pixels of a circle of radius r can not be larger than $k_A\%$ of the frame's pixels, where $|A(F)|$ is the total area (in pixels) of the input frame F and $k_A = 0.01$, or 1%, corresponds to the percentage of the frame that the maximum radius is allowed to occupy. The suggested values for k_θ and k_A were found via experimentation with multiple handheld objects.

Variable $\lambda_A = |A(F)|/(v_w \cdot v_h)$ represents the scale factor between the current resolution and v, the *base resolution* (in this case VGA). It makes it so the technique behaves uniformly under resolution changes. Values v_h and v_w represent height and width of v, respectively.

Finally, $\lambda_S = min(F_w, F_h)/min(v_w, v_h)$, where F_h and F_w represent height and width of F, respectively. The equation for λ_S represents the shortest side scale. It minimizes changes when under different (e.g., wider or narrower) aspect ratios. An alternative to using λ_S to handle

changes in aspect ratio is to use directly-proportional elliptical search circles, but that would divert from the circular search concept by creating directional search bias.

4.4 Binary Segmentation

A binary segmentation mask is constructed by evaluating every pixel within the search circles described previously. In an ideal world, where the model texture corresponds exactly to what will be found in the frame, the binary segmentation is very straightforward. Yet, due to the *reality gap*, tolerances must be established for a pixel's color to be considered a match. This *reality gap* concept is widely used for measurements in robotics and simulation. For the detection and tracking scenario, it can be described as the inherent difference in appearance that exists between a virtual object and its real counterpart, no matter how photorealistic is the said virtual object or how pristine is the real object's shape, color, and surface. Such differences might be due to aspects other than the object itself as well, such as illumination or camera imperfections.

The aforementioned reality gap segmentation tolerances used for pixel color matching are controlled by what is here called the *uncertainty coefficient*, or σ. The idea is that in a scenario where the developer (or the user) is very sure the texture is accurate, such as for 3D printed objects or synthetically-rendered scenes and applications, the uncertainty (and therefore tolerance to variation) is low. But in a real world scenario, that tends to be larger (see Section 6.1).

The core idea of the proposed segmentation approach consists of utilizing the photometric properties of the HSV space to help in overcoming reality gap challenges and select pixels. By utilizing hue and saturation information, a preprocessed color can be more easily identified in cases where, if using a space like RGB, it might not be so clear. Because this work relies on HSV, the approach focuses on pigmented objects. Grayscale (non-pigmented) objects do not possess reliable hue information (see the Appendix B).

To take advantage of the HSV properties, this technique borrows a concept from region-based approaches: the smoothed unit step function (see Fig. 8). In this case, the idea is to statistically highlight the difference between regions of HSV space itself, as to continuously regulate the uncertainty tolerance levels proportionally to the amount of pigmentation present.

This way, a segmented frame pixel p must satisfy the following conditions with respect to the preprocessed color c in order to be set as 1, or true, in the binary segmentation:

$$H(p_s) \geq d_\theta(p_h - c_h) \ \wedge \ p_s > k_s \ \wedge \ \eta \geq |p_s - c_s|, \tag{4.2}$$

where subscript h and s represent the hue and saturation channels, respectively. Function d_θ represents the shortest distance between hue values (considering they are cyclic) and constant $k_s = 0.05$ represents the minimum saturation threshold, indicating the point where there is so little pigmentation hue is no longer reliable (determined via experimentation as demonstrated in Appendix B). The saturation tolerance range is mapped by the logarithmic function $\eta =$

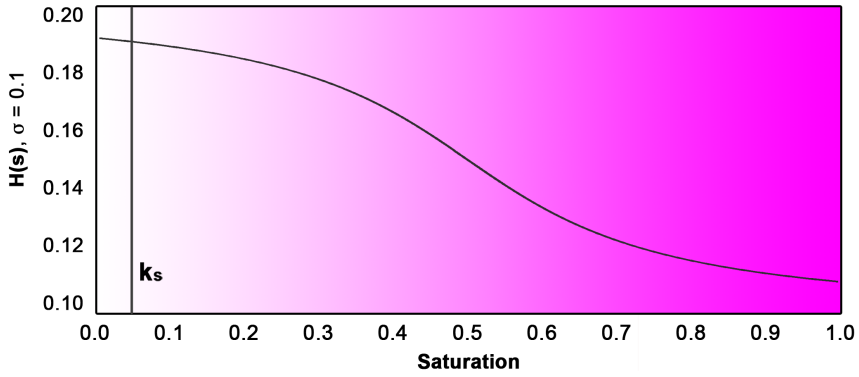


Figure 8: Behavior of $H(s)$ for $\sigma = 0.1$ as saturation changes. Desaturation threshold k_s is also highlighted.

$k_l \cdot ln(\sigma) + k_\eta$, where the vertical shift $k_\eta = 0.5$ sets the log curve's plateau at the maximum saturation tolerance window possible. The elasticity coefficient $k_l = 0.08$ both determines how fast that plateau will be reached and sets the function within the 0 to 1 quantization. Finally, H represents the hue tolerance proportional to the amount of pigmentation present, represented by a smoothed unit step function, adapted as follows:

$$H(s) = \sigma \cdot \left[\frac{3}{2} - \frac{1}{\pi} \cdot tan^{-1} \left(\frac{s - 0.5}{k_H} \right) \right], \tag{4.3}$$

where $k_H = 0.2$ represents the sharpness coefficient (Wang & Qian, 2018).

4.5 Mask Simplification

After every pixel in the search circles is subjected to the binary segmentation described in the previous section, an additional step is performed to avoid noise and erroneous background bits in the resulting mask. First, all existing external-most contours are identified (Suzuki & Abe, 1985) and the largest contour in terms of area is stored. This is assumed to be the main tracked object body (or body portion). Every contour with area less than $k_m = 5\%$ of the largest contour's is removed (see Fig. 9). This heuristic does not work if the erroneous portion is connected to the main body.

From this point, the binary mask will be referred to as Φ, so that $\{\Phi_f, \Phi_b\} \subseteq \Phi$ refer to the foreground (true) and background (false) pixel subsets, respectively.

