Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Tiril Sundby

# Towards Geometric Change Detection in Digital Twins using Dynamic Mode Decomposition, Object Detection and 3D Machine Learning

Master's thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
January 2021

NTNU
Norwegian University of Science and Technology

Tiril Sundby

# Towards Geometric Change Detection in Digital Twins using Dynamic Mode Decomposition, Object Detection and 3D Machine Learning

Master's thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
January 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

# Abstract

Digital twins are ment to bridge the gap between real-world physical systems and virtual representations. Both standalone and descriptive digital twins incorporate 3D models, which are the physical representations of objects building the digital replica. Digital twin applications are required to rapidly update internal parameters with the evolution of their physical assets. Due to an essential need for high-quality models for accurate physical representations, this causes the storage and bandwidth requirements for storing 3D model information to quickly exceed storage and bandwidth capacity.

In this work, we demonstrate a novel approach to geometric change detection in the context of a digital twin. We address the issue through a combined solution of Dynamic Mode Decomposition (DMD) for motion detection, YOLOv5 for object detection, and 3D machine learning for pose estimation. DMD is applied for background subtraction, enabling detection of moving foreground objects in real-time. The video frames containing detected motion are extracted and used as input to the change detection network. The object detection algorithm YOLOv5 is applied to extract the bounding boxes of detected objects in the video frames. Furthermore, the rotational pose of each object is estimated in a 3D pose estimation network. A series of convolutional neural networks (CNNs) conducts feature extraction for images and 3D model shapes. Then, the network outputs the estimated Euler angles of the camera orientation with respect to the object in the input image. By only storing data associated with a detected change in pose, we minimize necessary storage and bandwidth requirements while still being able to recreate the 3D scene on demand. To the best of our knowledge, a similar solution has not previously been attempted in a digital twin context.

# Sammendrag

En digital tvilling er en digital rekonstruksjon av et fysisk system i den virkelige verden. Flere typer digitale tvillinger bruker 3D-modeller som fysiske representasjoner av objekter i de digitale rekonstruksjonene. Digitale tvillinger krever jevnlige oppdateringer av interne parametre i henhold til utviklingen i verdier og ressurser. For å bygge digitale modeller som etterligner virkeligheten så nøyaktig som mulig må de fysiske modellene være av høy kvalitet. Dette fører til at mengden lagringsplass og båndbredde som kreves for lagring av informasjon om 3D-modeller raskt overstiger tilgjengelig kapasitet.

I dette arbeidet demonstrerer vi en ny tilnærming til deteksjon av geometriske endringer i sammenheng med digitale tvillinger. Tilnærmingen demonstrerer en innovativ løsning, der vi kombinerer bevegelsesdeteksjon gjennom Dynamic Mode Decomposition (DMD), objektdeteksjon ved bruk av YOLOv5, og 3D-maskinlæring til estimering av fysiske objekters posisjon og orientering. DMD henter ut videorammer der bevegelse blir detektert. YOLOv5 brukes så til å detektere objekter i videorammene. Videre estimeres posisjonen og orienteringen til 3D-objektene i videorammene gjennom et system basert på convolutional neural networks (CNN). Ved å fokusere på lagring av data direkte knyttet til detekterte endringer i fysiske objekter, minimerer vi kravene til nødvendig lagringplass og båndbredde, samtidig som vi fortsatt er i stand til å rekonstruere nødvendige 3D-scener ved behov. Metoden blir i dette prosjektet anvendt på eksperimenter med ekte data samlet ved hjelp av et eksperimentelt oppsett. Det har ikke blitt gjort funn av tidligere presenterte løsninger satt i kontekst til digitale tvillinger.

# Preface

This thesis is submitted as the final work of my Master's degree in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). It was carried out under the supervision of Professor Adil Rasheed, whom I want to thank for guiding and motivating me along the way, providing invaluable feedback and support.

I would like to thank Glenn Angel for being extremely helpful and efficient in designing and building the experimental set-up used in this project, and for answering all my requests. I would also like to extent my gratitude to Stefano Brevik Bertelli for helping out and facilitating this last semester.

Furthermore, I am grateful to the HPC group at NTNU for providing me with the necessary hardware to run my experiments. Finally, I would like to thank my friends in Trondheim for motivation and support throughout these five years.

Tiril Sundby

Trondheim, January 25th, 2020

# Table of Contents

# List of Tables

# List of Figures

# Abbreviations

| | | |
|---|---|---|
| 6DoF | = | Six Degrees of Freedom |
| AI | = | Artificial Intelligence |
| ANN | = | Artificial Neural Network |
| API | = | Application Programming Interface |
| CAD | = | Computer-Aided Design |
| CNN | = | Convolutional Neural Network |
| CPU | = | Central Processing Unit |
| CUDA | = | Compute Unified Device Architecture |
| CVAT | = | Computer Vision Annotation Tool |
| DL | = | Deep Learning |
| DMD | = | Dynamic Mode Decomposition |
| GPU | = | Graphics processing unit |
| HPC | = | High Performance Computing |
| IoT | = | Internet of Things |
| IoU | = | Intersect over Union |
| mAP | = | Mean Average Precision |
| ML | = | Machine Learning |
| MSE | = | Mean Squared Error |
| MLP | = | Multilayer Perceptron |
| PCA | = | Principal Component Analysis |
| ResNet | = | Residual Network |
| RoI | = | Region of Interest |
| SVD | = | Singular Value Decomposition |
| VMD | = | Video Motion Detection |
| YOLO | = | You Only Look Once |

Chapter 1

# Introduction

Digital twins are required to frequently update themselves according to the evolution of their physical assets. High-quality models are an essential factor in order to build digital replicas as similar to real world systems as possible. Given these requirements, 3D models used as input to physical simulators in digital twins can be enormous in size. Thus, the storage and bandwidth requirements for storing 3D models as a function of time will quickly exceed the storage and bandwidth capacity. In this thesis, we approach the issue through an innovative combination of motion detection, object detection and 3D pose estimation, to minimize the amount of collected information while still being able to recreate the 3D scene on demand.

## 1.1 Motivation and Background

Digital twins are one of the most intriguing prospects associated with the upcoming technology advancements of Industry 4.0 and the Internet of Things (IoT). This recent wave of digitalization has also affected the industry, where the need for viable approaches to real-time simulations are increasing. However, the complexity of many applicable processes results in high costs, storage incapacity, and computational and geometrical challenges.

A digital twin is defined as a virtual representation of a physical asset enabled through big data and simulators for real-time prediction, optimization, monitoring, controlling, and improved decision making [1]. While a digital twin operating as a digital sibling can be used for what-if analysis, risk assessment and uncertainty quantification, digital threads are used for transferring experience from one asset to the next iteration of assets [2, 3]. A digital twin's capability can be ranked on a scale from 0-5 (0 - standalone, 1 - descriptive, 2 - diagnostic, 3 - predictive, 4 - prescriptive, 5 - autonomy), as illustrated in Figure 1.1. Both standalone and descriptive digital twins may consist of 3D models, which are the physical representations of the objects building the digital twin [4].

3D models may be used as input to physical simulators in digital twins. High-quality models are an essential factor in order to build digital replicas as similar to real world systems as possible. 3D models based on these requirements can be enormous in size. Since digital twins are required to frequently update themselves according to the evolution of their physical assets, the storage and bandwidth requirements for storing 3D models as a function of time will quickly exceed the storage and bandwidth capacity [5, 6]. Extensive digital twins receive large amounts of real-time big data from sensor monitoring in its associated physical system. Data storage capacity has thus become one of the big challenges faced in the field [7]. To this end, this project proposes an innovative approach to geometric change detection in the context of a digital twin.

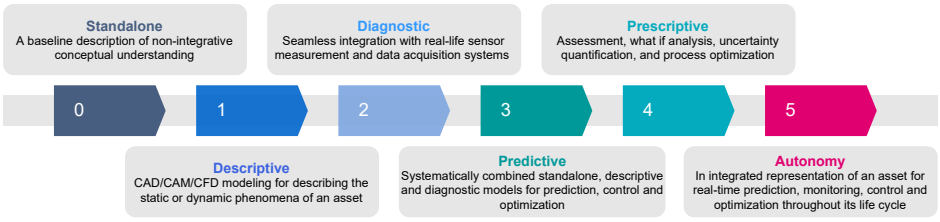



**Figure 1.1:** Digital twin capabilities, on a scale from 0-5

From a computer aspect, understanding the physical world involves accurately interpreting its surroundings. 3D understanding is important for various computer applications to operate in the real world [8, 9]. This is becoming increasingly important with the development of scientific fields such as autonomous vehicles, autonomous robotics, virtual reality, and augmented reality.

Detecting moving objects in video streams is a cornerstone objective in image processing and computer vision. Video surveillance, automation, and real-time monitoring are all applications that require real-time data processing. The demand for video processing has increased with the rising number of sensors used for monitoring, automation and surveillance technology applied in modern-day IoT applications. Several types of sensor data provide a foundation for performing motion detection. In this work, however, we focus on sensor data in the form of video streams.

In terms of digital twins, motion detection is a key element to detect changes in systems monitored through video surveillance. Background modeling is one of the methods applied for this purpose. Background modeling is a challenging task in practice, aiming to define models describing the nature of the background in video frames. This allows for extraction of moving foreground components for further processing and analysis, also in real-time [10]. Several applications presents viable results using the data-driven method Dynamic Mode Decomposition (DMD) for background modeling [11, 12]. Therefore, the method of DMD will be further investigated in this work.

In order for us to safely apply artificial intelligence (AI) methods to real-world systems, learning agents must understand, recognize, and interpret visual surroundings in three dimensions. While deep learning has contributed to significant improvements within 2D recognition, many applications for 3D data remain uncharted. There are several engineering challenges related to 3D machine learning, as operations on 3D data are much more complex than those of 2D data. However, this might change with the introduction of new tools for handling 3D data [13].

Recent advancements in AI and machine learning (ML) have opened up new possibilities for applying deep learning techniques to 3D format data. One of these applications is 3D pose estimation. Though pose estimation is considered to be an old computer vision problem, it is still a very relevant and active area of research [14, 15, 16]. This is applicable for predicting the behaviour of physical assets in digital twins.

Estimating 3D object structure from single RBG images is a challenging computer vision task. One of the reasons for this is the fairly recent introduction of large, properly annotated datasets applicable for training computer vision algorithms [17, 18, 19]. Several solutions have investigated the possibilities of performing 3D object detection and pose estimation from single images [20, 21, 22], which makes the methods much more applicable due to less input data requirements. Furthermore, estimating an object's continuous six degrees of freedom (6DoF) pose in terms of translation and rotation has been performed in some novel approaches for single RBG images [23, 24].

Today, many machine learning-based applications rely on algorithms trained to be instance-aware, expecting to be tested for the same object categories that they have been trained on. Earlier state-of-the-art deep pose estimation methods have been category-specific, simplifying estimation problem by assuming known input objects [25, 26]. Most of these methods require 3D CAD models as input, making them uapplicable to previously unseen objects [27, 28].

Recent contributions look at the possibility of estimating the 3D pose of objects from novel categories, i.e. objects not belonging to predefined categories used for training [29, 30]. Some of these methods argue that computer vision applications must be able to respond to previously unseen objects in real-world applications without requiring additional training or relying on 3D CAD models. Zhou et al. [31] and Grabner et al. [32] performed category-agnostic pose estimation on rigid objects with promising results, though they require similar data used for training and testing. However, in a digital twin application system operators will already know the object categories present at the site, thus the application may assume known objects without worrying about novel categories. Pose estimation applications can therefore be trained for category-specific data to ensure satisfactory system performance [33].

## 1.2 Objective

The main objective of this thesis is to combine the fields of 3D solid modeling and machine learning and explore the possibilities of using a novel approach of geometric change detection in the context of a digital twin, to minimize the amount of collected information while still being able to recreate a 3D scene on demand.

## 1.3 Research Questions

To the best of our knowledge, there is currently no published work on change detection in the context of a digital twin. To this end, the guiding questions governing the research are stated as:

- How can we analyse and validate 3D machine learning algorithms in the context of digital twins in a cost-effective manner?
- How can we create a workflow to detect rotational changes of solid models in three dimensions?

## 1.4 Outline of Report

In the following chapter, **Chapter 2**, we introduce relevant background material for this project. This will cover the theory behind motion detection, presenting in-depth the fundamentals of deep learning and 3D machine learning, including CNN's. In **Chapter 3**, we present relevant data, software and hardware framework and experimental set-up. Furthermore, we outline our methodology's specifics for change detection and describe the full workflow of the pipeline solution, including motion detection, 3D object detection, and 3D pose estimation. The final results and project insights are presented and discussed in **Chapter 4**. Finally, in **Chapter 5**, we conclude the project and discuss further work.

Chapter 2

# Theory

This chapter introduces relevant background topics for this project and establishes the theory required to have a thorough understanding of the methods presented in Chapter 3. In the first part of this section, an introduction to motion detection is given. Secondly, we give an in-depth explanation of object detection fundamentals, including a general overview of artificial neural networks (ANNs) and convolutional neural networks (CNNs). Then, we present methods and theory behind the topic of 3D pose estimation.

## 2.1 Motion Detection

Motion detection is the procedure of detecting a change in an object's position or orientation relative to its surroundings, or a change in surroundings relative to an object. Video motion detection (VMD) is the task of detecting motion in videos by analyzing differences in a sequence of video frames. For instance, a change in pixel intensity between consecutive video frames may be detected as motion.

The task of separating changes related to an object's movement from various noise factors, such as background noise, can be challenging. This challenge can be addressed by finding an applicable model describing the static background. By constructing a background model, object movement can be detected as a change in intensity compared to the background, given a certain threshold [34, 35], as illustrated in Figure 2.1.

By defining parts of a video frame as the background, we allow extraction of moving foreground objects that can be of potential interest, to be used as input in further analysis. This method is known as background subtraction or foreground detection, and it is an extensive field within image processing and computer vision.

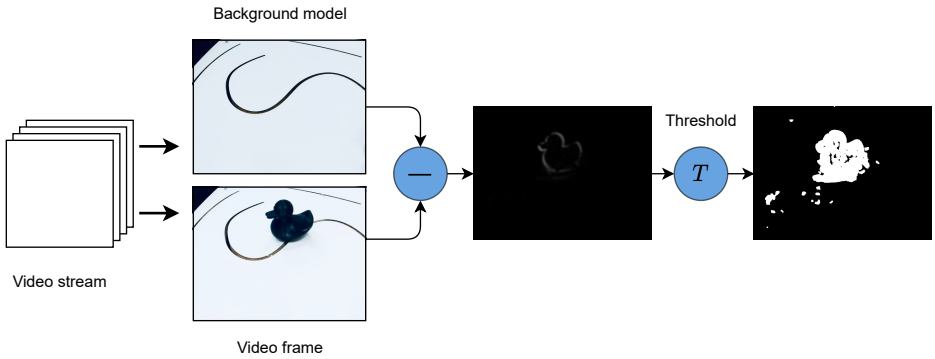



**Figure 2.1:** Illustration of background subtraction

### 2.1.1 Dynamic Mode Decomposition

Dynamic Mode Decomposition (DMD) is a data-driven method of matrix decomposition, originally introduced in the field of fluid mechanics [36]. DMD is capable of approximating nonlinear dynamics by providing reconstructions of coherent structures arising in dynamical systems [37]. It enables the evaluation of spatiotemporal structures, i.e., data collected in both space and time.

For a time series of data, the DMD computes a set of modes and eigenvalues, each with corresponding time dynamics defined in terms of a single eigenvalue [37]. These time dynamics are related to certain oscillation frequencies. DMD essentially performs background subtraction by computing DMD modes and differentiating between modes close to the origin and the remaining modes [10]. The DMD algorithm can be used as a diagnostic tool for system analysis and predicting future states. The combination of calculated modes and eigenvalues can produce a function approximating the system state at any given time.

DMD requires evenly spaced data sequences for computation, which applies well to the evenly distributed video frames in video streams. Consecutive video frames, referred to as snapshots, are flattened by vectorizing the pixel data in each snapshot. These vectors are then ordered in time as the column vectors of a matrix $\mathbf{D}$, as illustrated in Figure 2.2. Thus, the sequence of video frames is reshaped into a 2-dimensional spatiotemporal grid of size $\mathbb{R}^{nxm}$, where $m$ indicates the number of frames collected and $n$ indicates the number of pixels per frame. Each matrix element $x_{ts}$ is associated with a single pixel in both domains time $t$ and space $s$.

Furthermore, one assumes that two consecutive snapshots relate to each other with respect to time. This states the following expression, where the linear mapping $\mathbf{A}$ defines the relationship between consecutive snapshots. [11].

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t \tag{2.1}$$

When applying DMD to data generated by nonlinear dynamics, it assumes there exists an operator $\mathbf{A}$ that approximates the dynamics. The computed DMD modes and eigenvalues thus intend to approximate the eigenvalues and eigenvectors of $\mathbf{A}$. The eigenvalue decomposition of the linear operator portrays the underlying system dynamics of each snapshot.