4.6 Correspondence Search

The next step consists of searching for a match for every point in ζ. This is done by using a traditional (Drummond & Cipolla, 2002; Seo *et al.*, 2013b) edge-based linear 1D search for gradients along the normal vectors from Υ. This search uses the kernel $\psi = \begin{bmatrix} 1 & -1 \end{bmatrix}^T$ along the parametric pixel line of vector $dt\Upsilon_i \in \mathbb{R}^2$ over point $\zeta_i \in \zeta$, described as $\zeta_i + dt\Upsilon_i$. This linear

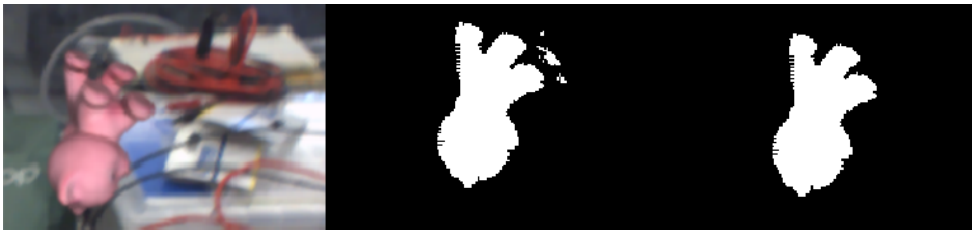


Figure 9: Mask simplification process. From left to right: input frame, initial segmentation, final result.

search can thus be expressed as

$$S(i,t) = \psi \cdot \left[\Phi_{\zeta_i + dt\Upsilon_i} \quad \Phi_{\zeta_i + d(t+1)\Upsilon_i} \right], \tag{4.4}$$

where $d \in \{-1,1\}$ corresponds to the search direction (inwards or outwards) so that $d = 1$ always points outwards from the model contour. The parametric term $t \in \{0, r-2\}$ is limited to $(r-2)$ to avoid matching with the edges of the circular search masks, as this indicates that the accepted pixel region would continue beyond the search range and does not correspond to a border. These cases are usually indicative of same-color background regions. The minimum range of 0 deals with cases where the object did not move at all.

Eq. (4.4) is calculated for every value of t until a result different from 0 is obtained. When and if that happens, the 2D-3D point tuple $(\mu_i,\ \zeta_i + dt\Upsilon_i)$ is stored in the vector χ as a possible match candidate. The 3D point μ_i represents the unprojected 3D equivalent of ζ_i from the codex.

Instead of alternating d values while t increases (Seo *et al.*, 2013b), the direction of d is defined as per Table 1. Set $W = A(\zeta) - \iota(\zeta)$, where $\iota(\zeta)$ is a mask indicating the search range, where all pixels for every search circle for every chosen point in ζ are set to 1. Thus, $O = W(\zeta) \cap \Phi_f$ represents whether there seems to be an occlusion in Φ_f. This is due to no pixels in W being reachable by the search circles, but being within the contour, so that they can only be set if the contour was not interrupted by occlusion and could thus be filled, as per Fig. 10.

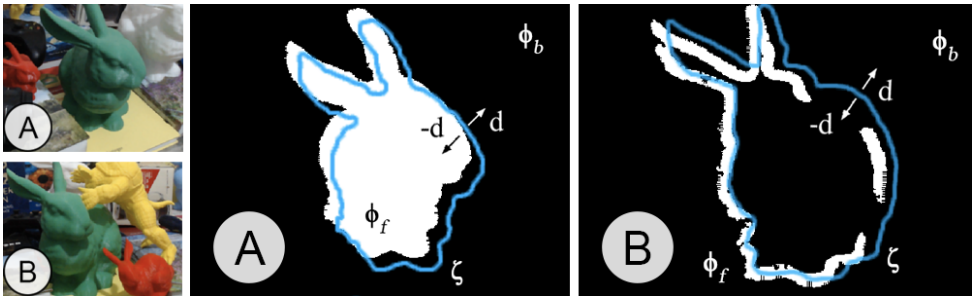


Figure 10: Segmentation without (A) and with (B) occlusion.

If the previous pose contour point $\zeta_i \in \Phi_f$, then the search continues outwards ($d = 1$) otherwise, if ζ_i is outside Φ_f, the search goes inwards ($d = -1$). If Φ_f is not filled and ζ_i is not in it, then it can be either outside or inside the current pose. Thus, it is first assumed to be inside the model and, to avoid matching the search strip, the first match going outwards is ignored, so

that only the second one is valid. If no two matches are found, then the point is likely outside the model and the search must look for the first match inwards.

Table 1: Guidelines for directional search and matching.

$\zeta_i \in \Phi_f$	$O = \varnothing$	d	**match stored in** χ
T	✖	1	1^{st}
F	F	-1	1^{st}
F	T	1 then -1	2^{nd} then 1^{st}

It can be argued that this segmentation step is an interesting addition to be considered for edge-based approaches because it circumvents the background gradients problem by postponing the gradient matching step to the binary mask. The downside remains erroneous color-based segmentation, which can be adjusted according to the limitations of the chosen segmentation module for each technique.

4.7 Iterative Pose Estimation

Given a set of 2D-3D correspondences χ, the current pose is estimated as the solution of a Perspective-*n*-Point (P*n*P) problem. The resulting matrix is iterated over by a Levenberg-Marquardt (LM) optimization scheme, using the last known pose as an initial estimation. The process lasts until either a local minimum for the reprojection error is found or 20 iterations are performed. The reprojection error can be defined as the sum of squared Euclidean distances between the 3D points' 2D projection (using the current pose estimation) and the original 2D matches. More details on the P*n*P problem and how to apply LM optimization to it can be found in the reference survey by EP*n*P's author Vincent Lepetit (Lepetit *et al.*, 2005).

5

ADDITIONAL DETAILS

This chapter provides details of the proposed tracker's C++ implementation. Further explanation is also given for the multi-object and mobile scenarios. Finally, Section 5.3 describes the accompanying tools proposed for augmented reality rendering and tracker benchmarking.

5.1 Implementation

All major steps are performed using a single CPU thread. The implementation relies on OpenCV[1], which is optimized for computational efficiency and multi-platform portability. More low-level details on the implementation of the technique can be found in Appendix C.

The mobile port utilizes the same C++ code via Android's Native Development Toolkit (NDK)[2]. Frames, in this case, are captured in Java through OpenCV for Android's *CameraView*, which is limited to 30 FPS but provides the frames as matrix objects compatible with the C++ implementation.

Every new input frame is converted from OpenCV's native BGR (or YUV in Android's case) to HSV and iterated by each tracker instance 12 times. Every iteration performs steps from Sections 4.2 through 4.7.

5.2 Tracking Multiple Objects

The technique has been extended to track multiple simultaneous objects without compromising the frame rate by having each tracker instance (one object per instance) running on a separate CPU thread. The amount of objects that can be tracked in parallel depends on the number of physical CPU cores.

Threads are managed in a producer-consumer scheme with the same number of threads and tracker-object instances, no instance being tied to any specific thread. One tracking job is queued per instance at every new frame. New frames are only processed after all jobs on the previous frame have finished, making it so the multi-object tracking is as fast as the slowest

[1] OpenCV
[2] Android NDK

tracker instance. The approach is not very suitable to mobile CPUs, as cores do not usually share the same clock speed. Thus, the Android port does not currently support multi-object tracking.