To compute the DMD, the spatiotemporal grid obtained from the reshaped snapshots is first split into two overlapping sequences; left sequence $\mathbf{X}$ and right sequence $\mathbf{X}'$.

$$\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_{m-1}}], \quad \mathbf{X}' = [\mathbf{x_2}, \mathbf{x_2}, ..., \mathbf{x_m}] \tag{2.2}$$

The two sequences are related through the linear operator $\mathbf{A}$

$$\mathbf{X}' = \mathbf{A}\mathbf{X} \tag{2.3}$$

Then, we compute the singular value decomposition (SVD) of $\mathbf{X}$ as

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^* \tag{2.4}$$

Furthermore, we approximate $\mathbf{A}$ using the following least-squares operation

$$\hat{\mathbf{A}} = \min \|\mathbf{X}' - \mathbf{A}\mathbf{X}\|_F^2 \tag{2.5}$$

The least-square estimate is then computed in equation 2.6. $\mathbf{U} \in \mathbb{C}^{m \times n}$ and $\mathbf{V} \in \mathbb{C}^{n \times n}$ are matrices consisting of the left and right singular vectors of $\mathbf{X}$, respectively.

$$\hat{\mathbf{A}} = \mathbf{X}'\mathbf{V}\Sigma^{-1}\mathbf{U}^* \tag{2.6}$$

Furthermore, we can compute the eigenvalue decomposition of $\hat{\mathbf{A}}$, where $\mathbf{W}$ is the eigenvector matrix and $\Lambda$ is the corresponding diagonal matrix containing the eigenvalues, $\lambda$.

$$\hat{\mathbf{A}}\mathbf{W} = \Lambda\mathbf{W} \tag{2.7}$$

The dynamic DMD modes $\Phi$ corresponding to the eigenvalues $\lambda$ are then given by

$$\Phi = \mathbf{X}\mathbf{V}\Sigma^{-1}\mathbf{W} \tag{2.8}$$

Any snapshot at time $t$, including future snapshots, can be reconstructed using DMD as in equation 2.9, where $\lambda$ is the eigenvalue, $\phi$ is the dynamic mode, and $b$ is the related amplitude.

$$x_{DMD}(t) \approx \sum_{i=1}^{n} b_i \phi_i \lambda_i \tag{2.9}$$

As illustrated in Figure 2.2, the fully reconstructed video sequence using DMD then gives the following low-rank factorization of a given video stream

$$\mathbf{X}_{DMD} \approx \Phi\mathbf{B}V \tag{2.10}$$

where $\mathbf{B}$ is the diagonal matrix of amplitudes

$$\mathbf{B} = \begin{pmatrix} b_1 & & & \\ & b_2 & & \\ & & \ddots & \\ & & & b_k \end{pmatrix} \tag{2.11}$$

and matrix $V$ is defined as the Vandermode matrix [38] of the eigenvalues.

$$V = \begin{pmatrix} 1 & \lambda_1 & \cdots & \lambda_1^{n-1} \\ 1 & \lambda_2 & \cdots & \lambda_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_k & \cdots & \lambda_k^{n-1} \end{pmatrix} \tag{2.12}$$

Thus, the modes $\phi$ describe the spatial structure of the matrix decomposition, and the eigenvalues in matrix $V$ describes the temporal evolution of the DMD modes, where the matrix elements of $V$ are distinct frequencies defining the temporal dynamics.

In practice, we cannot directly apply equation 2.10 for background subtraction. Therefore, the solution is a matrix decomposition into a low-rank component, describing the background, and a sparse component, describing the foreground [39, 40]. First, the computed DMD eigenvalues are related to Fourier modes $\omega$ as in equation 2.13.

$$\omega_i = \frac{log\lambda_i}{\Delta t} \tag{2.13}$$

The Fourier modes provide some important insights. The real part of a Fourier mode describes mode evolution over time, while the imaginary part describes mode oscillations. The approximate low-rank DMD can now be rewritten, using a time-vector $\mathbf{t}$, as

$$\mathbf{X}_{DMD} = \sum_{i=1}^{k} b_i \phi_i e^{\omega_i \mathbf{t}} \tag{2.14}$$

Equation 2.14 proves that the Fourier modes describe how the modes change with respect to time. We can therefore separate our DMD modes in two categories; Fourier modes that change very slowly are related to the low-rank background video, $\mathbf{L}$, while fast moving Fourier modes are related to the sparse foreground video, $\mathbf{S}$. 2.14 is rewritten as

$$X_{DMD} = \mathbf{L} + \mathbf{S} \approx \underbrace{\sum_{i \in l} b_i \phi_i e^{\omega_i \mathbf{t}}}_{\text{background}} + \underbrace{\sum_{i \in s} b_i \phi_i e^{\omega_i \mathbf{t}}}_{\text{foreground}} \tag{2.15}$$

Foreground video can be defined as the difference between the original video and the background video. Discarding the imaginary values, the foreground video can therefore be calculated as

$$\mathbf{S} = \mathbf{X} - \mathbf{L} \tag{2.16}$$

This approach is applicable for implementations described in Section 3.5.1.

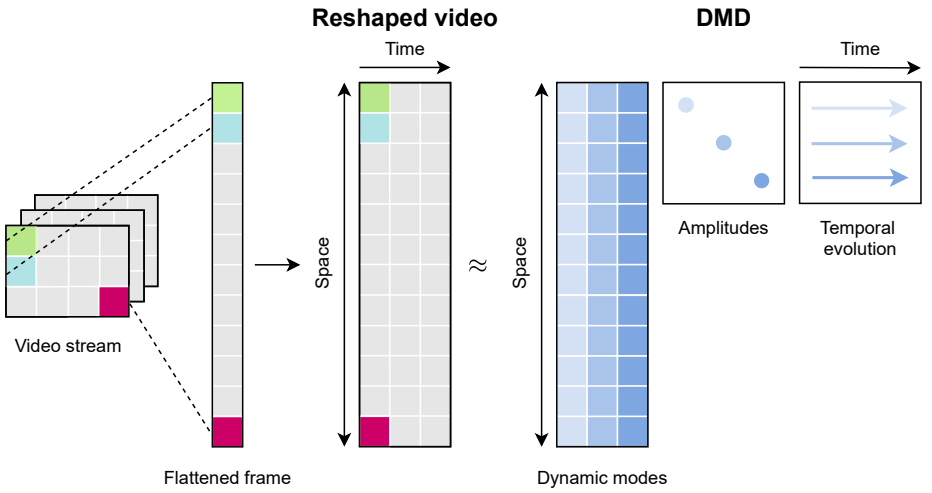



**Figure 2.2:** Illustration of DMD applied to a video stream. Initially, video frames are flattened into a 1-dimensional vector and ordered in time as the column vectors of a 2-dimensional spatiotemporal grid. DMD then constructs a decomposition in space and time; DMD modes containing the spatial structure, and the eigenvalues containing the temporal evolution.

## 2.2 Object Detection

Object detection is a computer vision task that combines object localization and object classification tasks. Thus, an object detection model aims to identify the presence of objects in input images and classify each identified object according to a set of defined classes. The detection of objects is marked by drawing a bounding box around each object. Object detection is considered a difficult task in computer vision, as both localization and classification have to yield accurate predictions for the network to output successful results. Object detection tasks are usually realized through approaches based on machine learning or deep learning. Both of these approaches will be further introduced in this section.

### 2.2.1 Machine Learning

Machine learning (ML) is the study of computational methods that use experiences to improve performance without being explicitly programmed [41]. Machine learning is an application of Artificial Intelligence (AI) that builds mathematical models based on sample data, allowing computers to learn from a series of examples [42]. Models are typically trained on parts of the sample data, the training data, before making predictions or decisions based on previously unseen test data.

Machine learning algorithms that uses labeled training data to infer a mapping function between pairs of inputs and outputs apply supervised learning. Labeled data is data that has already been classified. Thus supervised algorithms are designed to learn by ex-

ample. Supervised learning can be applied to both regression and classification tasks. In regression tasks, the aim is to approximate output values for a continuous set of values. In classification tasks, on the other hand, a set of discrete values defines the target value outputs. The discrete values are referred to as labels. In both regression and classification tasks, the learning procedure is the same; the learning algorithm generates an output $\hat{y}_i$ given an input $x_i$. The algorithm tries to minimize a loss function $L(y_i, \hat{y}_i)$ by iteratively updating its internal parameters according to the received feedback.

### 2.2.2 Artificial Neural Networks

An artificial neural network (ANN) is a computing system inspired by the human brain. ANNs are one of the most important tools in machine learning, and they provide the foundation for deep learning (DL) methods. ANNs can be used to approximate any given function [43], which makes them especially useful when applied to high complexity and dimensionality systems.

The fundamental building blocks of ANNs are a collection of interconnected processing elements called artificial neurons. These artificial neurons are built to resemble the biological neurons in a brain, typically modeled as in Figure 2.3. Real-valued signals are transmitted in between connected neurons. An artificial neuron receives several inputs $x_1, x_2, ..., x_n$ from its connected neurons and produces a single output. The neuron assigns a weight $w$ to each input $x$ and sums all the weighted inputs together with a bias term $b$. The weighted sum is then passed to an activation function $\varphi$, to estimate the output $y$.

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.17}$$

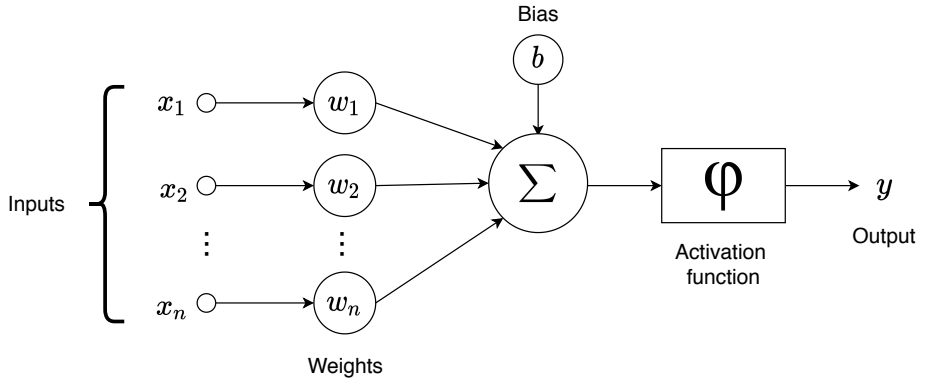



**Figure 2.3:** Simplified model of an artificial neuron

### Network Architecture

ANNs are built by arranging artificial neurons in connected layers. ANNs consists of one input layer, one output layer, and an arbitrary number of hidden layers. The neurons in the input layer represent features in received input data. The outputs of the input layer are connected to a hidden layer, not visible from the outside. The network architecture where the outputs of layer$_i$ are connected to the inputs of layer$_{i+1}$ are called feed-forward ANNs, or multilayer perceptrons (MLPs). Furthermore, networks with more than two hidden layers are called deep networks, applying deep learning for regression or classification tasks. Networks where all the outputs of one layer are connected to all the inputs of the next layer are called fully-connected layers, as illustrated in Figure 2.4. Implementations of an ANN is not a straight-forward procedure. The main challenge is usually to decide the optimal network size configuration. Whereas the size of the input and output layers are fixed, the size and numbers of hidden layers must be chosen based on the complexity of the input data and the effect on system performance.

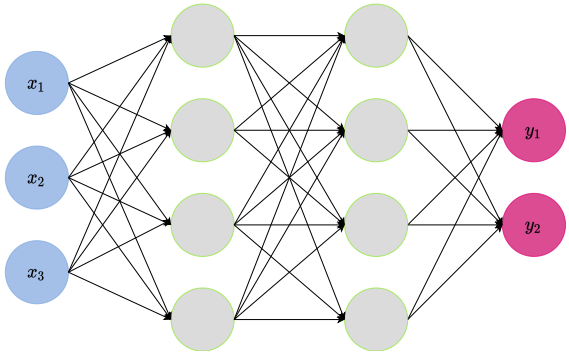



**Figure 2.4:** ANN architecture with two hidden layers

### Activation Function

Activation functions are nonlinear mathematical functions attached to each neuron in a neural network. They are essential parameters in deep learning, as they determine the output, accuracy, and efficiency of a network. Given a large enough input, the activation function fires, and the neuron is activated. This required input size is determined by a defined threshold. Nonlinear activation functions allow more complex mappings between inputs and outputs, necessary for modeling complex data. Three of the most commonly used nonlinear activation functions are Sigmoid, Hyperbolic Tangent (tanh), and Rectified Linear Unit (ReLU), respectively defined as

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.18}$$

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.19}$$

$$ReLU(x) = max(0, x) \tag{2.20}$$

#### Optimization and Backpropagation

ANNs create a mapping between input and output data by learning to recognize patterns in the data. This learning process is known as the training phase, where network parameters are altered to generate accurate network predictions.

A loss function is a measure of error used to quantify how well an ANN can approximate a target function. The loss function provides feedback to the network on how well the target function is approximated. A commonly used loss function is Mean Squared Error (MSE). MSE calculates the error as the difference between the predicted output value $\hat{y}$ and the target value $y$.

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.21}$$

As the loss is the penalty for prediction errors, perfect output predictions will result in zero loss. Thus, training poses an optimization problem aiming to minimize the loss function $L$. The most common method is approximating the minimum of the loss function using gradient descent. Gradient descent is an optimization algorithm that tries to minimize the loss function by iteratively updating model parameters in the direction opposite the gradient of the loss function. The optimizer takes iterative steps to reach a local minima in the following way:

$$\phi_{i+1} = \phi_i - \alpha \nabla L(\phi_i) \tag{2.22}$$

Here, $\phi$ represents the optimized network parameters, and $\alpha$ is the defined learning rate. The learning rate is a hyper-parameter determining the size of each iteration step. It is used for tuning the network during training, where it controls how much network weights are adjusted with respect to the loss function gradient at each step. However, directly applying this method to an ANN comes at a high computational cost. A more efficient, commonly used algorithm for training feed-forward ANNs is backpropagation. Backpropagation computes the loss function gradient with respect to each weight in a network, according to the chain rule. The algorithm iteratively propagates the loss backward in the network, traversing from the output layer to the input layer, computing the gradient one layer at a time [44]. Thus, the prediction error is iteratively corrected by adjusting internal weights until the error between prediction and target values is minimized. Weights are adjusted relative to the amount they contribute to the error using gradient descent.

### 2.2.3 Convolutional Neural Network

Convolutional neural networks (CNNs) are a specialized class of deep neural networks designed to process tensor data. CNNs are commonly used for image processing tasks and have generated exceptional results in image pattern recognition tasks [45]. As a result of its recent success, CNNs are considered to be the leading method for detection and recognition applications [46].

As deep neural networks, CNNs build on the idea of how distinct features are built from collections of smaller, low-level features. Collectively, these low-level features form local clusters that eventually represent parts of an object. Images, text, and speech are all built based on similar hierarchies, making them all applicable for pattern recognition using CNNs [46].

### Convolutional Layer

A CNN is essentially a form of ANN containing convolutional layers. Convolution is a linear mathematical operation on two functions, $f$ and $g$, where the convolution $f * g$ discloses how much one function is shifted over the other, eventually fusing the two functions. The standard mathematical expression for convolution applied to neural networks is defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau \tag{2.23}$$

In a more realistic scenario however, we may expect discretized input data and therefore rewrite equation 2.23 as a discrete convolution in the following way:

$$(f * g)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(t - \tau) \tag{2.24}$$

This is illustrated in Figure 2.5. In neural network applications, the input argument $f$ is referred to as the input and the argument $g$ is referred to as the kernel. The convoluted output $(f * g)(t)$ is also referred to as the feature map [47]. Thus, the convolutional layer inputs a tensor and abstracts it to a feature map, with the aim of learning features.

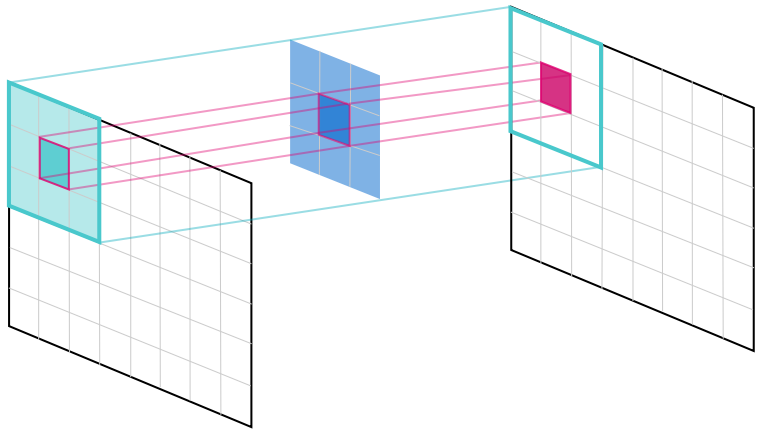


**Figure 2.5:** Matrix convolution

In our applications of CNNs, we will work with image inputs, represented as 2D arrays. We may therefore introduce convolution over two axes, using a 2D image $I$ as

input, and thereby defining a 2D kernel $K$ :

$$(\mathbf{I} * \mathbf{K})(i,j) = \sum_m \sum_n \mathbf{I}(m,n)\mathbf{K}(i-m, j-n) \tag{2.25}$$

The discrete convolution operation can now be viewed as a form of matrix multiplication, where the kernel is represented as a matrix carrying out the convolutional operation. The convolution aims to extract high-level features, such as edges, in input images. The kernel shifts its position and performs a matrix multiplication between the kernel matrix, K, and a given portion of the input image, as illustrated in Figure 2.6. The stride value parameter decides the length the filter moves at each iteration. The filter moves around until the entire image has been traversed.

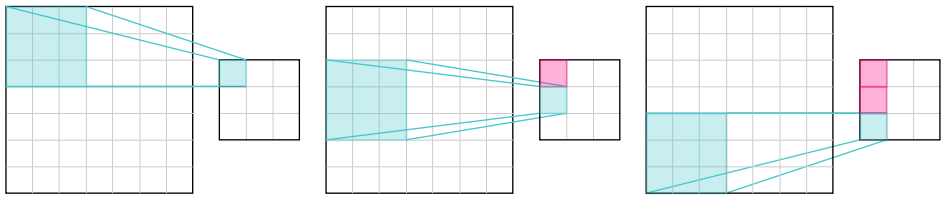


**Figure 2.6:** Convolution operation and shifting kernel

### Pooling Layer

CNN architecture is typically composed of a series of stages, where the convolutional layer and pooling layer populate the initial stages. The outputs from a convolutional layer are usually sent through a ReLU nonlinear activation function before being passed on to a pooling layer. Pooling layers are normally applied after convolutional layers to reduce the spatial size of the feature map output from the convolutional layer.

The convolutional layer detects feature conjunctions in previous layers, and the pooling layer fuses similar features into one [46]. The purpose is to decrease the required computational power used for processing the data. This is done through dimensionality reduction. Several convolutions, nonlinear activation functions, and pooling layers are typically stacked together before fully-connected layers are added to a CNN. Furthermore, training through backpropagation is performed in the same manner as for regular ANNs.