5.3 Additional Tools

5.3.1 Augmented Reality

An AR rendering engine has been implemented using OpenGL and C++, for desktop platforms only. It is mostly GPU-based and its CPU portion shares the same thread pool as the tracking queue (see Section 5.2), executing after all the present frame's jobs are done. This engine renders the object according to the provided input vertex and fragment shaders and can be used for displaying results as well as various sorts of AR experiences (see Fig. 1).

Tracked objects are rendered according to the most recent pose estimation performed by their respective tracker instance. The current camera frame is used as background. For augmentation (i.e., adding virtual elements to the real objects), tracked objects must be rendered first. In most AR experiences, those objects are only written to the depth buffer and not to the color buffer. This makes the objects seem invisible (i.e., the pixels where they are located display the background frame, which corresponds to the real object). Yet, depth buffering makes it so they are not transparent (i.e., they interact with and occlude other virtual objects).

5.3.2 3DPO Dataset

A real-world qualitative set of sequences, named the 3DPO dataset (3D Printed Object dataset), together with CAD models (see Fig. 11) and camera calibration is being made available. The set has scenes with multiple high-quality 3D-printed objects for the challenges of clutter, extreme motion, heavy occlusion, handling, and independent object-camera motion. It also contains two sequences with ground truth. More details in Appendix D.

Figure 11: Objects used in proposed sequences.

6

EXPERIMENTS AND EVALUATION

Experiments were executed in a laptop with a dual-core Intel Core i7 7567U CPU @ 3.50 GHz, 16 GB of RAM, and integrated Iris 650 graphics. Results are compared to the current state of the art using the publicly available implementation of RBOT[1] (see Appendix E for details on this setup). Values used for the reality-gap uncertainty were $\sigma = 0.1$ on OPT, $\sigma = 0.01$ on RBOT dataset, and $\sigma = 0.05$ on 3DPO (unless where explicitly specified). Initial poses were provided with the datasets, requiring no detection module.

6.1 Benchmark Comparison

For benchmark comparisons, this work used the publicly available datasets OPT[2] at 1920×1080 resolution and RBOT dataset[3] in its original resolution (640×512). Experiments were conducted using the AUC metric as described by OPT (Wu *et al.*, 2017). The pose error per frame ($e_{AUC} \in \mathbb{R}$) can be described as

$$e_{AUC} = \frac{1}{n} \sum_{i=1}^{n} ||E_{est} \cdot \tilde{v}_i - E_{gt} \cdot \tilde{v}_i||_2, \tag{6.1}$$

corresponding to the average Euclidean distance for all n CAD model vertices $v_i \in \mathbb{R}^3$ in homogeneous coordinate form, when subjected to both the pose $E_{est} \in SE(3)$ estimated by the tracker for that frame (see Eq. (2.6)) and the ground truth $E_{gt} \in SE(3)$ provided with the dataset. If $e_{AUC} < d_{AUC} \cdot k_{AUC}$ then the pose estimation for that frame is deemed successful. Constant $k_{AUC} \in \mathbb{R}$ corresponds to the predetermined error threshold and $d_{AUC} \in \mathbb{R}$ corresponds to the largest distance between two model vertices, representing the model's diameter. The percentage of dataset frames with successful estimations is then plotted as a curve against increasing values of $k_{AUC} \in \{0, 0.2\}$. The integration of such curve is known as the *area under curve* or AUC. Results can be seen in Table 2 and Fig.12.

AUC results from Table 2 show that Chest, House, and Ironman were reliably tracked, as they contain the least non-pigmented parts and are less textured (better fitting the scope). One

[1] github.com/henningtjaden/RBOT
[2] media.ee.ntu.edu.tw/research/OPT/
[3] cvmr.info/research/RBOT/

Table 2: AUC score table for the OPT dataset.

Method	Soda	**Chest**	**House**	**Ironman**	Bike	Jet	Average
Proposed	4.11	**11.76**	**13.45**	**15.70**	0.04	0.94	**7.67**
RBOT	**6.62**	9.41	6.84	7.36	**6.44**	**8.74**	7.57

interesting aspect of the proposed edge-based binary approach is that it is able to more finely match the object's silhouette than probabilistic approaches like region-based (see Fig.12).

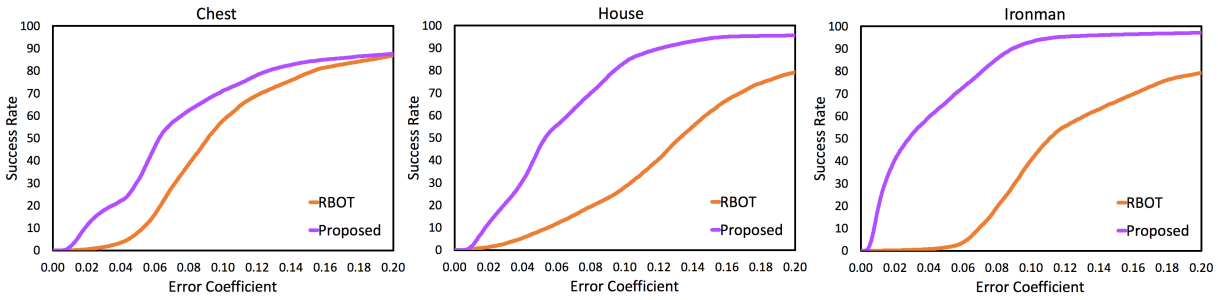


Figure 12: AUC curves for Chest, House, and Ironman.

The proposed method particularly struggled with objects Soda, Bike, and Jet. That was expected, as these objects are highly textured. In addition, Bike and Jet contain many non-pigmented portions and Soda has a highly symmetrical silhouette. In the case of Bike (Bugaev *et al.*, 2018) and Jet, models and textures were also not very accurate. Thus, it is safe to say that these three objects presented a worse-case scenario for this work (see Fig.13).

Figure 13: Failure case which shows the proposed work attaching itself to the pigmented portions the Bike object.

Table 3 shows more detailed results per OPT challenge. There it can be seen that the proposed work particularly excels in handling illumination changes. This is mostly due to the segmentation utilizing hue information. In terms of handling motion, the technique's ability to swiftly recover from partial loss has been beneficial (see Section 6.2).

A comparison was performed to evaluate how the segmentation manages motion blur. Segmented pixels and ground truth masks using all Level 5 motion (fastest) OPT sequences were used. Results in Table 4 show that the approach successfully handles blur, overcoming the blurred-edges problem of edge-based techniques by using the pigmented information.