### Residual Network

Deep neural networks are hard to train because they are prone to an issue referred to as vanishing gradient. With increasing numbers of layers, the repeated multiplications performed by the backpropagation algorithm eventually results in a very small gradient, and thus a declining system performance. This leads to the introduction of residual networks (ResNet) [48]. Residual networks address this challenge by reformulating network structure, applying shortcuts, or skip connections that skips some layers in the network.

These skip connections are typically implemented to skip two or three layers at a time. This simple step significantly simplifies training of deep learning networks, making them much easier to optimize and eventually avoid the problem of vanishing gradient [49].

### 2.2.4 YOLO - You Only Look Once

You only look once (YOLO) is a family of state-of-the-art object detection applications capable of effectively processing images in real-time. YOLO was first introduced by Redmon et al. [50] in 2016, providing one of the fastest object detection algorithms at the time. Since then, several versions of the YOLO application have been released. Most YOLO iterations have been implemented and maintained in the open-source Darknet framework [51]. In 2020, however, a PyTorch-based framework called YOLOv5 was released, outperforming all previous versions.

The YOLO algorithm varies from its predecessors in detection systems by applying a single neural network to an image. With this application, the detection network is provided with the full context of an input image at test time, which is advantageous compared to former solutions based on classifiers. The name ”You Only Look Once” relates to the fact that YOLO only performs a single network prediction, thus only extracting information one time per input image. This results in YOLO achieving much faster predictions than networks depending on multiple predictions per input image. The network first divides each input image into grid regions before predicting bounding box estimates and confidence scores for each region. The confidence score reflects how certain the network is of its predicted bounding boxes. YOLO predicts bounding boxes based on two image coordinates, bounding box width and bounding box height, per detection.

#### Evaluation Metric

The universal metric used for comparing and evaluating the performance of detection networks is mean Average Precision (mAP). The metric is calculated as the mean value of the average precision calculated separately for each class of objects in an image dataset. Furthermore, two performance measures commonly used in classification tasks are precision and recall. Precision is defined as the proportion of relevant instances in a set of retrieved instances. Precision thus aims to find the proportion of correct classifications among all classifications. Recall aims to find the proportion of actual identifications that are correctly classified. Precision and recall are computed according to equations 2.26 and 2.27, respectively. Any single prediction can be one of four definitions presented in Figure 2.7, relative to ground truth labeled data. We can compute a precision-recall curve for each class in our dataset by plotting precision against recall for each classifier.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.26}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.27}$$

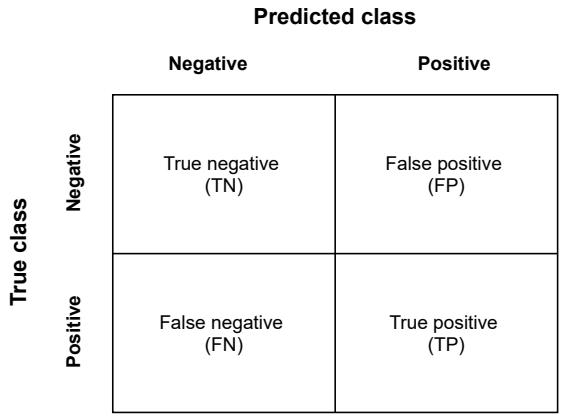



**Figure 2.7:** Confusion matrix for binary classification

Computations of prediction and recall are based on a measure defining how correct a prediction really is. This prediction correctness is be evaluated using a measure called Intersect over Union (IoU). IoU measures the overlap of two associated bounding boxes in object detection tasks. These refer to the predicted bounding box and the ground truth bounding box defining the boundaries of the real object. A given threshold defines the amount of overlap necessary for a prediction to be considered correct. IoU is computed according to equation 2.28, as the area of overlap divided by the area of union between the predicted bounding box, $BB_{pred}$, and the ground truth bounding box, $BB_{gt}$.

$$\text{IoU} = \frac{\text{Area}(B_{pred} \cap BB_{gt})}{\text{Area}(BB_{pred} \cup BB_{gt}} \tag{2.28}$$

## 2.3 3D Pose Estimation

3D machine learning is a field integrating machine learning, computer vision, and computer graphics to enhance 3D understanding. Recent years have provided significant progress in performing object detection, and segmentation in images [52, 53], perceiving 3D attributes is an essential factor in many real-world applications. 3D predictions such as shape and pose are important applications in robotics, autonomous vehicles, and visual and augmented reality, among others, where 3D perception is a key factor. Great strides have been made over the last couple of years to develop technology and deep learning methods applied to 3D data. Developed methods are applied to classification and semantic segmentation of 3D shapes and scenes, synthesis and reconstruction of 3D geometry, and 3D pose estimation. We will further focus on explaining 3D pose estimation, as this is relevant for this work.

Estimating the 3D pose of objects in 2D images is an essential task in applications related to 3D perception [54]. It is a challenging but fundamental computer vision problem, highly relevant in modern-day applications. 3D pose estimation is the task of predicting

the transformation of an object with respect to a defined reference pose, matching the spatial position of the object. Pose estimation can be used for identification, object manipulation, or Computer-Aided Design (CAD) model alignment, which are typical tasks emerging from the field of robotics-related computer vision. Essentially, recovering a full object pose for such applications requires high accuracy object detection of known 3D CAD models [55].

Generally, we may recover the 3D pose of an object either by localizing a set of keypoints describing the object shape or by estimating the object's viewpoint [56]. Recently published work apply many different methods to recover the 3D pose of objects, or humans, based on RBG or RBG-D images. Feature-matching methods are typically applied to RBG images, aiming to extract features of 3D objects and further recover a full 6DoF pose by either correctly matching the features against known objects in a feature database [57], or by matching coherent feature keypoints between estimating 2D images and 3D objects [58]. Other methods match 2D-3D coherence based on 2D predictions of the estimated 3D bounding boxes of objects [59]. However, these methods rely heavily on using textured input objects to extract shape features.

### 2.3.1 Object Orientation

Object orientation describes the placement of a rigid body and the imaginary rotation needed to move the object from the placement of reference to its placement at present. A change in placement may require both translation and rotation. When we talk about moving an object in the 3-dimensional space, we often talk about an object's six degrees of freedom (6DoF). 6DoF refers to a change in position in terms of translation on three perpendicular axes, combined with orientation changes in terms of rotation about three perpendicular axes.

Euler angles are often used to describe an object's rotation with respect to a fixed reference frame. Another orientation method most commonly used in astronomy is azimuth-elevation orientation. In Figure 2.8, azimuth is defined as the angle $\phi$ and elevation as the angle $\theta$. The $r$ parameter defines the distance from a specific point or object to the viewpoint. The point is defined in the 3-dimensional space by spherical coordinates. The Cartesian coordinates equivalent to the point's spherical coordinates are derived in equations 2.29-2.31.

$$\mathbf{X} = r\cos(\theta)\sin(\phi) \tag{2.29}$$

$$\mathbf{Y} = r\cos(\theta)\sin(\phi) \tag{2.30}$$

$$\mathbf{Z} = r\sin(\theta) \tag{2.31}$$

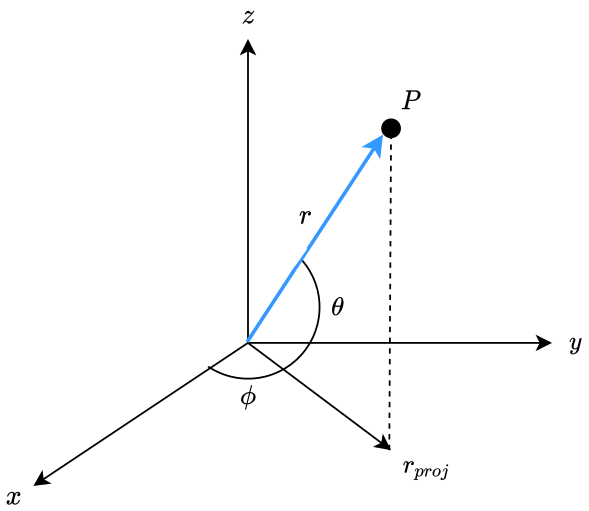



**Figure 2.8:** Illustration of spherical coordinates

An elemental rotation defines a rotation about one of the axes in a coordinate system. A rotation matrix is a transformation matrix used to perform rotations of an object in 3-dimensional space. A 3D rotation matrix describes three successive rotations about arbitrary coordinate axes. Equations 2.32-2.34 define elemental rotations of $\phi$, $\theta$ and $\psi$ degrees about the $x-$, $y-$ and $z-$ axis, respectively. A full object rotation about all three axes is defined by multiplying consecutive single rotations about an individual axis, resulting in a rotation matrix $\mathbf{R}$.

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s\phi & c\phi \end{bmatrix} \tag{2.32}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix} \tag{2.33}$$

$$\mathbf{R}_z(\psi) = \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.34}$$

Chapter 3

# Method and Set-up

In this chapter, the physical and computational set-up of our project is presented. First, we outline the software and hardware frameworks. Secondly, we describe the experimental set-up and datasets used in our experiments. Then, we present the methods applied for motion detection, object recognition, and 3D pose estimation. Finally, we present the full workflow of our approach.

## 3.1 Software Framework

Both the object detection and 3D pose estimation architectures used in this work were implemented in Python 3.6 using the open-source machine learning library PyTorch [60]. The pose estimation network implements Blender 2.77, an open-source library implemented as a Python module to render multi-views of 3D CAD model inputs. Furthermore, the MeshLab software is used to visualize 3D objects, and the Python library Matplotlib is used for creating most plots and diagrams.

The DMD motions detection algorithms were implemented in Python, enabling real-time processing through OpenCV (Open Source Computer Vision Library) [61]. OpenCV is an open-source software library for computer vision and machine learning. It provides an infrastructure for computer vision tasks and optimizes machine perception methods.

## 3.2 Hardware Implementation

Deep learning algorithms require large amounts of memory as well as computing power. Computing power usually requires a proper graphics processing unit (GPU), allowing efficient manipulation of computer graphics and image processing. When working with neural networks, running code on CPUs is not a viable solution. GPUs drastically reduces training time in many deep learning tasks. Today, many of the commonly used machine learning frameworks are built upon CUDA-enabled GPUs. CUDA (Compute

Unified Device Architecture) is a parallel computing platform and API used for general purpose processing. These CUDA-enabled GPUs are also referred to as GPGPUs.

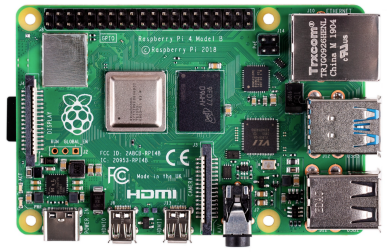
The pose estimation architecture in this project was trained and tested on the NTNU Idun computing cluster [62] to provide necessary computing power. The Idun cluster is a cooperation between the IT division and various faculties at NTNU, aiming to provide a computing platform enabling rapid testing and prototyping of HPC software. The Idun cluster has more than 70 nodes and 90 GPGPUs. Each node contains two Intel Xeon cores and a minimum 128 GB of main memory. All the nodes are connected to an Infiniband network. Half of the nodes are equipped with two or more Nvidia Tesla P100 or V100 GPUs. Idun's storage is provided by two storage arrays and a Lustre parallel distributed file system.

The object detection network applied in this work was trained and tested using Google Colab. Google Colab is an open-source computing platform developed by Google Research. The platform is a supervised Jupyter notebook service where all processing is performed in the cloud while providing free access to GPU computing resources. Google Colab provides easy implementations and compatibility between different machine learning frameworks.

## 3.3 Experimental Set-Up

To demonstrate our approach to geometric change detection, we constructed an experimental set-up based on a Raspberry Pi 4 kit [63] and a connected Raspberry Pi camera. Raspberry Pi 4 is the fourth generation single-board computer developed by the Raspberry Pi Foundation in the United Kingdom. It is widely used within a range of fields. The Raspberry Pi is designed to run GNU/Linux, and it features a processing unit powerful enough to run various image processing algorithms, as utilized in this work. The hardware specifications of the Raspberry Pi 4 are presented in the table in Figure 3.1b. The Raspberry Pi 4 was connected to a Raspberry Pi HQ camera, version 1.0, and a 6mm compatible lens during experimentation. This is a high-quality camera offering high image resolution, see the table in Figure 3.2b.

We built a camera tripod from 3D printed parts and two steel arms. A square platform with laser-cut trenches was designed and set up to mount the camera tripod. A torch was connected to one of the steel arms, serving as an external light source. The two steel arms were set up to move independently of each other in order to record videos with external lighting applied from different angles. The full set-up is displayed in Figures 3.3 and 3.4. While running the real-time DMD algorithm, the Raspberry Pi was connected to an external monitor displaying the scene captured by the Raspberry Pi camera and associated video processing results. A set of scripts were implemented on the Raspberry Pi 4, containing DMD and 3D pose estimation algorithms. This part is further explained in Section 3.5.1. Furthermore, the Raspberry Pi was connected using an Ethernet cable to ensure a stable internet connection and a designated Raspberry Pi power source.



**(a)**

| | |
|---|---|
| Processor | Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz |
| Memory | 8GB LPDDR4 |
| GPU | Broadcom VideoCore VI @ 500 MHz |
| OS | Raspbian Pi OS |

**(b)**

**Figure 3.1:** Illustration and technical specifications of the Raspberry Pi 4 Model B

**(a)**

| | |
|---|---|
| Sensor | Sony IMX477 |
| Sensor Resolution | 4056 x 3040 pixels |
| Sensor Image Area | 6.287mm x 4.712mm |
| Pixel Size | 1.55$\mu$ x 1.55$\mu$ |
| Focal Length | 6mm |
| Resolution | 3 MegaPixel |

**(b)**

**Figure 3.2:** Illustration and technical specifications of the Raspberry Pi HQ V1.0 camera and the 6mm IR CCTV lens

**Figure 3.3:** Overview of the experimental set-up

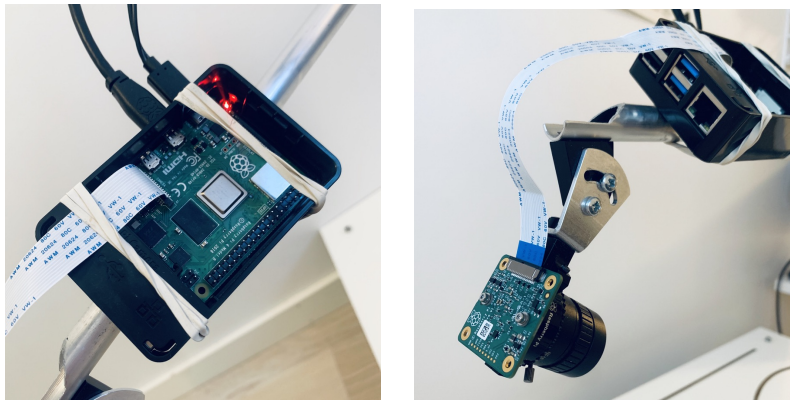
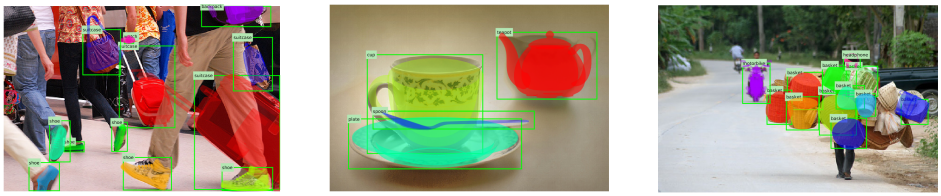


**Figure 3.4:** The Raspberry Pi 4B and Raspberry Pi camera mounted on the camera tripod

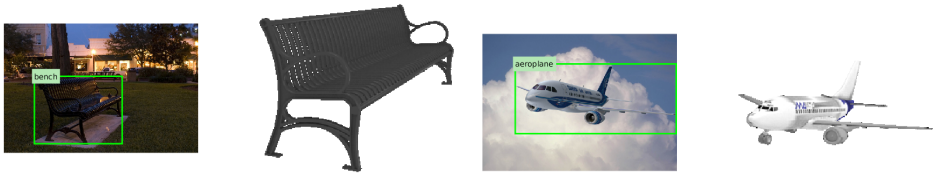
## 3.4 Datasets

The algorithms used in this work were trained on two large-scale datasets. We also created one annotated dataset of our own. The ObjectNet3D [17] and Pascal3D [18] datasets are used for training the 3D pose estimation network. These datasets are built to provide 3D pose and shape annotations for various detection and classification tasks. The ObjectNet3D database consists of 100 object categories, 44,147 object shapes, and 90,127 images with 201,888 objects, where objects are aligned with 3D shapes providing accurate 3D pose annotation and 3D shape annotation for each 2D object, as illustrated in Figure 3.5. Similarly, the Pascal3D database consists of 12 image categories with more

than 3,000 instances per category.



**(a)** 3D pose annotation examples




**(b)** 3D shape annotation examples

**Figure 3.5:** Examples of 3D pose and shape annotation results from ObjectNet3D

A set of 3D CAD models from various random categories were used for object detection and pose estimation in this work. The model files were downloaded from GrabCAD and Free3D. We used STL files for 3D printing and associated OBJ files as input to our pose estimation network.

**Figure 3.6:** 3D CAD models used for 3D printing and as inputs to the 3D pose estimation netowork, displayed in MeshLab.

Furthermore, data used for testing both the object detection algorithm and the pose estimation network came from a set of collected images and sampled video frames from the Raspberry Pi. A dataset of 193 images of 3D printed objects from the nine object categories presented in Figure 3.6 was collected using the experimental set-up described in 3.3. The collected dataset contains images of

- Single objects seen from different angles
- Multiple objects seen from different angles
- Objects occluded by other objects, meaning objects where some are in focus and some are blurred in the background

In addition, we took images both in natural lighting and with additional lighting from four different angles using the torch attached to the camera tripod. Preprocessing and image augmentation was applied to the images before exportation. Three augmented images were created for each image in the dataset, in order to provide the network with a sufficiently large dataset for training. Eventually, our final image dataset consisted of 463 images. The image processing is further described in Section 3.5.2.

## 3.5 Geometric Change Detection

The overall approach in this work is presented in Figure 3.7. We consider a cubical room consisting of 3D objects, each capable of moving with six degrees of freedom. 3D CAD models of these objects are saved at time $t = 0$. A single RGB camera is pointed towards the collection of objects. When the scene is stationary, the camera does not record anything. However, as soon as objects start to move, the motion is detected using a motion detection algorithm based on Dynamic Mode Decomposition (DMD). The whole sequence of motion ($t = t_1 - t_2$) is then recorded. When the scene becomes stationary again, the whole sequence is deleted after saving the last video frame containing detected motion.

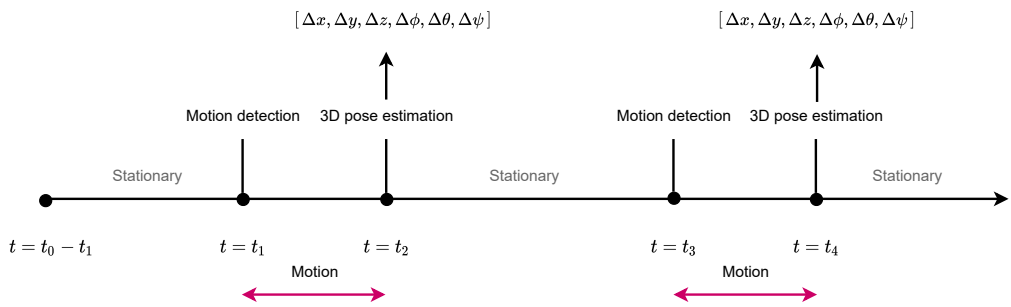



**Figure 3.7:** Workflow of our approach geometric change detection

Furthermore, the last frame is analyzed using YOLOv5 and a 3D pose estimation algorithm to estimate the applied effects of translation and rotation, after which only changes in the six variables ($\Delta x, \Delta y, \Delta z, \Delta\phi, \Delta\theta, \Delta\psi$), corresponding to the changes in degrees of freedom, are saved. In the following section, we give a brief overview of the applied methods of motion detection, object detection, and 3D pose estimation utilized in this work. We also present the details of our set-up that has been used to mimic physical assets.

### 3.5.1 DMD Implementation

We implemented a DMD methods in Python for testing the application of motion detection on our experimental set-up. The method was implemented according to Algorithm 1, based on coding examples from the book Kutz et al. [37]. The Python implementation enables real-time motion detection using OpenCV on a system webcam, in our case, the Raspberry Pi HQ camera. For testing purposes, most of the DMD processing was performed using prerecorded videos and images. Some processing was also run on the Idun cluster for additional computational power.

Several configurations of the DMD algorithm were tested. Figure 3.7 presents the full workflow of our approach, where DMD is the initial catalyst. The DMD implementation follows the steps presented in Section 2.1.1, where the computed background is subtracted from the original video, and moving objects in the video foreground are outputted.

**Algorithm 1:** Dynamic Mode Decomposition (DMD)

**Require:** Input matrix $\mathbf{D} \in \mathbb{R}^{m \times n}$, target rank $k$
**procedure** DMD($\mathbf{D}, k$)
$\mathbf{D} \leftarrow x(t_0), ..., x(t_m)$
$\mathbf{X}, \mathbf{X}' = \mathbf{D}$
$\text{SVD}(\mathbf{X}, k) = \mathbf{U}, \Sigma, \mathbf{V}$
$\tilde{\mathbf{A}} = \mathbf{U}^* * \mathbf{X} * \mathbf{V} * \Sigma^{-1}$
$\mathbf{W}, \lambda = \text{eig}(\tilde{\mathbf{A}})$
$\Phi^{DMD} \leftarrow \mathbf{X}' * \mathbf{V}\mathbf{S}^{-1} * \mathbf{W}$
$\mathbf{b} = lstsq(\Phi, \mathbf{x}_1)$
$V = Vandermonde(\Lambda)$

### 3.5.2 Data Annotation and Training of YOLOv5

For detecting objects in images, we chose the YOLOv5 framework, which is the latest version in the family of YOLO frameworks at the time of writing. YOLOv5 is the first YOLO implementation written in the PyTorch framework, and it is therefore considered to be more lightweight than previous versions while at the same time offering great computational speed. There are no considerable architectural changes in YOLOv5 compared to the previous versions YOLOv3 and YOLOv4. Performance of different YOLO versions is illustrated in Figure 3.8, from the Github repository of YOLOv5 [64]. There are yet no published articles on YOLOv5 at the time of writing.

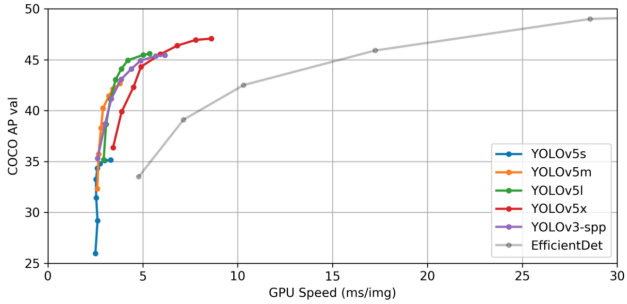
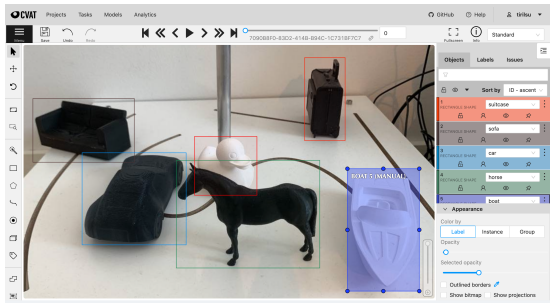



**Figure 3.8:** Performance graph for different versions of YOLO tested on the COCO AP dataset

YOLOv5 was trained to recognize the 3D CAD models from our experimental set-up. To train our object detector through supervised learning, we required a properly annotated dataset. The dataset was collected using the experimental set-up as specified in 3.4. We used CVAT (Computer Vision Annotation Tool) for labeling images. CVAT is an open-source, web-based image annotation tool produced by Intel. To annotate the images, we drew bounding box around the objects that we wanted our detector to localize, and assigned the object categories that we wanted out detector to classify.



**(a)** Screenshot of the image annotation process in CVAT

**(b)** Screenshot of the fully annotated dataset in Roboflow

**Figure 3.9:** Procedure of annotating dataset used for supervised learning in image annotation tools CVAT and Roboflow

Furthermore, the fully annotated dataset was opened in Roboflow where techniques of preprocessing and augmentation were applied. Data augmentation involves altering training images to extend the original dataset with a synthetic dataset to provide more training data. Data augmentation aims to improve model performance during training. Several augmentation techniques, such as rotation, crop, gray-scale, exposure, and noise measures, were applied. The final image dataset was split into three parts, in a 70-30-10 pattern. Approximately 70% of the images were applied to the training set, 20% to the validation set, and the remaining 10% was applied to the test set. The procedures from CVAT and Roboflow are illustrated in Figure 3.9.

The YOLOv5 network was implemented in Google Colab using PyTorch. We chose the biggest YOLOv5 model available for training the network, namely YOLOv5x (x-large). The network was trained using the Adam optimizer and the training parameters presented in Table 3.1.

| Parameter | Value |
|---|---|
| Image size | 640 |
| Batch size | 16 |
| Epochs | 300 |
| Device | GPU |
| Learning rate | $10^{-2}/10^{-3}$ |

**Table 3.1:** Choice of training parameters for YOLOv5x

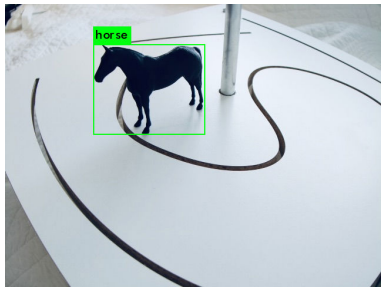
### 3.5.3 Pose Estimation: Translation

The initial plan for this work was to calculate the 6DoF pose of the objects in our images. The idea was to estimate object translation in Cartesian coordinates using a combination of bounding box coordinates for x and y from the YOLOv5 network and triangle similarity. Given an object with a known width, placed a given distance from our camera, we can measure the elusive pixel width in an image and derive the perceived focal length between our object and the camera. By configuring our camera with a known distance, it will then be able to estimate other distances in future experiments.

However, our current experimental set-up does not allow us to determine the camera's orientation and viewpoint. Without this knowledge, we cannot know the objects' center of mass in our detected bounding boxes. Therefore, calculations based on measured object size and triangle similarities become inaccurate. With some alterations and additions to our experimental set-up, such as adding a depth-camera, determining the camera viewpoint would not be an issue.

The lack of necessary apparatus in our set-up results from inadequate testing in the design process. Construction of the experimental set-up was disrupted due to circumstances concerning the ongoing COVID-19 pandemic. Necessary equipment became unavailable

to the author, and there was little time to properly test the implementation. Eventually, this prevented us from doing necessary alterations to the experimental set-up to enable depth estimation in images. On this note, we have chosen to further disregard the estimation of object translation in our geometric change detection approach. However, we consider this to be a feasible objective with some simple alterations. Thus, proposals for future alternations are presented in Section 5.3.




| | |
|---|---|
| Left x: | 157 |
| Top y: | 69 |
| Width: | 186 |
| Height: | 148 |

**Figure 3.10:** Example of detected bounding coordinates from YOLOv5

### 3.5.4 Pose Estimation: Feature Extraction

To extract image features and 3D shape features, we implement the deep pose estimation method presented by Xiao et al. [65]. The proposed method extracts features from both inputs in two separate, parallel branches, one for image features and one for 3D shape features. The method extracts image features using a CNN with 18 layers, called ResNet-18. In the parallel branch, 3D shape features are extracted through image rendering using the Blender module for Python [66]. The method implements virtual cameras pointing towards the input 3D model, rendering images from different model angles, as illustrated in Figure 3.11. A total of 216 images are rendered per input object. The rendered images are further used as input to a series of CNN's. A joint feature vector is then created by combining the feature vectors extracted from the image and the 3D CAD model.

**Figure 3.11:** A selection of rendered images using Blender

### 3.5.5 Pose Estimation: Orientation

The output used to estimate the pose of each input object is the three Euler angles indicating the orientation of the camera with respect to the reference frame of the object in the images. The angles are estimated using a multi-layer perceptron with three hidden layers, each followed by batch normalization and the ReLU activation function. The three estimated Euler angles are azimuth, elevation, and in-plane rotation, as described in Figure 3.12. A combined method of classification and regression uses the joint feature vector obtained earlier to estimate the angles. The angles are equally divided into several bins, for which a probability is calculated based on individual bin classification scores.

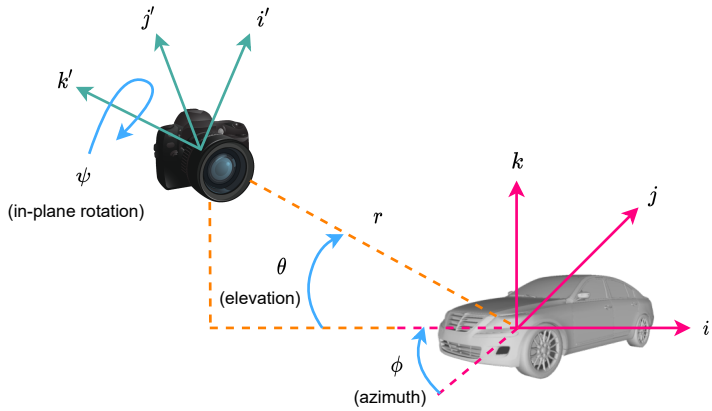



**Figure 3.12:** Illustration of angles azimuth, elevation and in-plane rotation describing the pose of a camera with respect to an object.

The main goal of the orientation estimation is to accurately align a provided 3D shape with an object in an image in terms of pose. The coordinate system of each given 3D shape is defined using three axes (i,j,k) with origo placed in the center of the shape. The three axes are set to point towards significant features in the provided shapes. Secondly, the camera's coordinate system (i',j',k') is defined with the front of the camera facing the 3D object along the negative axis of k'.

The rotation transformation of the two coordinate systems can be defined using a rotation matrix $\mathbf{R}(\phi, \theta, \psi)$, introducing three new variables; azimuth ($\phi$), elevation ($\theta$) and in-plane rotation ($\psi$). Likewise, the translation transformation of the camera is defined by a matrix $T(\phi', \theta', \psi')$ introducing distance $r$ as a new variable. The rotation matrix is computed from Euler angles using the formula in equation 3.1, where $\mathbf{R}_X$ and $\mathbf{R}_Z$ define rotations around the X and Z axis, respectively.

$$\mathbf{R} = \mathbf{R}_Z(\psi)\mathbf{R}_X\left(\theta - \frac{\pi}{2}\right)\mathbf{R}_Z(-\phi) \tag{3.1}$$

The method utilizes a single loss function combining outputs of classification and regression. The loss function sums cross-entropy loss from the classification with loss from the regression. The final network was trained on the ObjectNet3D dataset, as this provided

better system performance than the network trained on Pascal3D. Furthermore, the network was trained using the Adam optimizer for 300 epochs. The parameters used for training the network are presented in Table 3.2. The architecture of the 3D pose estimation network is illustrated in Figure 3.13.

| Parameter | Value |
|---|---|
| Batch size | 16 |
| Epochs | 300 |
| Device | GPU |
| Learning rate | $10^{-4}/10^{-5}$ |

**Table 3.2:** Choice of training parameters for the 3D pose estimation network

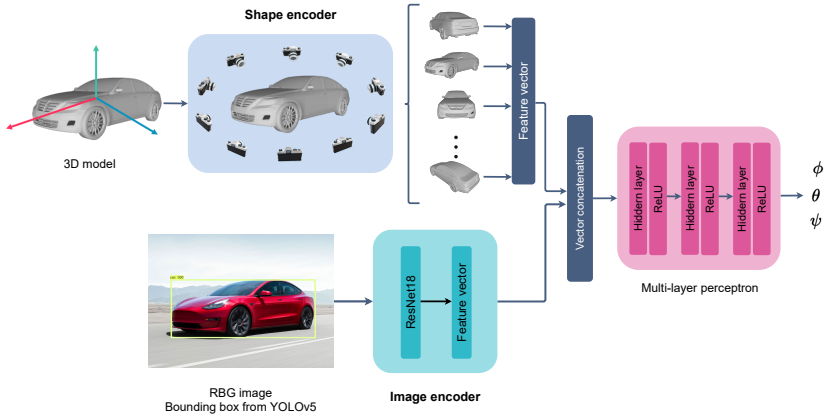



**Figure 3.13:** Architecture of the 3D pose estimation network implemented in Xiap et al. [66]

### 3.5.6 Pose Estimation: Evaluation Metric

The workflow of our geometric change detection approach consists of several modules. Each module was separately trained and tested to eventually optimize the combined workflow. Several tests were conducted using images of different 3D CAD models from various angles, to provide a thorough basis for evaluating system performance on 3D pose estimation.

Evaluating the 3D pose estimation results is a bit more complicated than evaluating those of our YOLOv5 object detection network. The 3D pose estimation network outputs the three Euler angles associated with the orientation of our camera with respect to the object it is looking at. Furthermore, the angle outputs are given in terms of azimuth, elevation, and in-plane rotation. The common interpretation of rotation relates to the azimuth angle,

measured as the angle between the perpendicularly projected vector from the camera to the object on the reference plane and a reference vector on the same reference plane, as previously illustrated in Figure 2.8.

| | | | |
|---|---|---|---|
| North | 0° | South | 180° |
| North-northeast | 22.5° | South-southwest | 202.5° |
| Northeast | 45° | Southwest | 225.5° |
| East-northeast | 67.5° | West-southwest | 247.5° |
| East | 90° | West | 270° |
| East-southeast | 112.5° | West-northwest | 292.5° |
| Southeast | 135° | Northwest | 315° |
| South-southeast | 157.5° | North-northwest | 337.5° |

**Table 3.3:** Definitions of true north-based azimuths

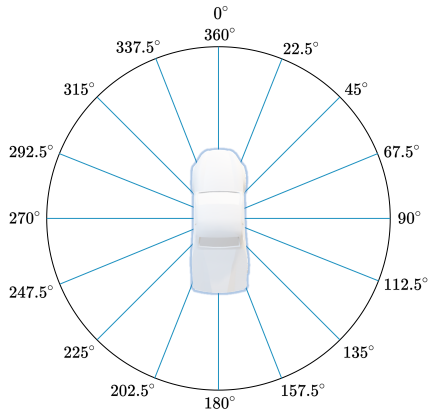



**Figure 3.15:** Illustration of the true north-based reference system

Furthermore, the azimuth estimation results are defined according to a true north-based azimuth reference system, presented in the table in Figure 3.3. Figure 3.14 was created as a tool to evaluate the accuracy of the predicted azimuth estimates. The figure displays a reference wheel from which one can verify the camera's angle of rotation according to the angle azimuth, where angle 0° refers to the angle directly north of the observer.

## 3.6 Overview of the Method

The workflow of our approach to geometric change detection is presented in Figure 3.16. The dataset used for testing our method was collected using an experimental set-up based on a camera tripod connected to a Raspberry Pi 4. The collected dataset contains videos of our set of 3D printed objects. Some of the videos are prone to different sources of noise. The dataset also consists of several images of the objects taken from different angles. Furthermore, we apply a DMD algorithm to detect motion in our videos. This is also tested for real-time videos.