Combined tests were also performed as to better understand the impact of dynamic radii. The test compared variations on search range (fixed) to Eq.(4.1) in the Scale Changes (zoom)

Table 3: AUC scores per challenge

Object	Method	Illum. Changes	Free Motion	Scale Changes	Trans-lation	Rota-tion
Chest	Proposed	**8.75**	**9.77**	**13.69**	10.19	**12.23**
	RBOT	6.64	2.43	7.08	**10.86**	11.67
House	Proposed	**15.19**	**13.70**	**7.78**	**16.12**	**14.67**
	RBOT	11.36	2.05	3.40	11.02	7.82
Ironman	Proposed	**16.34**	**10.86**	**14.75**	**14.92**	**16.85**
	RBOT	6.52	5.45	5.78	5.41	8.81
Average	Proposed	**13.43**	**11.44**	**12.06**	**13.74**	**14.58**
	RBOT	8.17	3.31	5.42	9.10	9.94

Table 4: Average results for motion blur segmentation test.

Measure	Chest	House	Ironman
False Negatives (%)	3.50	1.98	6.82
True Positives (%)	98.75	96.99	96.74

scenes of the OPT dataset and the Cat scene from the RBOT dataset. Table 5 shows dynamic radius is preferable for generality purposes. The metric of the RBOT dataset as proposed by RBOT (Tjaden *et al.*, 2018) is simpler than the AUC, as follows: if the estimated 4×4 pose matrix (as per Eq. (2.6)) is either over 5 cm of translation or 5^o of rotation away from the ground truth, the estimation for that frame is deemed to have failed. The score corresponds to the percentage of the 1000 estimated frames with accepted poses. Unlike in OPT where trackers are only reset at the first frame of every sequence, in the RBOT dataset whenever a pose is rejected the tracker is reset to the ground truth before the next frame is processed.

Table 5: Results for different search ranges (in px).

Object	AUC Scores (OPT Dataset)				
	$r = r_0$	$r = r_1/4$	$r = r_1/2$	$r = r_1$	Eq.(4.1)
Chest	13.65	13.71	**13.90**	**13.90**	13.69
House	2.48	7.72	7.81	**8.15**	7.78
Ironman	14.03	**15.63**	11.59	5.92	14.75
Average	10.05	**12.35**	11.10	9.32	12.07
	RBOT Dataset Scores				
Cat	57.90	67.30	94.10	88.70	**97.20**

Next, the textureless pigmented objects from RBOT's proposed dataset were used together with its proposed metric (Tjaden *et al.*, 2018) on the scene *regular* to evaluate robustness to similar-color backgrounds by varying σ tolerance (see Table 6).

Numbers for the object Cat show that the proposed work excels in handling similar-color

Table 6: Evaluation on varying σ values. Percentages S, TP, and FN stand respectively for score (Tjaden *et al.*, 2018), and averages of true positives and false negatives (in %) for the segmented pixels with respect to the ground truth segmentation masks.

Object	**Proposed** (varying σ values)									**RBOT**
	0.10			0.05			0.01			
	S	TP	FN	S	TP	FN	S	TP	FN	S
Ape	36.7	82.6	9.5	**48.4**	92.5	**7.6**	47.4	**98.1**	33.5	85.2
Duck	43.8	96.2	**5.5**	**52.2**	99.7	6.1	44.9	**100.0**	21.6	87.1
Cat	13.5	43.7	18.8	50.4	81.2	**2.7**	**97.2**	**99.5**	7.1	95.5
Squirrel	25.4	67.8	18.5	37.5	77.2	**15.5**	**70.5**	**97.8**	34.2	99.2

background when there's little uncertainty, even surpassing RBOT's high score, but suffers from background matching otherwise. Such low uncertainty values are only possible because the materials are very accurate to the objects, which means the technique requires high-quality 3D model materials to perform well under similar-color backgrounds. Ape and Duck were precisely segmented as well, but their quick-rotating, simple silhouettes created inconsistencies in the estimated poses. Squirrel was not properly segmented even at low uncertainty due to large portions of *same-color* background, which is a limitation of the proposed segmentation approach (different from *similar-color*).

For completeness sake, the different challenges from the RBOT dataset were also evaluated for the Cat object, still using RBOT's metric. Results from Table 7 show the technique handles such challenges comparably to RBOT.

Table 7: Comparison of RBOT scores per challenge for the Cat object (higher is better).

Method	**Regular**	**Dynamic Light**	**Occlusion**	**Average**
Proposed	**97.2**	**97.0**	87.0	**93.7**
RBOT	95.5	95.2	**90.1**	93.6

One thing to note is that RBOT's metric is very strict, resetting the tracker when it is not necessarily lost. Thus, to show how the proposed technique is able to recover from partial tracking losses on its own, a no-reset methodology (Garon & Lalonde, 2017) was used on the RBOT dataset. The translation and rotation errors were plotted separately for the regular and occlusion scenes, without ever resetting the trackers since the first frame. Results from Fig. 14 show that the proposed work is always able to recover the object's position (translation), whereas RBOT is unable to recover since the first loss.

Because detection module calls are usually computationally expensive, this reduced need is highly desirable. In addition, this adds flexibility to minor losses right from the start or the tracking process, meaning there's less requirement for a robust detection module. This fits well with the proposed low hardware requirement application scenario, especially considering that

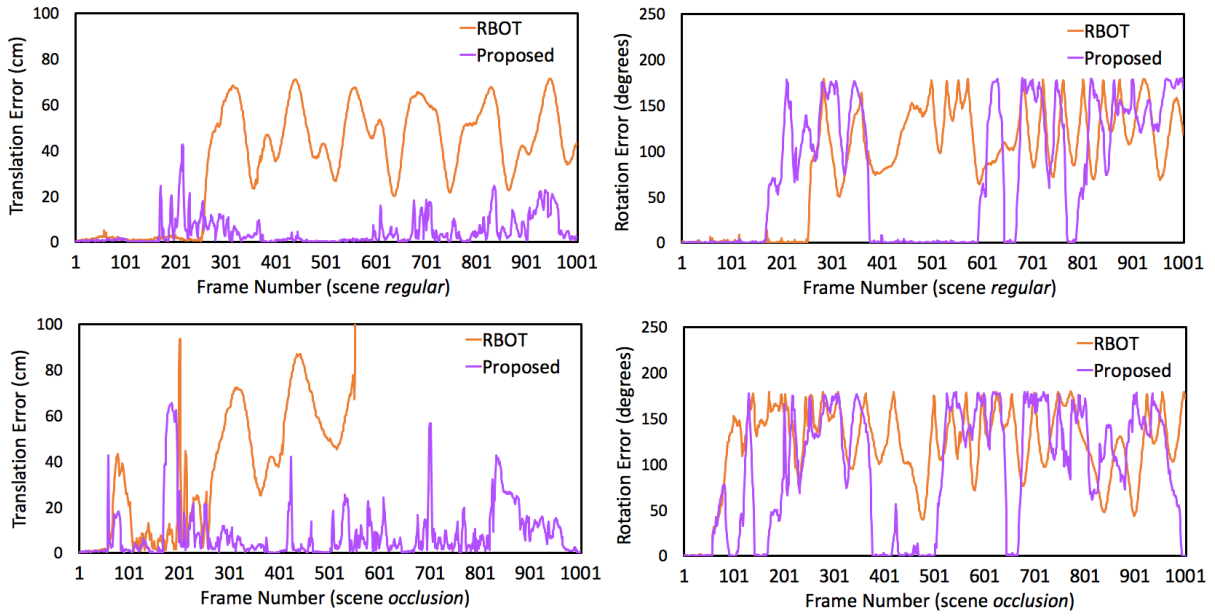


Figure 14: Recovery from partial loss results.

popular mobile trackers can be manually-aligned by a user[4], given that state-of-the-art detectors use deep learning and powerful GPUs.