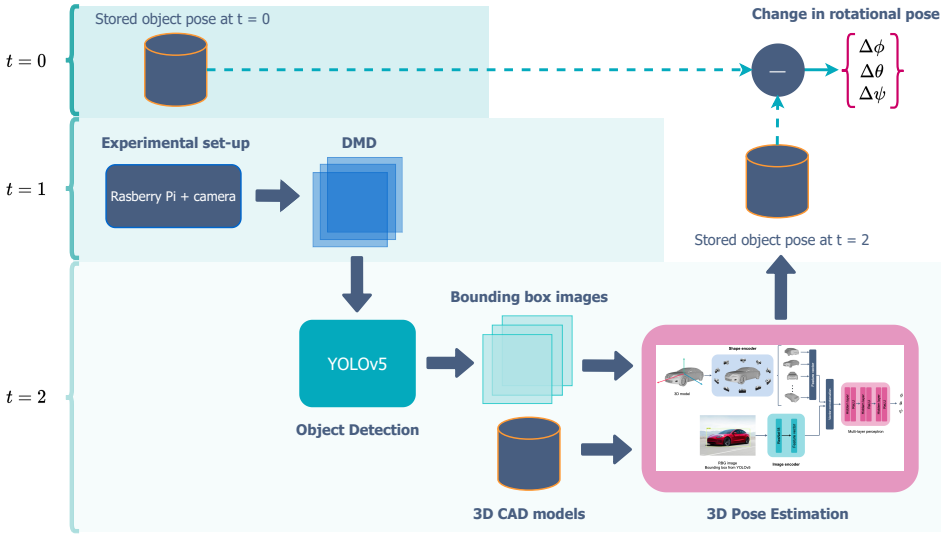



**Figure 3.16:** Workflow for geometric change detection

Video frames where DMD detects motion are used as input to the object detection network. Here, a pre-trained YOLOv5 network detects objects in the images and crops the images according to their bounding box estimations. The cropped images are further used as input to the pose estimation network, together with CAD models of our 3D objects.

The pose estimation network performs two subsequent tasks of feature detection. First, a shape encoder renders images of our input CAD model from different angles and elevations and inputs these to a CNN. The CNN then extracts object features in the rendered images and saves them in a feature vector. Secondly, the cropped bounding box images are used as input to a ResNet, which extracts object features in the images.

Furthermore, the two output feature vectors from our shape encoder and image encoder are connected to a single feature vector and used as input to a feed-forward ANN (multi-layer perceptron). The ANN utilizes a loss function combined with regression and classification. Eventually, the network outputs a pose estimate based on the camera's rotation with respect to the object in the image. The pose is based on three Euler angles; azimuth $\phi$, elevation $\theta$, and in-plane rotation $\psi$. By comparing the real change in rotation of our camera with the estimated rotation, we get the pose estimation error. The angle output from a 3D pose estimation may be defined using a rotation matrix, eventually describing an object's change in pose.

Chapter 4

# Results and Discussion

This section presents the results obtained in our approach to geometric change detection. First, we outline and discuss the results of performing motion detection using DMD. Secondly, we present results from our trained object detection network. Finally, we present the predicted results for 3D pose estimation based on our experimental set-up and discuss the viability of our results. We also discuss how our approach can be applied to a digital twin scenario.

## 4.1 Reference Models

The 3D printed CAD models used for all experiments throughout this project are presented in Figure 4.1. These are to be used as a reference when evaluating the results for DMD and pose estimation. However, due to symmetry issues and lack of texture, only the first eight models are used for testing the pose estimation method. This choice will be discussed later on, along with how results from using different input models vary.

**Figure 4.1:** 3D printed CAD models used in experiments

## 4.2 Motion Detection with DMD

The first module in our workflow involves real-time motion detection using DMD. In this section, we evaluate the performance of DMD on our video set and how suitable the algorithm is for performing background modeling. DMD is an essential part of our workflow, as it is where we collect the video frames that create the foundation for further processing. Our ability to first detect motion and then extract relevant video frames is challenged by some factors common for the task of image processing.

The video set used for evaluating DMD in this project consists of six videos recorded using our experimental set-up. The first three videos are recorded under similar circumstances without any significant noise sources. These are therefore referred to as baseline videos. The other three videos are subjected to different kinds of external noise, based on the common challenges outlined below. Results from processing all six videos are presented in the following section. The common challenges related to video processing and motion detection are:

- **Dynamic background**: Objects moving in the background without being a part of the relevant foreground, such as waves, rain, or trees moving in the wind.
- **Inactive foreground objects**: Foreground objects that are periodically stationary in video sequences makes it hard for DMD to differentiate.
- **Monochrome frames**: Images where the moving object have the same color, i.e. pixel intensity, as the background.
- **Lighting conditions**
  - **Deficient light conditions**: Poor lighting conditions are especially problematic this particular project when using fully black objects.
  - **Gradual changes in lighting**: The Raspberry Pi camera uses a few seconds to calibrate lighting conditions at the beginning of video sequences. This causes a change in lighting, which is sometimes recognized as movement by DMD.

The relevant videos were imported in our Python environment and tested for different configurations of DMD. Initially, the original videos were converted to gray-scale and re-scaled to conserve memory on the computer running the algorithm. Re-scaling, or down-sampling, is applied to reduce computational power and time. However, it is important to output image frames of decent quality in our application, as these images are further used as input to the change detection network. Therefore we chose a rather low re-scaling factor of 0.25, even though it significantly increases the processing time.

Furthermore, video frames were flattened and stacked as the column vectors of a matrix $\mathbf{X}$. Then, steps of computing the SVD of $\mathbf{X}$ and the components and eigenvalue decomposition of $\tilde{\mathbf{A}}$ are implemented according to Algorithm 3.5.1.

After computing the DMD modes, the background is separated from the foreground.

DMD does this according to the explanation in Section 2.1.1. The computed DMD modes with the lowest oscillation frequency, i.e. smallest changes in time, are defined as the static background. Furthermore, we can extract the background from the original frame, leaving only the foreground.

Visual evaluation results of background subtraction for three baseline videos from our dataset are presented in Figure 4.2. The top row shows an example frame from the original video. In the second row, the predicted background frame is based on computed DMD modes with low oscillation frequency. The third row displays the predicted foreground frame based on computed DMD modes with high oscillation frequency. Finally, the fourth row displays the filtered results after extracting the predicted background frame from the original video frame. These videos are defined as baseline results since none of them are subjected to any external noise sources.

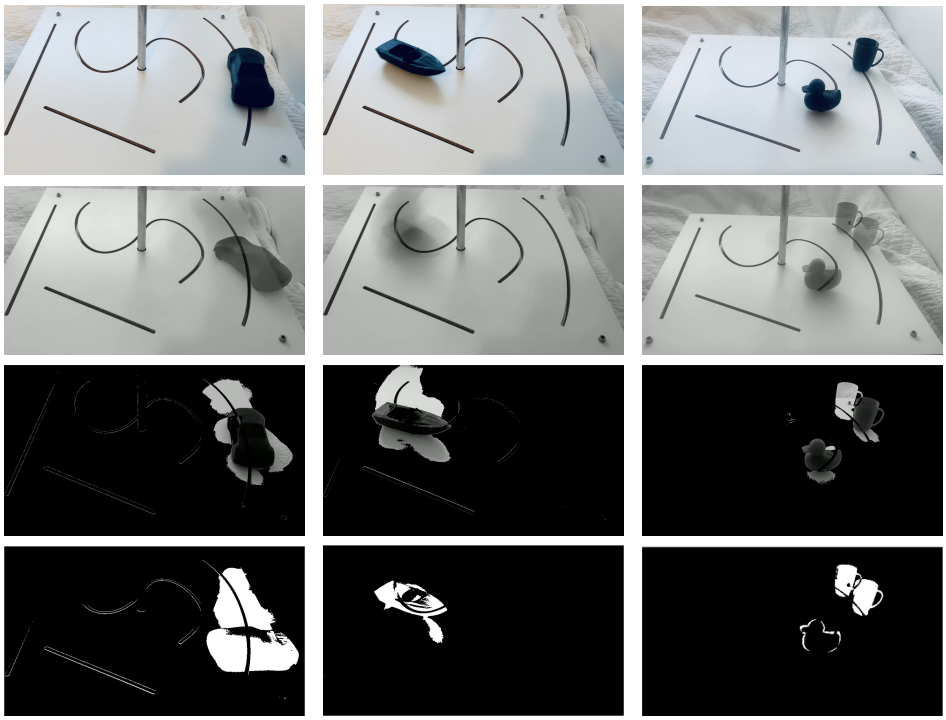


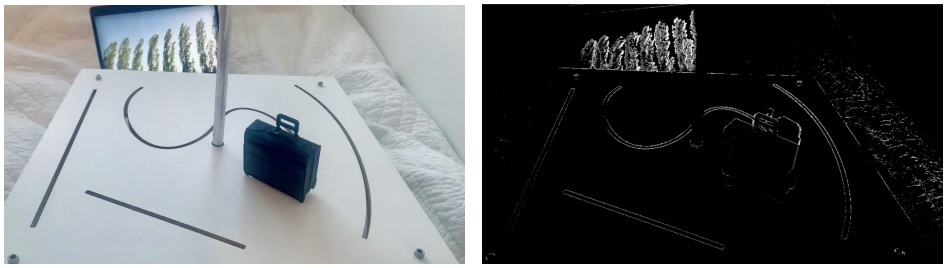
**Figure 4.2:** Visual evaluation results for example frames from three baseline videos. The top row shows the original video frames. The second and third row shows the predicted static background and the predicted foreground frame, respectively. The fourth row shows the filtered foreground after subtracting the background from the original frame.

Looking at the results in Figure 4.2, the DMD algorithm appears to successfully classify DMD modes based on their associated high or low oscillation frequencies. The results of background predictions displayed in the second row are not perfect, as contours of the foreground objects are visible. However, judging by the results in row four, the slightly inaccurate background predictions does not seem to affect the final filtered foreground results.

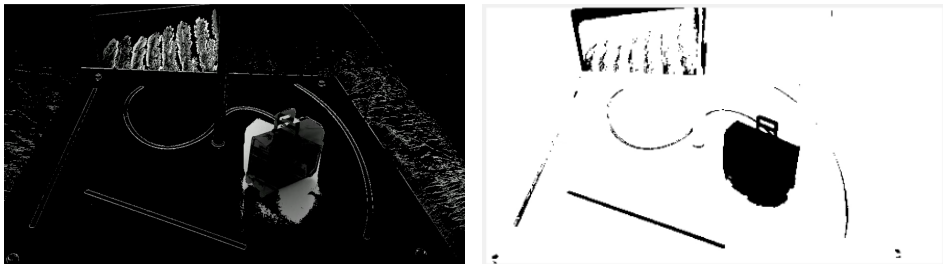
The left image in Figure 4.3a displays an original video frame from a video where subtle background movement is applied as a source of noise. A screen displaying trees blowing in the wind is displayed in the frame's background while the motion from the moving foreground object is detected. Initially, the 3D object is static. DMD then detects the dynamic background as the foreground, as displayed in the image to the right.

Furthermore, the left image in Figure 4.3b displays the DMD results for the predicted foreground when the 3D object starts moving. Both the dynamic background and the moving 3D object are now predicted to be parts of the foreground. Finally, the predicted background is extracted from the original video frame and the resulting foreground is filtered. This result is displayed in the image to the right, illustrating how DMD now considers almost the whole frame as a part of the foreground. The visual results clearly show that distinguishing dynamic background changes from moving objects in the foreground is a limitation to DMD.

Another weakness of DMD is the method's ability to detect moving foreground objects with the same pixel intensity as the background, i.e., monochrome frames. These visual results are presented in Figure 4.5. Furthermore, Figure 4.4 presents visual DMD results of a video where a sudden change in lighting conditions is applied as a noise source. Changes in lighting conditions as a noise factor resulted in detections quite similar to the baseline results. In fact, by comparing the processed frames from before and after the sudden change in lighting, it actually seems as if some previous noise factors were removed from the video frames. Change in lighting conditions might not be as much of an issue when running in real-time compared to running on prerecorded videos due to the need for initial camera calibrations.



**(a)** Original video frame to the left and DMD foreground prediction of the dynamic background to the right.



**(b)** DMD foreground prediction of the dynamic background and the moving 3D object to the left, and the filtered foreground to the right.

**Figure 4.3:** DMD results for a scene subjected to noise from a dynamic background

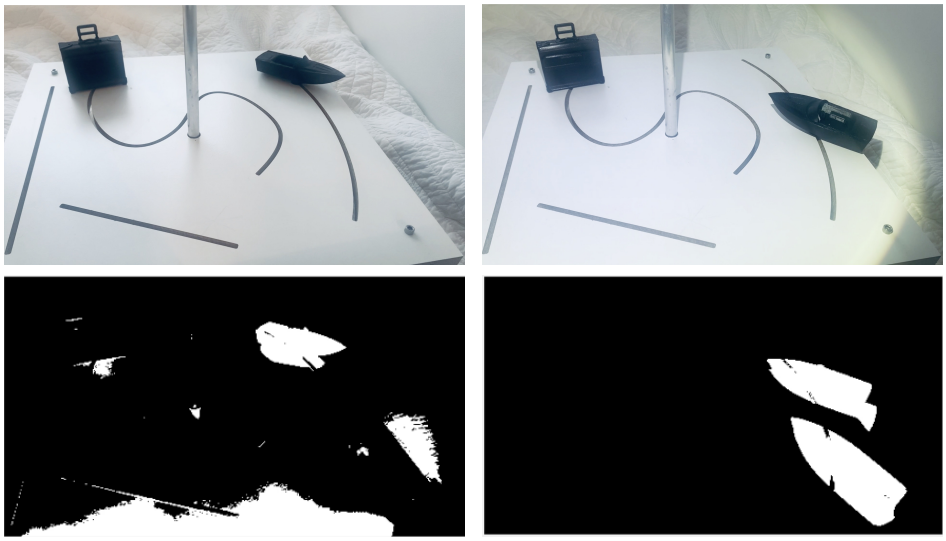


**Figure 4.4:** DMD results for a scene subjected to sudden changes in lighting conditions. The first row displays original frames, and the second row displays filtered foreground frames.

**Figure 4.5:** DMD results on foreground object with same pixel intensity as background object. The original frame frame is displayed to the left and the filtered foreground is displayed to the right.

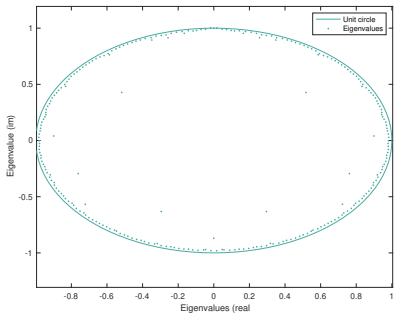
The spectrum of exponential eigenvalues and computed SVD modes for the six videos used for testing are plotted in Figure 4.6 and Figure 4.7, respectively. The eigenvalues are all plotted within the unit circle, with absolute values less than one. This indicates that the modes contain a stable evolution without signs of neither descending nor decreasing growth. From the SVD spectra in the right column, it is clear that most of the SVD modes are related to low oscillation frequencies, indicating little or no detected movement. The modes with higher frequency thus contain more spatial information.

The results presented in this section display some of the strengths and limitations of the DMD algorithm. The suitability of using DMD for motion detection is evaluated on real data collected using our experimental set-up.

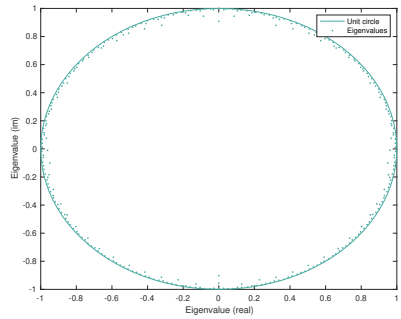
We chose to use DMD over PCA methods based on arguments from previously published results. Results from [11] and [10] argue that various DMD methods prove viable candidates for applying fast processing of high-quality videos, especially when fast processing is considered more important than high accuracy. Fast processing enables real-time detection, which is a key element in our application. It may also be stated that post-processing techniques may enhance system performance in challenging scenes. The main challenge of DMD revealed through our experiments relates to noise from dynamic backgrounds and inactive foreground objects.

Real-time DMD applications are expensive given high-resolution videos, both in terms of computational time and memory. However, the need for a full video reconstruction is often unnecessary for performing background subtraction. In many applications, determining a static background model is sufficient for performing background subtraction. A process of mode selection must be completed in order to find the static background model that most accurately represents the original background.

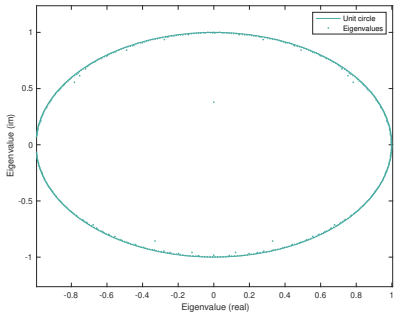
Furthermore, reducing computational cost and runtime is necessary to ensure sufficient real-time performance. To this end, [10] presents a solution where DMD is computed on a combination of original and compressed data to accelerate calculations.