6.2 Imprecise Initialization

To further understand how robust the proposed technique is to an imprecise previous pose, a very extensive Imprecise Initialization test (Garon & Lalonde, 2017) was conducted. For this test, 10 frames were selected at random from the sequence Slow of the 3DPO dataset, which has 1155 frames at 30 Hz. This sequence (unlike OPT ones) has a lot of clutter and background gradients. Unlike the RBOT dataset's, the sequence is also not synthetic, containing motion blur (an important aspect for temporally-consistent real-time tracking). It consists of smooth camera movements around a still Green Bunny, with comprehensive camera rotations and translations while moving closer and farther away from the object. Ground truth was obtained with the help of ArUco markers.

Initial pose rotation and translation perturbations are measured separately. Rotation perturbations range from 5 to 60^o and are incremented 5^o at a time. Translation perturbations range from 10 to 130 mm in 10 mm increments. For each increment, 40 random samples are taken. Before computing the error, the tracker is called 15 times for each frame to take the temporal factor into consideration. Mean and standard deviation for the L2 norms of the errors (with respect to the ground truth) of each of the 10 sample for each of the perturbations are plotted in Fig. 15.

The proposed technique has achieved consistent translation robustness for up to 60 mm,

[4]Vuforia Model Targets

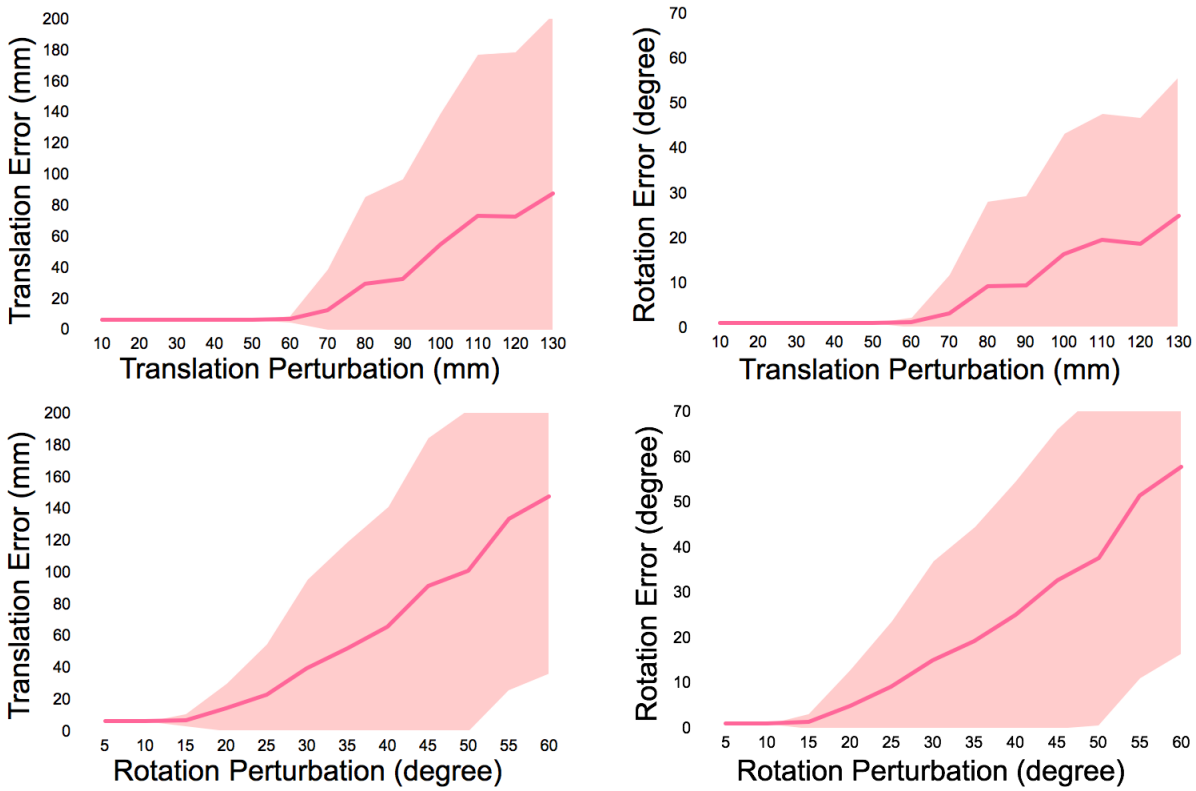


Figure 15: Initial perturbation test. Lines represent the average error. Margins represent the standard deviation.

which corresponds to roughly 30% of the model's size. For rotation, the technique achieved constantly reliable results for up to 15^o perturbation. The deep learning work that first proposed this methodology has also performed well in this test (Garon & Lalonde, 2017) (albeit using a different RGB-D sequence and object). It achieved robustness levels of 100 mm and 30^o. In their case, though, all perturbation averages (even the smallest ones) had at least 8 mm and 8^o of error. This is a good illustration of the trade-offs between geometric and machine learning methods. The similarity between these very different approaches is that both are supplied with some sort of previous knowledge regarding the model's color and texture.

Model-based techniques do not tend to directly assess this imprecise initialization issue, simply falling back to a costly detection module. Since real-world applications usually can not count with a ground truth value, robustness to detection error is a relevant factor and should not be overlooked. This serves as argument for using texture information alongside geometry on model-based approaches.

6.3 Performance and Scalability

The first frame from the RBOT dataset sequence *regular* and the object Cat (with 2500 vertices) were used for this test. More object instances were added on top of the same object to simulate a multi-object scenario. The proposed work's memory requirement was orders of

magnitude smaller than the state of the art's, at $\sim$ 10 MB per object tracked simultaneously, on top of a baseline value of $\sim$ 10 MB. RBOT's memory consumption was $\sim$ 4.3 GB per object, which makes multi-object impractical for most low-end devices. In terms of runtime, RBOT scaled linearly with $\sim$ 70 ms per new object. The proposed work had an initial runtime of $\sim$ 30 ms, with approximately $\sim$ 3 ms per new object until a limit of 3 objects (amount of threads able to run in parallel efficiently on the dual-core CPU). Afterwards, the proposed work's runtime scaled linearly (like RBOT) with $\sim$ 30 ms per new object.

The second test had a similar setup but used only one object at a time. Cat objects with different simplification levels (amount of vertices) were used, as follows: 2500, 5000, 7500, 10000, 12500. RBOT scaled at $\sim$ 4.3 GB of RAM per new 2500 vertices, whereas the proposed work showed no significant difference, staying at about 20 MB. In terms of runtime, RBOT scaled $\sim$ 10 ms per 2500 vertices from a baseline of $\sim$ 70 ms. The proposed work scaled about 2 ms per 2500 vertices, from a baseline of $\sim$ 30 ms. The single-object mobile port of the proposed work ran at $\sim$ 50 ms on a Samsung Galaxy S10 phone.

In terms of pipeline time costs, this work averaged roughly 2.5 ms per pipeline iteration, consisting of: encoding, segmentation, matching, and pose estimation. Preprocessing averaged 112 ms per 100 triangles with diffuse texture and 3 ms for those without.

7

CONCLUSION

This dissertation has presented a novel edge-based 6DoF tracking technique focused on textureless, pigmented objects. The technique, for the intended scope of objects, achieves comparable results to the present state of the art while surpassing it in relevant aspects. It particularly excels in challenges like illumination changes, motion, and recovering from imprecise poses. The technique has been shown to be well suited to either manual alignment or imprecise automated detection modules. It is also single-core, RGB-only, and much more lightweight, thus more well suited for less-powerful hardware (e.g., mobile) as well as multi-object tracking. Other contributions beside the main technique include a new way to avoid fixed search ranges, an efficient silhouette encoding technique, a brief study on using material information for model-based 6DoF object tracking, and a new segmentation step which handles problems such as background gradients and gradients lost to motion blur. These contributions have enabled edge-based tracking to be once again competitive (in terms of precision and robustness) against the region-based state of the art, albeit for a smaller scope of objects. Finally, sequences focused on 3D printed textureless objects and an AR rendering tool are both being made available, hopefully to enrich further research in the area.

7.1 Future Work

As future work, the technique's main limitations have proven to be symmetry and imprecise textures along with similar of same-color backgrounds. The former can be aided by photometric constraints (Zhong & Zhang, 2019) and the latter can be tackled by using temporal foreground-background segmentation strategies as performed by region-based approaches.

REFERENCES

Bay, H., Tuytelaars, T., & Van Gool, L. (2006). Surf: Speeded up robust features. In *European conference on computer vision*, 404–417.

Bugaev, B., Kryshchenko, A., & Belov, R. (2018). Combining 3d model contour energy and keypoints for object tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 53–69.

Choi, C. & Christensen, H. I. (2012). 3d textureless object detection and tracking: An edge-based approach. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3877–3884.

Drummond, T. & Cipolla, R. (2002). Real-time visual tracking of complex structures. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):932–946.

Foley, J. D., Van, F. D., Van Dam, A., Feiner, S. K., Hughes, J. F., Angel, E., & Hughes, J. (1996). *Computer graphics: principles and practice*, volume 12110. Addison-Wesley Professional.

Garon, M. & Lalonde, J.-F. (2017). Deep 6-dof tracking. *IEEE transactions on visualization and computer graphics*, 23(11):2410–2418.

Gonzalez, R. C., Woods, R. E., & Eddins, S. L. (2004). *Digital image processing using MATLAB*. Pearson Education India.

Han, P. & Zhao, G. (2019). A review of edge-based 3d tracking of rigid objects. *Virtual Reality & Intelligent Hardware*, 1(6):580–596.

Harris, C. & Stennett, C. (1990). Rapid-a video rate object tracker. In *BMVC*, 1–6.

Hartley, R. & Zisserman, A. (2003). *Multiple view geometry in computer vision*. Cambridge university press.

Hidayatullah, P. & Zuhdi, M. (2015). Color-texture based object tracking using hsv color space and local binary pattern. *International Journal on Electrical Engineering and Informatics*, 7(2):161.

Kehl, W., Tombari, F., Ilic, S., & Navab, N. (2017). Real-time 3d model tracking in color and depth on a single cpu core. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 745–753.

Koller, D., Daniilidis, K., & Nagel, H.-H. (1993). Model-based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer 11263on*, 10(3):257–281.

Lepetit, V., Fua, P., *et al.* (2005). Monocular model-based 3d tracking of rigid objects: A survey. *Foundations and Trends® in Computer Graphics and Vision*, 1(1):1–89.

Lepetit, V., Moreno-Noguer, F., & Fua, P. (2009). Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155.

Levenberg, K. (1944). A method for the solution of certain nonlinear problems. *Quart. Appl. Math.*, 2:164–168.

Lima, J. P., Teichrieb, V., Kelner, J., & Lindeman, R. W. (2009). Standalone edge-based markerless tracking of fully 3-dimensional objects for handheld augmented reality. In *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology*, 139–142.

Lima, J. P. S. d. M. (2014). Object detection and pose estimation from rectification of natural features using consumer rgb-d sensors.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, 2:1150–1157.

Morel, J.-M. & Yu, G. (2009). Asift: A new framework for fully affine invariant image comparison. *SIAM journal on imaging sciences*, 2(2):438–469.

Mur-Artal, R. & Tardós, J. D. (2017). Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262.

Phung, S. L., Bouzerdoum, A., & Chai, D. (2005). Skin segmentation using color pixel classification: analysis and comparison. *IEEE transactions on pattern analysis and machine intelligence*, 27(1):148–154.

Prisacariu, V. A. (2012). *Shape knowledge for segmentation and tracking*. PhD thesis, University of Oxford.

Prisacariu, V. A. & Reid, I. D. (2012). Pwp3d: Real-time segmentation and tracking of 3d objects. *International journal of computer vision*, 98(3):335–354.

Sáez, Á., Bergasa, L. M., López-Guillén, E., Romera, E., Tradacete, M., Gómez-Huélamo, C., & del Egido, J. (2019). Real-time semantic segmentation for fisheye urban driving images based on erfnet. *Sensors*, 19(3):503.

Saravanakumar, S., Vadivel, A., & Ahmed, C. S. (2011). Multiple object tracking using hsv color space. In *Proceedings of the 2011 International Conference on Communication, Computing & Security*, 247–252.

Seo, B.-K., Park, H., Park, J.-I., Hinterstoisser, S., & Ilic, S. (2013a). Optimal local searching for fast and robust textureless 3d object tracking in highly cluttered backgrounds. *IEEE transactions on visualization and computer graphics*, 20(1):99–110.

Seo, B.-K., Park, J., Park, H., & Park, J.-I. (2013b). Real-time visual tracking of less textured three-dimensional objects on mobile platforms. *Optical Engineering*, 51(12):127202.