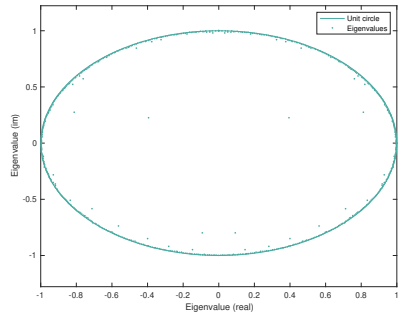

**(a)** Baseline video 1



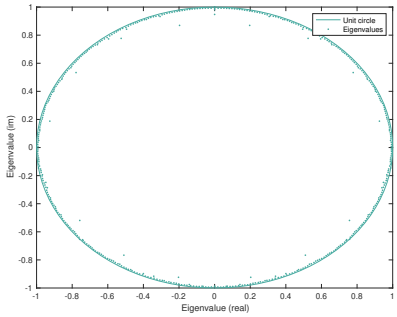

**(b)** Baseline video 2



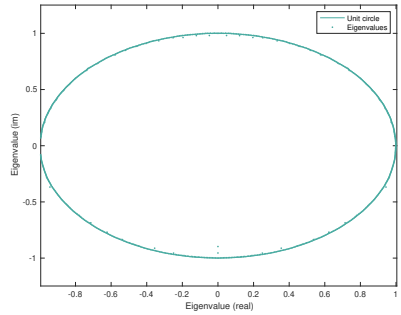

**(c)** Baseline video 3




**(d)** Dynamic background

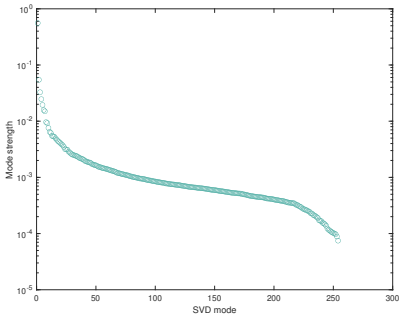



**(e)** Change in lighting

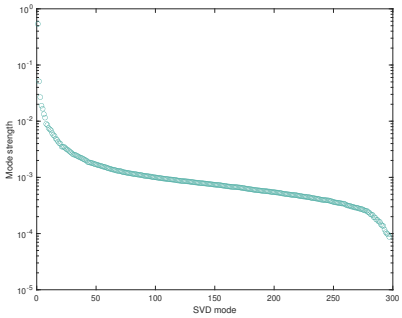



**(f)** Monochrome frame

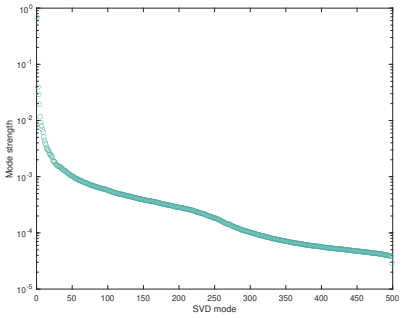
**Figure 4.6:** Complex eigenvalues from DMD plotted for the six videos in the dataset



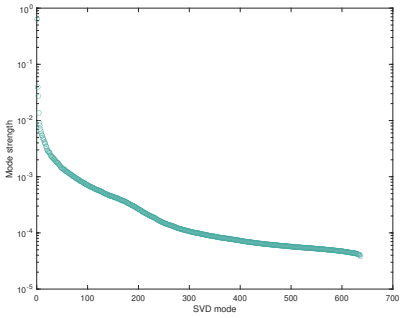

**(a)** Baseline video 1



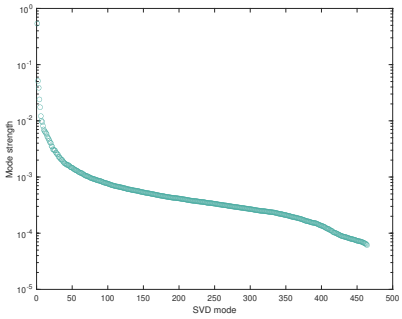

**(b)** Baseline video 2



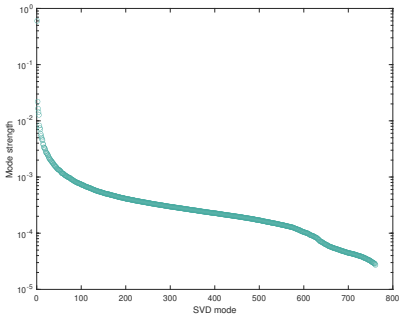

**(c)** Baseline video 3




**(d)** Dynamic background




**(e)** Change in lighting




**(f)** Monochrome frame

**Figure 4.7:** Normalized spectrum of SVD modes from DMD plotted for the six videos in the dataset

## 4.3 Object Detection using YOLOv5

The object detection algorithm YOLOv5 was implemented in PyTorch and trained for 300 epochs on a dataset collected from our experimental set-up. The dataset consists of 463 images. We chose to use the network model YOLOv5x, which is currently the biggest available network in the YOLOv5 family.

YOLOv5 outputs images containing labeled bounding boxes for all detected objects. In computer vision tasks, it is commonly considered important to build systems not dependent on being trained for previously seen categories to enable detection of novel categories in test environments. However, the situation is arguably different in a digital twin setting. Most digital twins models aim to be highly accurate imitations of real-life systems. Therefore, specific 3D objects present in a digital twin system will already be known, along with system dynamics, possible noise sources, etc. Training a system for specific object categories will significantly improve system performance while also being a relevant solution to this fully supervised system.

Different configurations of YOLOv5 were tested using images of the 3D CAD models in our experimental set-up. System performance is measured by comparing the ground truth data, i.e., manually labeled images, to data labeled by our YOLOv5 network. After applying preprocessing and augmentation to our dataset, the final YOLOv5 model was trained on the 463 annotated images. A precision-recall curve and training loss curves were used as diagnostic tools for deciding when the model was sufficiently trained.

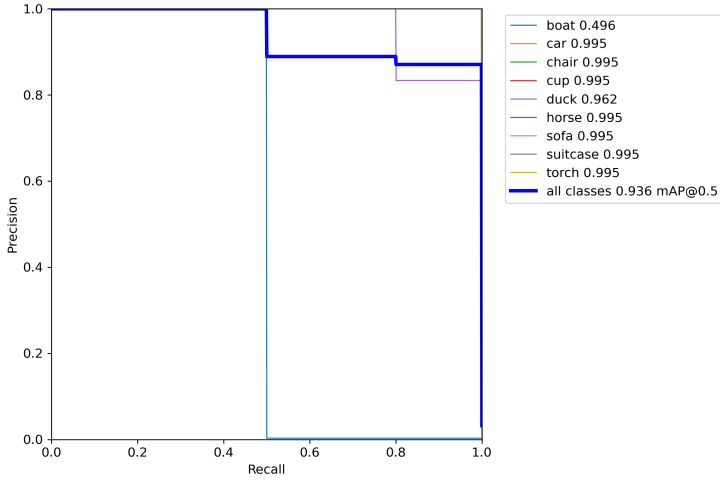



**Figure 4.8:** Precision-recall curve for our nine object categories from YOLOv5

Figure 4.8 displays the plotted precision-recall curve for our object detection network on nine object categories. The overall mean network performance on all object classes results in a high mAP of 0.936. This is a good result clearly marked by the precision-recall curve plotted as a thick blue line. Most object categories obtain high mAP results, resulting in a high average score. However, the boat category significantly reduces the average, with an mAP score of 0.496. There may be many reasons for this, including an uneven distribution of boat objects in the datasets used for training. The YOLOv5 network was not trained for perfection in our geometric change detection approach. The network was merely trained to illustrate how an object detection network can be trained using collected data for a specific purpose. On this note, we find that the results illustrated in Figure 4.8 are sufficient for our project.

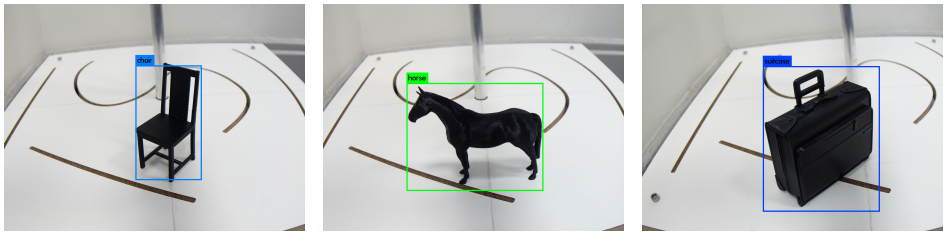



**Figure 4.9:** Examples of objects detected by YOLOv5

Figure 4.9 illustrates an example of detected objects and their respective bounding boxes. Our YOLOv5 network does this for most iterations. Furthermore, we know, based on several previously published sources, that the YOLO network can be trained to output close to perfect predictions given proper training data and the right configurations.

When starting this research project, we chose to focus on estimating object pose in terms of rotation, assuming translation would be fairly simple in comparison. However, the translation did not appear to be as easy to estimate as first assumed, given the experimental set-up we chose.

The initial idea was to utilize bounding box coordinates based on YOLOv5 object detection combined with triangle similarities. Bounding box coordinates provide the position of a detected object in 2D, assuming a stationary camera with a known orientation. The distance from the camera to the detected object can then be found using triangle similarities in images. We can calculate the perceived focal length from our camera to the object given objects of known width. A round of calibration is initially required, where we provide the algorithm with the object width and distance.

In a stationary set-up, where all objects are facing the camera from the side, the method of calculating perceived focal length seemed to work reasonably well. However, we soon realized that initializing the camera orientation would be challenging in a more realistic setting, since our current experimental set-up includes no way of accurately knowing which way our camera is pointing. Some improvement for the experimental set-up to counteract these challenges are presented in 5.3.

## 4.4 3D Pose Estimation

Following the video frame extraction from our motion detection algorithm, we evaluated the performance of our implemented pose estimation system. As outlined in Section 3.5, our evaluation method for pose estimation is based on qualitative results from repeated tests on image frames containing different objects from our set of 3D CAD models. The pose estimation algorithm was tested on objects within nine different object categories.

The algorithm we used builds upon a network that does not require training on specific categories used for testing [65]. The intention is for the estimation method to work on novel object categories. The method requires a 3D CAD model in OBJ format as input, used for rendering images of each object from different angles, as explained in Section 2.3. It also requires the input of a cropped out bounding box containing the object used for pose estimation. This is the purpose of our trained YOLOv5 network.

### 4.4.1 Rotation

Given a 3D CAD model and a single RBG image cropped as each object's bounding box, the pose estimation algorithm predicts the camera's orientation with respect to the object we are looking at. The network, presented in Section 2.3, outputs the three Euler angles equivalent to the camera's orientation. A change in the camera orientation is equivalent to a change in pose given a stationary camera. We will further address the estimated angle output as the estimated object pose. Our aim is for the predicted pose to have an orientation as similar to the object in the original image as possible.

In a digital twin scenario, we can assume that we would have a camera rig of known parameters. Thus, estimating our camera's elevation with respect to the "ground" may be irrelevant. The same argument can be used for the in-plane rotation angle. However, the object pose is estimated as a combination of these three angles, therefore we cannot simply discard some angles. Instead, we chose to focus on the azimuth angle for evaluation. All presented results are thus based on the most accurate azimuth predictions.

The estimated angles from the pose estimation algorithm trained on ObjectNet3D and based on best-fit azimuth angle are shown in Table 4.1. We chose to base our results on the azimuth angle as this is most relevant to our objective. This angle is also easily evaluated. Each experiment predicts a change in angle between two different object poses. No pair of object poses are the same for any experiments. We rank the predictions based on azimuth estimations and present the best results for selected object poses. Results are presented both as estimated angle error in degrees and as a percentage error. Percentage errors for azimuth predictions are calculated according to equation 4.1.

$$\text{Percentage error} = \frac{\mid \Delta\phi_{real} - \Delta\phi_{pred} \mid}{\Delta\phi_{real}} * 100 \tag{4.1}$$

Percentage errors for elevation and in-plane rotation are calculates as the percentage of the total range of each angle; 180 degrees for elevation and 360 degrees for in-plane rotation. We calculate the errors the following way due to the fact that the true changes in both elevation and in-plane rotation are zero for all our experiments.

Out of the total eleven 3D printed objects presented in Figure 4.1, the results of seven are presented in Table 4.1. Pose estimations for the two boat models are combined in the results for the boat object category. The remaining three objects were discarded from the testing phase because they proved to be unrecognizable to the pose estimation algorithm.

The cup model was discarded due to rotational symmetry, which is a common problem in pose estimation applications [67, 68, 69]. The model is completely symmetrical from most angles, as the cup handle is the only remark that may be used to determine the object pose. Therefore, the pose estimation algorithm did not predict any meaningful results for the cup model. However, when comparing prediction results for test images taken from a high elevation and at ground level, the pose estimation algorithm managed to predict satisfying results for the elevation angle. Even so, as we decided to focus on predicting the azimuth angle, these results were considered insufficient. Furthermore, we discarded the duck model due to a lack of texture. Both the white and the black models are assumingly unrecognizable to the pose estimation algorithm, as it fails to predict any results for both objects.

No substantial difference was found between novel object categories and object categories used for training the pose estimation network. There may be many reasons for this, including that the 3D objects in our images aren't close in resemblance to the images of real-life objects used for training. However, the result indicates that the 3D CAD model used as input is enough for the network to predict decent results, regardless of the object class.

Combined estimation errors for all three output angles, based on best-fit azimuth, are visualized in Figure 4.10. Based on these results, it is the object categories car, chair, and torch that performs worst for elevation and in-plane rotation. However, by merely looking at estimation results for best-fit azimuth alone in Figure 4.11a, prediction results are sufficiently accurate.

| Obj. class & expt. no. | boat 1 | boat 2 | boat 3 | boat 4 | boat 5 | car 1 | car 2 | car 3 | car 4 | car 5 | chair 1 | chair 2 | chair 3 | chair 4 | chair 5 | horse 1 | horse 2 | horse 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Angle error estimate | | | | | | | | | | | | | | | | | | |
| $\lvert(\phi_f - \phi_0)\rvert$ | 1.95 | 0.74 | 0.19 | 2.56 | 0.39 | 1.48 | 4.37 | 1.86 | 0.38 | 1.22 | 2.28 | 0.40 | 2.07 | 2.12 | 2.09 | 0.44 | 2.15 | 2.72 |
| $\lvert(\theta_f - \theta_0)\rvert$ | 2.42 | 28.82 | 0.60 | 3.06 | 0.38 | 0.13 | 32.48 | 17.90 | 18.03 | 32.99 | 1.00 | 0.35 | 22.98 | 28.46 | 28.24 | 14.43 | 1.17 | 16.99 |
| $\lvert(\psi_f - \psi_0)\rvert$ | 2.16 | 1.39 | 0.18 | 3.52 | 0.45 | 0.55 | 13.71 | 12.45 | 11.90 | 27.29 | 1.63 | 0.62 | 16.63 | 16.63 | 16.60 | 30.34 | 0.44 | 0.15 |
| (%) error estimate | | | | | | | | | | | | | | | | | | |
| $\lvert(\phi_f - \phi_0)\rvert$ | 0.54 | 0.21 | 0.05 | 0.71 | 0.1 | 0.41 | 1.21 | 0.52 | 0.11 | 0.34 | 0.63 | 0.11 | 0.58 | 0.59 | 0.58 | 0.12 | 0.6 | 0.76 |
| $\lvert(\theta_f - \theta_0)\rvert$ | 1.34 | 16.01 | 0.33 | 1.7 | 0.21 | 0.07 | 18.04 | 9.94 | 10.02 | 18.33 | 0.56 | 0.19 | 12.77 | 15.81 | 15.69 | 8.02 | 0.65 | 9.44 |
| $\lvert(\psi_f - \psi_0)\rvert$ | 0.6 | 0.39 | 0.05 | 0.98 | 0.13 | 0.15 | 3.8 | 3.46 | 3.31 | 7.58 | 0.45 | 0.17 | 4.62 | 4.61 | 4.61 | 8.43 | 0.12 | 0.04 |

| Obj. class & expt. no. | horse 4 | horse 5 | torch 1 | torch 2 | torch 3 | torch 4 | torch 5 | sofa 1 | sofa 2 | sofa 3 | sofa 4 | sofa 5 | sc 1 | sc 2 | sc 3 | sc 4 | sc 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Angle error estimate | | | | | | | | | | | | | | | | | |
| $\lvert(\phi_f - \phi_0)\rvert$ | 2.40 | 0.27 | 2.26 | 0.40 | 1.62 | 2.05 | 0.52 | 0.85 | 1.36 | 3.82 | 2.93 | 3.41 | 0.02 | 2.25 | 1.10 | 2.21 | 0.50 |
| $\lvert(\theta_f - \theta_0)\rvert$ | 3.39 | 0.09 | 32.16 | 0.72 | 30.89 | 31.14 | 0.85 | 1.53 | 0.47 | 18.25 | 16.73 | 1.41 | 0.57 | 19.10 | 1.91 | 34.43 | 0.12 |
| $\lvert(\psi_f - \psi_0)\rvert$ | 0.40 | 0.16 | 1.69 | 30.97 | 29.97 | 0.36 | 1.16 | 0.16 | 0.85 | 1.88 | 1.71 | 0.65 | 2.07 | 1.18 | 1.16 | 2.37 | 2.06 |
| (%) error estimate | | | | | | | | | | | | | | | | | |
| $\lvert(\phi_f - \phi_0)\rvert$ | 0.67 | 0.08 | 0.63 | 0.11 | 0.45 | 0.57 | 0.14 | 0.24 | 0.38 | 1.06 | 0.81 | 0.95 | 0.01 | 0.63 | 0.31 | 0.61 | 0.14 |
| $\lvert(\theta_f - \theta_0)\rvert$ | 1.88 | 0.05 | 17.87 | 0.4 | 17.16 | 17.3 | 0.47 | 0.85 | 0.26 | 10.14 | 9.29 | 0.78 | 0.32 | 10.61 | 1.06 | 19.13 | 0.07 |
| $\lvert(\psi_f - \psi_0)\rvert$ | 0.11 | 0.04 | 0.47 | 8.6 | 8.33 | 0.1 | 0.32 | 0.04 | 0.24 | 0.52 | 0.48 | 0.18 | 0.58 | 0.33 | 0.32 | 0.66 | 0.57 |

**Table 4.1:** Pose estimation errors presented for five experiments per object category, for a total of seven categories. Results are based on best-fit azimuth predictions, and are presented both as estimated angle errors in degrees and as percentage errors based on the given range of each angle.