Solanki, K., Madhow, U., Manjunath, B., Chandrasekaran, S., & El-Khalil, I. (2006). Print and scan'resilient data hiding in images. *IEEE Transactions on Information Forensics and Security*, 1(4):464–478.

Suzuki & Abe (1985). Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46.

Tan, D. J. & Ilic, S. (2014). Multi-forest tracker: A chameleon in tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1202–1209.

Tan, D. J., Navab, N., & Tombari, F. (2017). Looking beyond the simple scenarios: Combining learners and optimizers in 3d temporal tracking. *IEEE transactions on visualization and computer graphics*, 23(11):2399–2409.

Tjaden, H. (2019). *Robust Monocular Pose Estimation of Rigid 3D Objects in Real-Time*. PhD thesis.

Tjaden, H., Schwanecke, U., Schömer, E., & Cremers, D. (2018). A region-based gauss-newton approach to real-time monocular multiple object tracking. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1797–1812.

Tsai, R. (1987). A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4):323–344.

Wang, B., Zhong, F., & Qin, X. (2019). Robust edge-based 3d object tracking with direction-based pose validation. *Multimedia Tools and Applications*, 78(9):12307–12331.

Wang, C. & Qian, X. (2018). Heaviside projection–based aggregation in stress-constrained topology optimization. *International Journal for Numerical Methods in Engineering*, 115(7):849–871.

Wang, G., Wang, B., Zhong, F., Qin, X., & Chen, B. (2015). Global optimal searching for textureless 3d object tracking. *The Visual Computer*, 31(6-8):979–988.

Wu, P.-C., Lee, Y.-Y., Tseng, H.-Y., Ho, H.-I., Yang, M.-H., & Chien, S.-Y. (2017). [poster] a benchmark dataset for 6dof object pose tracking. In *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*, 186–191.

Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334.

Zhong, L. & Zhang, L. (2019). A robust monocular 3d object tracking method combining statistical and photometric constraints. *International Journal of Computer Vision*, 127(8):973–992.

A

DEPTH TESTING

Because the proposed approach is only interested in the outermost 2D silhouette edges at the last known pose, edge overlap is infrequent. Objects with rounded edges do not contain 2D silhouette edges overlapping. Objects with straight-angled edges (such as boxes, or the Chest from the OPT dataset) also avoid significant silhouette overlap due to perspective projection.

A simple CPU-based z-buffer was implemented for experiment on this. It had a big performance impact, but was completely functional. As expected, for rounded-edge objects it made little to no difference, and for the straight-angled ones in scenes where the edges were overlapping results were slightly better. Surprisingly, depth-testing made results much worse when dealing with rotations. A possible explanation is that by allowing some edge overlap, segmentation can work better for colors that are not directly visible, but that can appear with a small rotation. Thus, as not including depth testing made the work both faster and more precise, the choice has been made to not utilize it.

B

SATURATION THRESHOLD

HSV space by definition only has overlap in hue values when there is zero pigmentation present, in which case all hue values can lead to the same color (see Fig. 16). That is not true in the real world, though, where illumination, materials, camera sensors, and color-conversion algorithms play a role in the final color as perceived by a tracking program.

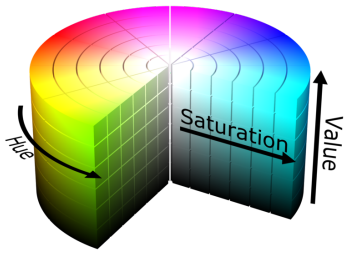


Figure 16: HSV color model in a classical cylindrical representation.*

*Image publicly available by CC BY-SA 3.0 (Creative Commons - Attribution) license, courtesy of Michael Horvath.

This concept can be better illustrated via a *print-scan* attack (Solanki *et al.*, 2006). Except, instead of a scanner, two cameras were used: a Logitech HD Pro Webcam C920 at 1920×1080 resolution and an Apple iPhone X's Rear-Facing Camera at 1920×1080 resolution. Full RGB images and isolated hue channel can be seen in Fig. 17.

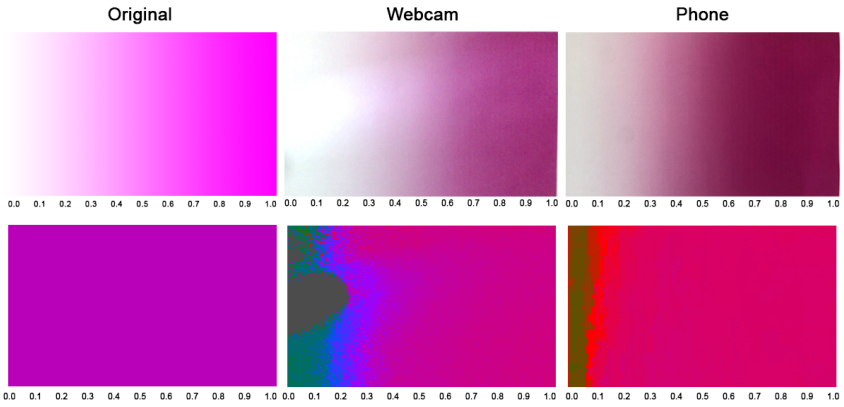


Figure 17: Experimentation for hue unreliability as saturation varies. Top row shows the full RGB images, bottom row the isolated hue channel.

C

IMPLEMENTATION TIPS

A few implementation tricks might severely increase performance and might be key to obtain real-time capabilities using the proposed technique. Thus, they are listed in the following paragraphs with replication easiness in mind.

It is recommended to always favor pre-allocation of values, such any matrix used repeatedly on the execution loop, when creating the tracker instance as opposed to doing it at every frame. Also favor inline functions and pass values, vectors, and matrices by reference whenever possible. This technique does not require much memory at all, thus, there is no reason to have concerns with pre-allocating where necessary in favor of performance.

The 2D search normals of ζ will always have x and y components ranging between 0 and 2 pixels, thus, this can also be preprocessed as opposed to calculating hundreds of square-roots at every iteration.

When rendering the wireframe using Bresenham's algorithm, especially for high-poly objects, most times edges will be no longer than 2 pixels, those being the vertices themselves. Thus, for such cases, the algorithm can be avoided.

Many elements of the pixel segmentation equations that relate to the preprocessed model can be computed when the tracker object is instantiated. This significantly improves performance as opposed to repeating thousands operations every frame. Moreover, hue values can be normalized to 0-180 as opposed to the traditional 0-360, enabling storage in 8-bit images and making data containers interchangeable with the RGB equivalents.

All masks ranging from r_0 to r_1 can also be preprocessed when creating the tracker object, having their centers at the canonical 2D origin $(0,0)$. These can then be added to every point in ζ as needed during runtime. This is much more efficient than repeatedly drawing and painting circles.

D

3DPO DATASET AND SETUP

The proposed qualitative dataset actually contains 2 quantitative sequences (with ground truth for every frame as opposed to only the first one): *Slow* and *Still*. Objects used in the videos were 3D printed in PLA after the dataset's 3D models, and material colors were set according to the filament. The exception being the Glass Bottle, which was modeled after the real thing (a standard Heineken beer bottle, easily obtainable). The setup can be seen in Fig. 18. Objects are illustrated in Fig. 11 and sequences in Fig. 19.

Figure 18: 3DPO's recording setup with the webcam on a tripod, as seen by an observer.

Sequences were captured using a standard RGB webcam with 640×480 resolution at 30 FPS. Sequences, CAD models, and camera calibration are publicly available.

$$f = \begin{bmatrix} 643.2870808241204, \\ 644.3426395729206 \end{bmatrix} \quad c = \begin{bmatrix} 317.1686355903425, \\ 219.3068634190016 \end{bmatrix} \quad d = \begin{bmatrix} 0.000909967262509, \\ 0.340232634571985, \\ -0.003160702756864, \\ -0.001101383931263, \\ -1.264632700808084 \end{bmatrix}$$

It is the author's hope that these sequences (albeit qualitative) can help with the literature gap concerning publicly available textureless real sequences for 6DoF object tracking.

Name: Backflips
Challenges: - fast motion
- motion blur
Length: 347 frames
Main Object: Green Bunny

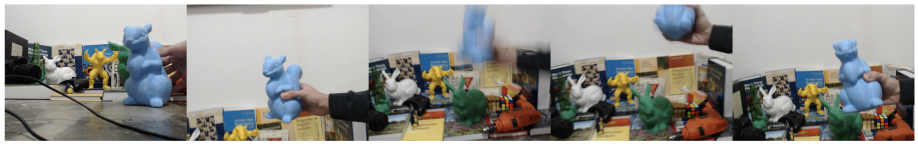
Name: Clipping
Challenges: - clipping
- motion blur
Length: 376 frames
Main Object: Red Cat

Name: Occlusion 1
Challenges: - occlusion
Length: 333 frames
Main Object: Red Cat

Name: Occlusion 2
Challenges: - occlusion
- handling
Length: 545 frames
Main Object: - Green Bunny
- Blue Squirrel

Name: Shaking 1
Challenges: - motion blur
- handling
Length: 184 frames
Main Object: Green Bunny

Name: Shaking 2
Challenges: - motion blur
- fast motion
- handling
- clipping
Length: 349 frames



Name: Independent Motion
Challenges: - simultaneous object and camera motion
- handling
- clipping
- motion blur
Length: 362 frames
Main Object: Blue Squirrel

Name: Multiple Objects
Challenges: - handling
Length: 560 frames
Main Object: all

Name: Refractive
Challenges: - handling
- occlusion
- refraction
Length: 411 frames
Main Object: Glass Bottle

Name: Slow
Challenges: - multiple viewpoint changes
Length: 1155 frames
Main Object: Green Bunny

Name: Still
Challenges: - none
Length: 1800 frames
Main Object: all

Figure 19: 3DPO sequences.

E

PUBLICLY AVAILABLE IMPLEMENTATIONS

As far as the literature review has shown, the best publicly available real-time RGB 6DoF trackers, which are commonly used for comparisons, are PWP3D (Prisacariu & Reid, 2012), GOS (Wang *et al.*, 2015), and RBOT (Tjaden *et al.*, 2018).

PWP3D is nearly 10 years old and has since inspired and been surpassed in precision by other works. RBOT, for instance, builds heavily upon PWP3D and was already extensively shown to surpass it in multiple papers (Tjaden, 2019). Thus, adding PWP3D to the comparison would not accurately represent the state of the art.

As for GOS, it is also fairly outdated, but none of the multiple newer techniques built on top of it have a publicly available implementation. Moreover, all newer works built on top of GOS are tested and compared to each other using the same textureless dataset, which would be very interesting to run the proposed work on. The problem is this dataset is not public and none of the authors responded when contacted. In the tests for this dissertation, GOS's results averaged below 35% on the RBOT dataset for all objects in every scene (with the reset metric), and not even 1% without a reset. GOS was also unable to recover from minor losses and was extremely sensitive to motion, which caused it to fail almost immediately in all 3DPO sequences when the objects or the camera were moved. Thus, it felt out of place in the state-of-the-art comparison.

Next, here is an explanation on why the results using the publicly available implementation of RBOT on the OPT dataset are lower than what is reported in RBOT's paper (Tjaden *et al.*, 2018). First, it is important to make clear that RBOT's paper's results for the RBOT dataset were fully reproduced for this work, and that the publicly available implementation of RBOT worked with no problems in the challenging 3DPO dataset as well. The AUC metric used for this work was confirmed to be correct by reproducing the OPT results of other works which used the publicly available implementation of GOS (Bugaev *et al.*, 2018). This AUC metric was also confirmed by RBOT's author when contacted. The author also provided the information that OPT objects needed some resampling to work with the publicly available implementation. Unfortunately, the authors did not have access to the modified objects anymore. For this work, the publicly available implementation could not be made to reach the numbers, so it was decided to use the best results that were obtained. There was no way around this issue because it was

necessary to run the publicly available implementation in order to build the AUC curve charts and scores per challenge for comparison.

Still, as a personal note, I want to make it clear that RBOT's publicly available implementation does work far better, overall, than any other 6DoF RGB tracker I have tested when reviewing the literature, both open source and from the industry. I would also like to point out that these lower OPT results are still comparable to the best publicly available real-time monocular 6DoF trackers. One last thing to mention is that the proposed work's superior numbers in the 3 OPT objects are also superior to the higher numbers in RBOT's original paper, so the points made in the previous chapters remain relevant.

To achieve the numbers reported from the publicly available implementation of RBOT, the object meshes Jet, Bike, and Chest from OPT (Soda, House, and Ironman worked without needing changes) and Cat, Squirrel, Ape, and Duck from RBOT dataset were simplified to 5000 vertices (as instructed by RBOT's paper) using MeshLab's[1] Quadric Edge Collapse Decimation with 0% reduction, 100% quality threshold, and set to preserve the boundaries, topology, and normals of the mesh. This simplification is necessary as RBOT's RAM consumption is otherwise unmanageable. Additionally, although RBOT is tuned to a 640×512 resolution (with parameters such as radius fixed for it according to the paper), it was found that the best results yielded for the publicly available implementation on OPT were obtained by running it at OPT's original 1920×1080 resolution. The code version came from the GitHub commit $7f14bce72430ba4feef213a63d1784ab93b0da96$, the most recent as of the writing of this dissertation.

[1] MeshLab