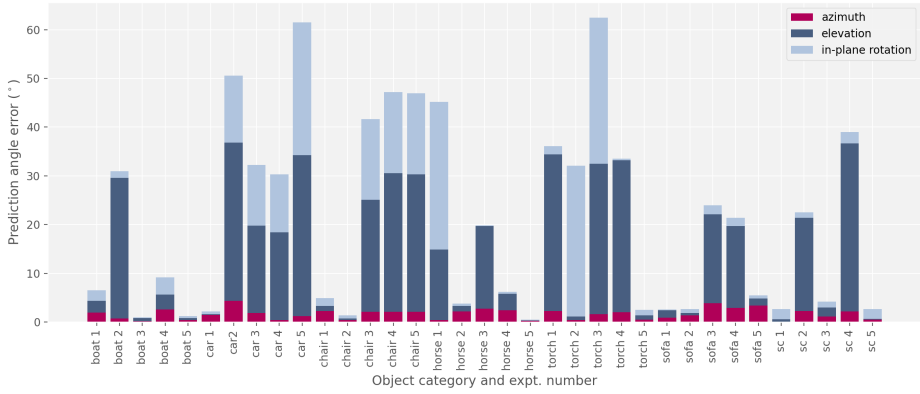
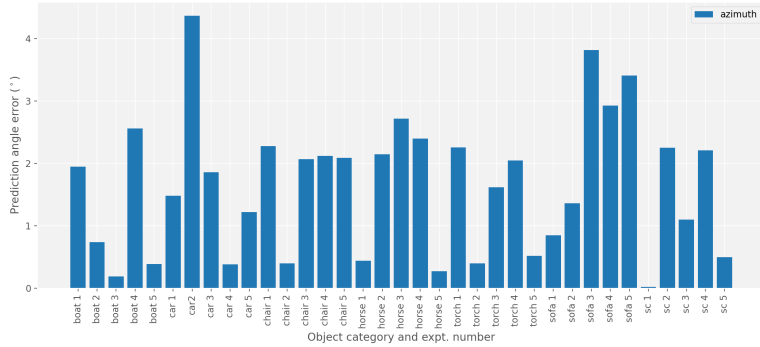
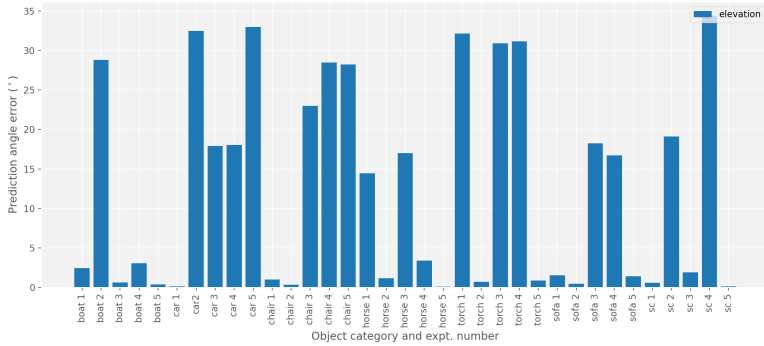


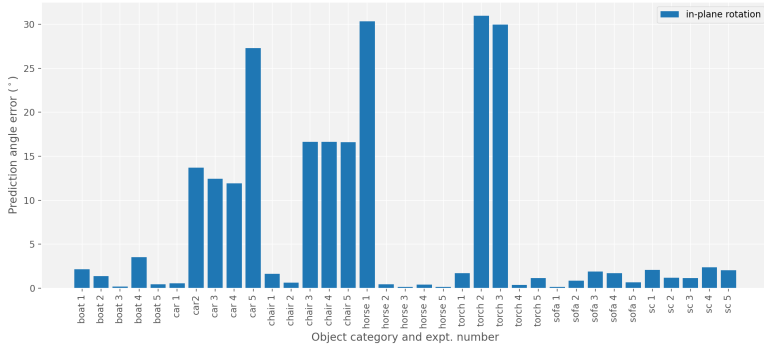

**Figure 4.10:** Combined angle error estimates for azimuth, elevation and in-plane rotation on five conducted experiments per object category, for a total seven object categories. Presented results are based on best-fit azimuth predictions.




**(a) Azimuth**

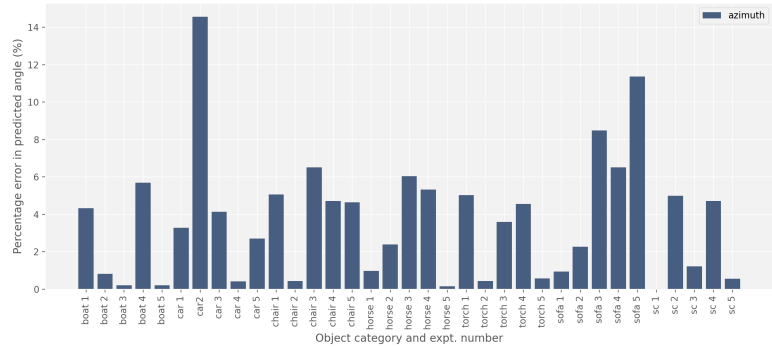



**(b) Elevation**

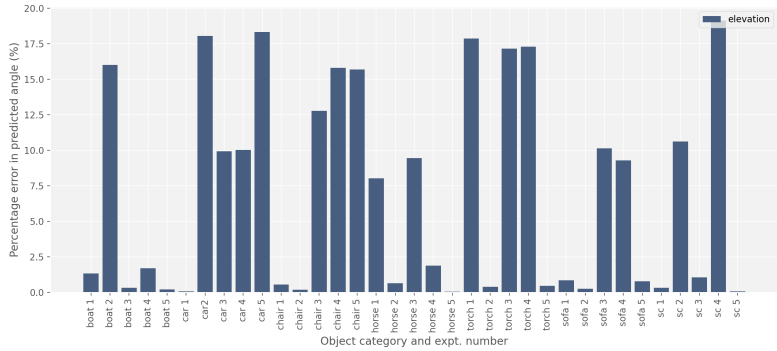



**(c) In-plane rotation**

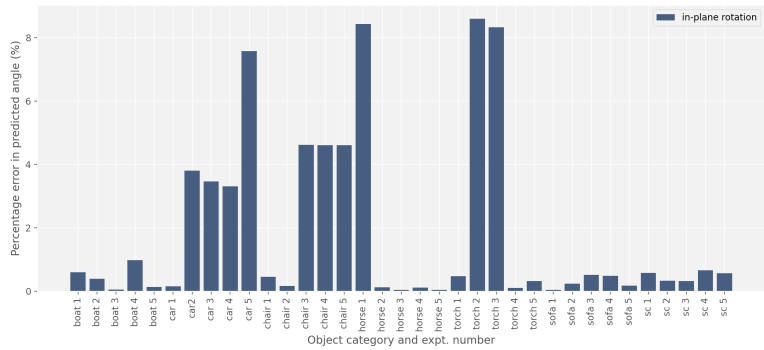
**Figure 4.11:** Estimated angle errors results in degrees for azimuth, elevation and in-plane rotation, respectively, based on best-fit azimuth predictions.




**(a) Azimuth**




**(b) Elevation**




**(c) In-plane rotation angle**

**Figure 4.12:** Percentage errors of predicted angles for elevation and in-plane rotation, respectively, based on best-fit azimuth predictions. Azimuth errors are calculated according to equation 4.1. Errors for rotation and in-plane rotation are calculated based on the total range of each angle; $180^\circ$ for elevation and $360^\circ$ for in-plane rotation.

## Recognizing Causes of Failure

Several aspects have been proved to affect the pose estimation results of our experimental set-up. The most prominent are:

- **Symmetrical objects**: Detecting the pose of partly or fully symmetrical objects is a key challenge in pose estimation. Specific measures have to be applied in order to address this challenge [70, 69]. Our 3D cup model is not entirely symmetrical due to the handle; however, from most rotations around the angle azimuth, the object appears to be entirely symmetrical about the rotational axis.
- **Black and texture-less objects**: Detecting the pose of monochrome, texture-less objects are a challenge for the 3D pose estimation network. Most of our objects are 3D printed as black CAD models, making it difficult to accurately detect object texture using our experimental set-up. Ensuring proper lighting conditions throughout experiments has therefore been a key priority. However, even in bright daylight, shadows may cause objects to appear texture-less in images.
- **Shape similarities from different angles**: The shape of some objects appears similar even when seen from completely different angles. This typically poses a challenge to the pose estimation network, especially when it appears in combination with poor lighting conditions or texture-less objects, as previously mentioned. For instance, the front and back of the 3D car model may appear very similar in images, making it almost impossible for the pose estimation algorithm to correctly distinguish between the different orientations.

Several different solutions performing 3D pose estimation were researched and tested upon the initialization of this project. Many of these solutions have in common that they are built to provide good system performance on specific issues that haven't been sufficiently handled earlier, for instance, object symmetries or real-time pose estimation of objects in videos. Furthermore, various methods require different formats of input data and different training regimes.

We require a universal solution that is easily applicable to new, collected data for our geometric change detection approach. We also require single image inputs to make predictions based on video frames extracted from detected motion sequences. The pose estimation network implemented in Xiao et al. [65] provided all this while also enabling training on new datasets. In general, we find that requiring accurate 3D models as inputs is unsuitable for many applications. However, in our case, we may use the same argument as previously, stating that all objects would already be known to system operators in a digital twin context. Thus, providing 3D models of relevant objects can be considered a feasible task.

During research into currently existing solutions, we found that few solutions actually publish software frameworks or source code. As mentioned earlier, the field of 3D pose estimation has seen rapid growth over the last couple of years. Numerous solutions are developed to handle various issues, and performance is evaluated on how well these solutions can be applied in real-world scenarios. However, without published source-code,

we are unable to reproduce results and test the methods for our application. This is a widely known issue in the machine learning community, addressing the need for more open-source solutions to ensure scientific progress [71]. Furthermore, many AI-based applications are black-box methods, making it impossible for outsiders to neither understand nor reproduce system results without knowing specific system parameters and training regimes [72].

The rapid advances in tools and frameworks built for the machine learning community also fuel another issue. Compatibility between different machine learning frameworks are often limited and requires specific release versions. Therefore, implementation and testing of previously published machine learning solutions becomes a process aiming to find compatible versions of all required frameworks [73]. This problem is commonly addressed using package managers and creating virtual environments in Python or Anaconda.

Chapter 5

# Conclusion and Future Work

## 5.1 Conclusion

The initial objective of this thesis was:

*To combine the fields of 3D solid modeling and machine learning and explore the possibilities of using a novel approach of geometric change detection in the context of a digital twin, to minimize the amount of collected information while still being able to recreate a 3D scene on demand.*

We achieved the above objective by reducing the amount of data recorded to the bare minimum using Dynamics Mode Decomposition, and by exploiting the information contained in 3D CAD models, which are generally discarded in traditional image based object detection and classification algorithms, using 3D machine learning. The main contributions of this thesis can be enumerated as follows:

- A cost-effective experimental set-up for analyzing and validating 3D machine learning algorithms in the context of digital twins.
- A workflow consisting of Dynamic Mode Decomposition, 2D object detection, and 3D pose estimation based on CNN's to estimate rotational changes in three dimensions.

The workflow can be extended and utilized to perform change detection in digital twins, given that each module in the workflow are specifically trained for objects present in the monitored environment. The camera technology and lighting must be sufficiently tailored for the expected scene conditions. Although we have not demonstrated it here, the workflow proposed in this work will also be capable of estimating the pose of multiple objects in a single image, since the 2D object detection module consisting of YOLO already takes this into account. The method demonstrated in this work will significantly reduce the amount of data archived (6DoF pose instead of images and videos) to recreate the scene at any point of time, on demand.

## 5.2 Lessons learned

In retrospect, our research efforts exposed several aspects of the work that could have been carried out differently. If the author were to reproduce the same work again, some changes would have been made in the project's early phase.

Most importantly we would ensure that a meticulous plan was in place before starting the design phase. Furthermore, we would employ a more systematic approach to choosing design parameters through rigorous design of the experiments and the experimental set-up. For our choice of solid models in the experimental set-up we would include variations in size, color, shape and texture. We would also aim to quantify the uncertainties at an early stage, in order to avoid certain error sources in our design.

## 5.3 Future Work

The current work introduced a novel concept to geometric change detection and demonstrated its value. However, to make the approach more robust and accurate the following extensions to this work are proposed:

- The experimental set-up has no efficient solution for identifying the camera's current orientation, which is why we cannot demonstrate reliable results for estimated object translation. Therefore, we cannot claim to estimate the full 6DoF pose of our 3D objects in images. A simple and low-cost solution to this problem is to apply a laser pointer to the camera, enabling identification of the current camera viewpoint, which can then be initialized with an orientation pointing in the direction of the relevant objects.

- Future testing and enhancements in terms of reliability and robustness require training the full system pipeline for specific data relevant to a potential digital twin application. Thus, the individual networks need to be trained explicitly on the 3D models present in the application. Considering the system model should be trained for specific object categories in future iterations, the need for 3D pose estimation of novel categories is irrelevant. Solutions only applicable to pre-trained categories might enhance the system's robustness and reliability through more accurate system performance.

- For our solution, we deliberately chose to utilize low-cost equipment to achieve our objective. However, other solutions are available by discarding cost requirements. We found it challenging to estimate all three translational coordinates in RGB images; with depth being the real challenge, RGB-D cameras (depth sensors) can extract depth information from images on a per-pixel basis. Thus, several pose estimation methods are developed specifically for using RBG-D images. Where solutions for RGB images require knowledge of object dimensions to compute perceived focal length from the camera to the object, systems with depth sensors have one less thing to worry about.

Some of the above work was to be included in the current work. However, it had to be discarded as a result of limited access to the experimental facility as a result of the COVID-19 restrictions at the time.

# Bibliography


[1] A. Rasheed, O. San, and T. Kvamsdal. Digital Twin: Values, Challenges and Enablers From a Modeling Perspective. *IEEE Access*, 8:21980–22012, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.2970143. Conference Name: IEEE Access.

[2] Victor Singh and K. E. Willcox. Engineering Design with Digital Thread. *AIAA Journal*, 56(11):4515–4528, 2018. ISSN 0001-1452. doi: 10.2514/1.J057255. URL https://doi.org/10.2514/1.J057255. Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/1.J057255.

[3] Azad M. Madni, Carla C. Madni, and Scott D. Lucero. Leveraging Digital Twin Technology in Model-Based Systems Engineering. *Systems*, 7(1):7, March 2019. doi: 10.3390/systems7010007. URL https://www.mdpi.com/2079-8954/7/1/7. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[4] Yu Zheng, Sen Yang, and Huanchong Cheng. An application framework of digital twin and its case study. *Journal of Ambient Intelligence and Humanized Computing*, 10(3):1141–1153, March 2019. ISSN 1868-5145. doi: 10.1007/s12652-018-0911-3. URL https://doi.org/10.1007/s12652-018-0911-3.

[5] R. Minerva, G. M. Lee, and N. Crespi. Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models. *Proceedings of the IEEE*, 108(10):1785–1824, October 2020. ISSN 1558-2256. doi: 10.1109/JPROC.2020.2998530. Conference Name: Proceedings of the IEEE.

[6] Timothy D. West and Mark Blackburn. Is Digital Thread/Digital Twin Affordable? A Systemic Assessment of the Cost of DoD's Latest Manhattan Project. *Procedia Computer Science*, 114:47–56, January 2017. ISSN 1877-0509. doi: 10.1016/j.procs.2017.09.003. URL http://www.sciencedirect.com/science/article/pii/S1877050917317970.

[7] C. L. Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, 275:314–347, August 2014. ISSN 0020-0255. doi: 10.1016/j.ins.2014.01.015. URL http://www.sciencedirect.com/science/article/pii/S0020025514000346.

[8] Danilo Jimenez Rezende, S. M. Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised Learning of 3D Structure from Images. *Advances in Neural Information Processing Systems*, 29: 4996–5004, 2016. URL https://papers.nips.cc/paper/2016/hash/1d94108e907bb8311d8802b48fd54b4a-Abstract.html.

[9] Niall O' Mahony, Sean Campbell, Lenka Krpalkova, Daniel Riordan, Joseph Walsh, Aidan Murphy, and Conor Ryan. Computer Vision for 3D Perception. In Kohei Arai, Supriya Kapoor, and Rahul Bhatia, editors, *Intelligent Systems and Applications*, Advances in Intelligent Systems and Computing, pages 788–804, Cham, 2019. Springer International Publishing. ISBN 978-3-030-01057-7. doi: 10.1007/978-3-030-01057-7_59.

[10] N. Benjamin Erichson, Steven L. Brunton, and J. Nathan Kutz. Compressed Dynamic Mode Decomposition for Background Modeling. *Journal of Real-Time Image Processing*, 16(5):1479–1492, October 2019. ISSN 1861-8200, 1861-8219. doi: 10.1007/s11554-016-0655-2. URL http://arxiv.org/abs/1512.04205. arXiv: 1512.04205.

[11] N. Benjamin Erichson and Carl Donovan. Randomized Low-Rank Dynamic Mode Decomposition for Motion Detection. *Computer Vision and Image Understanding*, 146:40–50, May 2016. ISSN 10773142. doi: 10.1016/j.cviu.2016.02.005. URL http://arxiv.org/abs/1512.03526. arXiv: 1512.03526.

[12] Jacob Grosek and J. Nathan Kutz. Dynamic Mode Decomposition for Real-Time Background/Foreground Separation in Video. *arXiv:1404.7592 [cs]*, April 2014. URL http://arxiv.org/abs/1404.7592. arXiv: 1404.7592.

[13] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3D Deep Learning with PyTorch3D. *arXiv:2007.08501 [cs]*, July 2020. URL http://arxiv.org/abs/2007.08501. arXiv: 2007.08501.

[14] Yann Labbé, Justin Carpentier, Mathieu Aubry, and Josef Sivic. CosyPose: Consistent multi-view multi-object 6D pose estimation. *arXiv:2008.08465 [cs]*, August 2020. URL http://arxiv.org/abs/2008.08465. arXiv: 2008.08465.

[15] Kiru Park, Timothy Patten, and Markus Vincze. Pix2Pose: Pixel-Wise Coordinate Regression of Objects for 6D Pose Estimation. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7667–7676, October 2019. doi: 10.1109/ICCV.2019.00776. URL http://arxiv.org/abs/1908.07433. arXiv: 1908.07433.

[16] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. DPOD: 6D Pose Object Detector and Refiner. *arXiv:1902.11020 [cs]*, August 2019. URL http://arxiv.org/abs/1902.11020. arXiv: 1902.11020.

[17] Yu Xiang, Wonhui Kim, Wei Chen, Jingwei Ji, Christopher Choy, Hao Su, Roozbeh Mottaghi, Leonidas Guibas, and Silvio Savarese. ObjectNet3D: A Large Scale Database for 3D Object Recognition. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 160–176, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46484-8. doi: 10.1007/978-3-319-46484-8_10.

[18] Y. Xiang, R. Mottaghi, and S. Savarese. Beyond PASCAL: A benchmark for 3D object detection in the wild. In *IEEE Winter Conference on Applications of Computer Vision*, pages 75–82, March 2014. doi: 10.1109/WACV.2014.6836101. ISSN: 1550-5790.

[19] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, Lecture Notes in Computer Science, pages 740–755, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10602-1. doi: 10.1007/978-3-319-10602-1_48.

[20] Abhishek Kar, Shubham Tulsiani, João Carreira, and Jitendra Malik. Category-Specific Object Reconstruction from a Single Image. *arXiv:1411.6069 [cs]*, May 2015. URL http://arxiv.org/abs/1411.6069. arXiv: 1411.6069.

[21] Ching-Hang Chen and Deva Ramanan. 3D Human Pose Estimation = 2D Pose Estimation + Matching. *arXiv:1612.06524 [cs]*, April 2017. URL http://arxiv.org/abs/1612.06524. arXiv: 1612.06524.

[22] Jiajun Wu, Tianfan Xue, Joseph J. Lim, Yuandong Tian, Joshua B. Tenenbaum, Antonio Torralba, and William T. Freeman. 3D Interpreter Networks for Viewer-Centered Wireframe Modeling. *International Journal of Computer Vision*, 126 (9):1009–1026, September 2018. ISSN 0920-5691, 1573-1405. doi: 10.1007/s11263-018-1074-6. URL http://arxiv.org/abs/1804.00782. arXiv: 1804.00782.

[23] Georgios Pavlakos, Xiaowei Zhou, Aaron Chan, Konstantinos G. Derpanis, and Kostas Daniilidis. 6-DoF Object Pose from Semantic Keypoints. *arXiv:1703.04670 [cs]*, March 2017. URL http://arxiv.org/abs/1703.04670. arXiv: 1703.04670.

[24] Sida Peng, Yuan Liu, Qixing Huang, Hujun Bao, and Xiaowei Zhou. PVNet: Pixel-wise Voting Network for 6DoF Pose Estimation. *arXiv:1812.11788 [cs]*, December 2018. URL http://arxiv.org/abs/1812.11788. arXiv: 1812.11788.

[25] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. Real-Time Seamless Single Shot 6D Object Pose Prediction. *arXiv:1711.08848 [cs]*, December 2018. URL http://arxiv.org/abs/1711.08848. arXiv: 1711.08848.

[26] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. *arXiv:1711.00199 [cs]*, May 2018. URL http://arxiv.org/abs/1711.00199. arXiv: 1711.00199.

[27] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. DeepIM: Deep Iterative Matching for 6D Pose Estimation. *International Journal of Computer Vision*, 128(3):657–678, March 2020. ISSN 0920-5691, 1573-1405. doi: 10.1007/s11263-019-01250-9. URL http://arxiv.org/abs/1804.00175. arXiv: 1804.00175.

[28] Zhigang Li, Gu Wang, and Xiangyang Ji. CDPN: Coordinates-Based Disentangled Pose Network for Real-Time RGB-Based 6-DoF Object Pose Estimation. October 2019. doi: 10.1109/ICCV.2019.00777.

[29] Keunhong Park, Arsalan Mousavian, Yu Xiang, and Dieter Fox. LatentFusion: End-to-End Differentiable Reconstruction and Rendering for Unseen Object Pose Estimation. *arXiv:1912.00416 [cs]*, June 2020. URL http://arxiv.org/abs/1912.00416. arXiv: 1912.00416.

[30] Tingbo Hou, Adel Ahmadyan, Liangkai Zhang, Jianing Wei, and Matthias Grundmann. MobilePose: Real-Time Pose Estimation for Unseen Objects with Weak Shape Supervision. *arXiv:2003.03522 [cs]*, March 2020. URL http://arxiv.org/abs/2003.03522. arXiv: 2003.03522.

[31] Xingyi Zhou, Arjun Karpur, Linjie Luo, and Qixing Huang. StarMap for Category-Agnostic Keypoint and Viewpoint Estimation. *arXiv:1803.09331 [cs]*, July 2018. URL http://arxiv.org/abs/1803.09331. arXiv: 1803.09331.

[32] Alexander Grabner, Peter M. Roth, and Vincent Lepetit. 3D Pose Estimation and 3D Model Retrieval for Objects in the Wild. *arXiv:1803.11493 [cs]*, March 2018. URL http://arxiv.org/abs/1803.11493. arXiv: 1803.11493.

[33] Alexander McDermott Miller, Ramon Alvarez, and Nathan Hartman. Towards an extended model-based definition for the digital twin. *Computer-Aided Design and Applications*, 15(6):880–891, November 2018. ISSN null. doi: 10.1080/16864360.2018.1462569. URL https://doi.org/10.1080/16864360.2018.1462569. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/16864360.2018.1462569.

[34] M. Piccardi. Background subtraction techniques: a review. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, volume 4, pages 3099–3104 vol.4, October 2004. doi: 10.1109/ICSMC.2004.1400815. ISSN: 1062-922X.

[35] Ahmed Elgammal, David Harwood, and Larry Davis. Non-parametric Model for Background Subtraction. In David Vernon, editor, *Computer Vision — ECCV 2000*, Lecture Notes in Computer Science, pages 751–767, Berlin, Heidelberg, 2000. Springer. ISBN 978-3-540-45053-5. doi: 10.1007/3-540-45053-X_48.

[36] Jonathan H. Tu, Clarence W. Rowley, Dirk M. Luchtenburg, Steven L. Brunton, and J. Nathan Kutz. On Dynamic Mode Decomposition: Theory and Applications. *Journal of Computational Dynamics*, 1(2):391–421, 2014. ISSN 2158-2505. doi: 10.3934/jcd.2014.1.391. URL http://arxiv.org/abs/1312.0041. arXiv: 1312.0041.

[37] J. Kutz, Steven Brunton, Bingni Brunton, and Joshua Proctor. *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*. SIAM, November 2016. ISBN 978-1-61197-449-2.

[38] Dan Kalman. The Generalized Vandermonde Matrix. *Mathematics Magazine*, 57(1):15–21, January 1984. ISSN 0025-570X. doi: 10.1080/0025570X.1984.11977069. URL https://doi.org/10.1080/0025570X.1984.11977069. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/0025570X.1984.11977069.

[39] Emmanuel J. Candes, Xiaodong Li, Yi Ma, and John Wright. Robust Principal Component Analysis? *arXiv:0912.3599 [cs, math]*, December 2009. URL http://arxiv.org/abs/0912.3599. arXiv: 0912.3599.

[40] Thierry Bouwmans, Andrews Sobral, Sajid Javed, Soon Ki Jung, and El-Hadi Zahzah. Decomposition into Low-rank plus Additive Matrices for Background/Foreground Separation: A Review for a Comparative Evaluation with a Large-Scale Dataset. *Computer Science Review*, 23:1–71, February 2017. ISSN 15740137. doi: 10.1016/j.cosrev.2016.11.001. URL http://arxiv.org/abs/1511.01245. arXiv: 1511.01245.

[41] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning, second edition*. MIT Press, December 2018. ISBN 978-0-262-35136-2. Google-Books-ID: dWB9DwAAQBAJ.

[42] D. Michie, D. Spiegelhalter, and Charles Taylor. Machine Learning, Neural and Statistical Classification. *Technometrics*, 37, January 1999. doi: 10.2307/1269742.

[43] David A. Winkler and Tu C. Le. Performance of Deep and Shallow Neural Networks, the Universal Approximation Theorem, Activity Cliffs, and QSAR. *Molecular Informatics*, 36(1-2):1600118, 2017. ISSN 1868-1751. doi: https://doi.org/10.1002/minf.201600118. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/minf.201600118. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/minf.201600118.

[44] ROBERT Hecht-nielsen. III.3 - Theory of the Backpropagation Neural Network**Based on "nonindent" by Robert Hecht-Nielsen, which appeared in Proceedings of the International Joint Conference on Neural Networks 1, 593–611, June 1989. © 1989 IEEE. In Harry Wechsler, editor, *Neural Networks for Perception*, pages 65–93. Academic Press, January 1992. ISBN 978-0-12-741252-8. doi: 10.1016/B978-0-12-741252-8.50010-8. URL http://www.sciencedirect.com/science/article/pii/B9780127412528500108.

[45] Keiron O'Shea and Ryan Nash. An Introduction to Convolutional Neural Networks. *arXiv:1511.08458 [cs]*, December 2015. URL http://arxiv.org/abs/1511.08458. arXiv: 1511.08458.

[46] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, May 2015. ISSN 1476-4687. doi: 10.1038/nature14539. URL https://www.nature.com/articles/nature14539. Number: 7553 Publisher: Nature Publishing Group.

[47] Jeffrey Heaton. Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning: The MIT Press, 2016, 800 pp, ISBN: 0262035618. *Genetic Programming and Evolvable Machines*, 19, October 2017. doi: 10.1007/s10710-017-9314-z.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. URL http://arxiv.org/abs/1512.03385. arXiv: 1512.03385.

[49] Sihan Li, Jiantao Jiao, Yanjun Han, and Tsachy Weissman. Demystifying ResNet. *arXiv:1611.01186 [cs, stat]*, May 2017. URL http://arxiv.org/abs/1611.01186. arXiv: 1611.01186.

[50] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *arXiv:1506.02640 [cs]*, May 2016. URL http://arxiv.org/abs/1506.02640. arXiv: 1506.02640.

[51] Joseph Redmon. Darknet: Open Source Neural Networks in C, 2013. URL http://pjreddie.com/darknet.

[52] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv:1506.01497 [cs]*, January 2016. URL http://arxiv.org/abs/1506.01497. arXiv: 1506.01497.

[53] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *arXiv:1703.06870 [cs]*, January 2018. URL http://arxiv.org/abs/1703.06870. arXiv: 1703.06870.

[54] Yang Xiao and Renaud Marlet. Few-Shot Object Detection and Viewpoint Estimation for Objects in the Wild. *arXiv:2007.12107 [cs]*, July 2020. URL http://arxiv.org/abs/2007.12107. arXiv: 2007.12107.

[55] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again. *arXiv:1711.10006 [cs]*, November 2017. URL http://arxiv.org/abs/1711.10006. arXiv: 1711.10006.

[56] Shubham Tulsiani and Jitendra Malik. Viewpoints and Keypoints. *arXiv:1411.6067 [cs]*, April 2015. URL http://arxiv.org/abs/1411.6067. arXiv: 1411.6067.

[57] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004. ISSN 1573-1405. doi: 10.1023/B:VISI.0000029664.99615.94. URL https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[58] Uzair Nadeem, Mohammed Bennamoun, Roberto Togneri, and Ferdous Sohel. Unconstrained Matching of 2D and 3D Descriptors for 6-DOF Pose Estimation. *arXiv:2005.14502 [cs]*, May 2020. URL http://arxiv.org/abs/2005.14502. arXiv: 2005.14502.

[59] Mahdi Rad and Vincent Lepetit. BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth. *arXiv:1703.10896 [cs]*, March 2018. URL http://arxiv.org/abs/1703.10896. arXiv: 1703.10896.

[60] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv:1912.01703 [cs, stat]*, December 2019. URL http://arxiv.org/abs/1912.01703. arXiv: 1912.01703.

[61] Gary Bradski. The openCV library. *Dr. Dobb's Journal of Software Tools*, 25, January 2000.

[62] Magnus Själander, Magnus Jahre, Gunnar Tufte, and Nico Reissmann. EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure. *arXiv:1912.05848 [cs]*, December 2020. URL http://arxiv.org/abs/1912.05848. arXiv: 1912.05848.

[63] Warren Gay. *Raspberry Pi Hardware Reference*. Apress, January 2014. ISBN 978-1-4842-0800-7. doi: 10.1007/978-1-4842-0799-4.

[64] Glenn Jocher, Alex Stoken, Jirka Borovec, and Liu Changyu. ultralytics/yolov5: v4.0 - nn.SiLU() activations, Weights & Biases logging, PyTorch Hub integration, January 2021. URL https://zenodo.org/record/4418161.

[65] Yang Xiao, Xuchong Qiu, Pierre-Alain Langlois, Mathieu Aubry, and Renaud Marlet. Pose from Shape: Deep Pose Estimation for Arbitrary 3D Objects. *arXiv:1906.05105 [cs]*, August 2019. URL http://arxiv.org/abs/1906.05105. arXiv: 1906.05105.

[66] Blender - a 3D modelling and rendering package, 2018. URL http://www.blender.org.

[67] Enric Corona, Kaustav Kundu, and Sanja Fidler. Pose Estimation for Objects with Rotational Symmetry. *arXiv:1810.05780 [cs]*, October 2018. URL http://arxiv.org/abs/1810.05780. arXiv: 1810.05780.

[68] Tomas Hodan, Daniel Barath, and Jiri Matas. EPOS: Estimating 6D Pose of Objects with Symmetries. *arXiv:2004.00605 [cs, eess]*, April 2020. URL `http://arxiv.org/abs/2004.00605`. arXiv: 2004.00605.

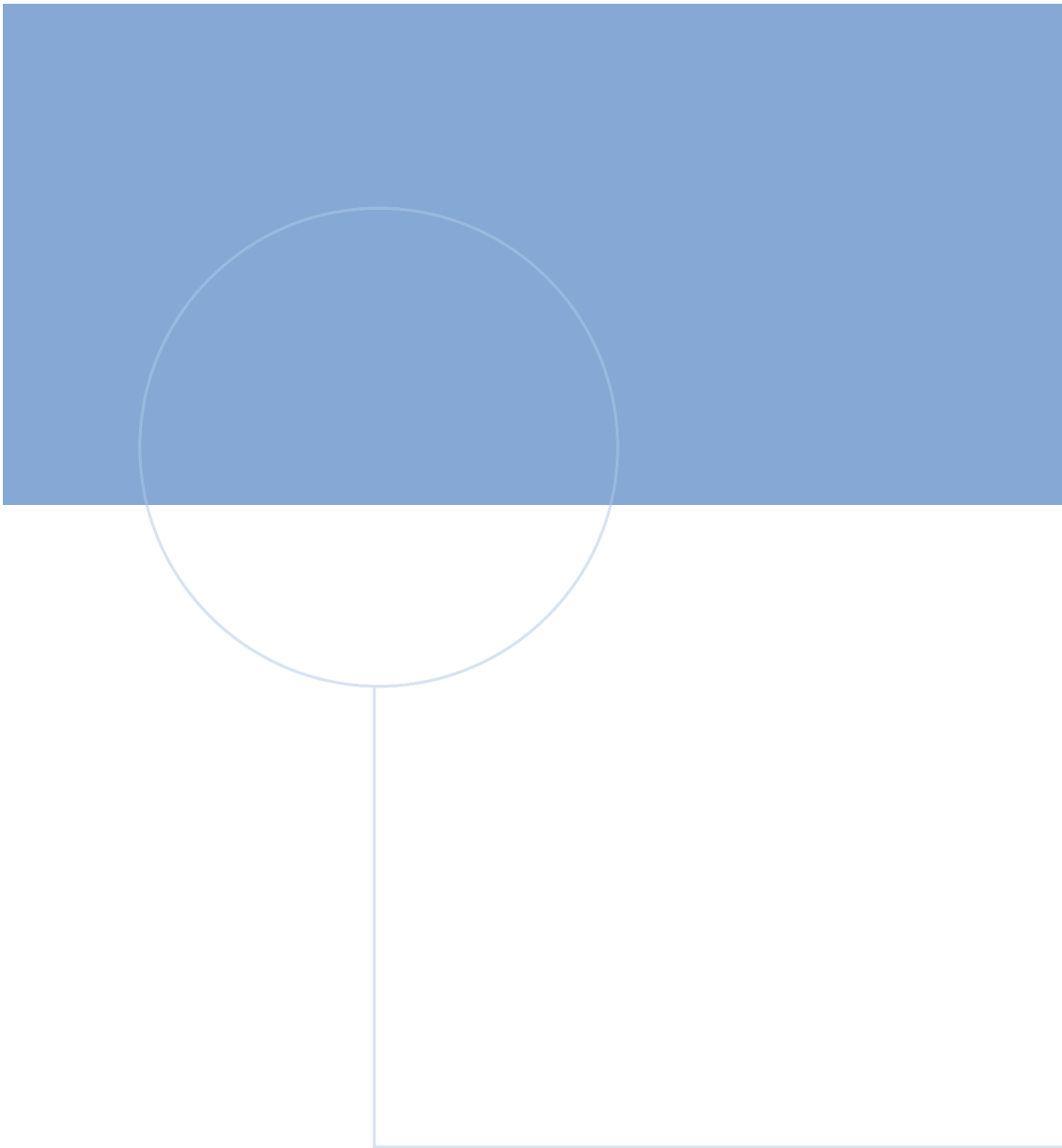
[69] P. Ammirato, J. Tremblay, M. Liu, A. C. Berg, and D. Fox. SymGAN: Orientation Estimation without Annotation for Symmetric Objects. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1657–1666, March 2020. doi: 10.1109/WACV45572.2020.9093450. ISSN: 2642-9381.

[70] Enric Corona, Kaustav Kundu, and Sanja Fidler. Pose Estimation for Objects with Rotational Symmetry. *arXiv:1810.05780 [cs]*, October 2018. URL `http://arxiv.org/abs/1810.05780`. arXiv: 1810.05780.

[71] Sören Sonnenburg, Mikio Braun, Cheng Soon Ong, Samy Bengio, Leon Bottou, Geoffrey Holmes, Yann Lecun, Klaus-Robert Müller, Fernando Pereira, Carl Rasmussen, Gunnar Rätsch, Bernhard Schölkopf, Alexander Smola, Pascal Vincent, Jason Weston, and Robert Williamson. The Need for Open Source Software in Machine Learning. *J. Mach. Learn. Res.*, 8:2443–2466, January 2007.

[72] A. Adadi and M. Berrada. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2870052. Conference Name: IEEE Access.

[73] J. M. Zhang, M. Harman, L. Ma, and Y. Liu. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Transactions on Software Engineering*, pages 1–1, 2020. ISSN 1939-3520. doi: 10.1109/TSE.2019.2962027. Conference Name: IEEE Transactions on Software Engineering.



NTNU

Norwegian University of
Science and Technology