Master's thesis

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

Julia Maria Graham

Geometric change detection in the context of Digital Twin, leveraging Dynamic Mode Decomposition, Object Detection and innovations in 3D technology

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

NTNU
Norwegian University of
Science and Technology

Julia Maria Graham

Geometric change detection in the context of Digital Twin, leveraging Dynamic Mode Decomposition, Object Detection and innovations in 3D technology

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

Preface

The research carried out in this report is submitted as my Master's degree that finalizes the integrated five-year engineering program Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU).

During the past two years of my studies, my interest for Data modeling and Machine Learning have emerged as a result of following courses on these topics. I was therefore drawn to writing a master thesis that would allow for cultivating these interests, with the possibility of examining novel approaches within their scope. This work tackles both well-known and established Machine Learning methods, and examines innovations based on a purely data-driven technique, and a non-Machine Learning method which is respectively Dynamic Mode Decomposition, and Sparse Representation Classification. These models, among others, are further used in a framework for performing Change Detection in the context of a Digital Twin, which is the main focus of this work. To facilitate the change detection application, an experimental setup is built.

I want to express my deepest appreciation towards my supervisor, Adil Rasheed, for his continuous assistance, valuable insights and directions, and for motivating me along this journey.

Furthermore, I want to thank Glenn Angell for his invaluable assistance in helping me build and design the experimental setup, and for meeting my requests with patience and efficiency.

Lastly, I want to recognize the HPC group at NTNU for providing me with the necessary computing power needed to conduct my experiments.

Julia Maria Graham

Trondheim, 20th June, 2021

Contents

List of Figures

List of Tables

Abstract

With the blooming era of digitalization, Digital Twin technology has emerged as the prevailing technology for industry. Through simulating, predicting and optimizing physical manufacturing systems and processes, Digital Twins liberate industry data providing insightful information while embodying the potential for innovation. Moreover, the enabling factor for an updated and accurate Digital representation are large quantities of sensor data that requires sufficient storage and transmission bandwidth.

To circumvent these expensive requirements, the following work implements a cost-effective approach for updating the digital replica of the physical environment, without the need for excessive storage, computation power and bandwidth. The core idea is to have a camera monitor a physical scene in real-time where selected objects reside. Upon detecting a change in the scenery, more specifically the movement of an object, this movement is detected, the object is localized and the new orientation of the object is estimated. This estimated pose can further be used to update the digital replica by reconstructing the new scene on demand. The enabling technologies are Dynamic Mode Decomposition for motion detection, Yolo for object detection and 3D machine learning for pose estimation. An alternative non-Deep Learning method for performing image recognition in the proposed approach is also investigated.

Furthermore, within the realm of 3D technology it is recognized that 3D modeling and Computer Aided Design are powerful and accessible methods that provide accurate representations of real objects. Thus, advancements of 3D modeling technology begs the question of how it can be leveraged for use in the real world. This thought is examined in the following work as well, by programmatically acquire synthetic data for training various Deep learning and non-Deep Learning models, and further examine their performance in the real world.

Sammendrag

I takt med den økende digitaliseringen har Digital Tvilling-teknologi vist seg å bli en betydningsfull teknologi for industri. Ved simulering, prediksjon og optimering av fysiske produksjonssystemer og prosesser, frigjør Digitale Tvillinger industridata, og bidrar dermed med innsiktsfull informasjon som også skaper rom for innovasjon. Den muliggjørende faktoren som sørger for en stadig oppdatert og korrekt Digital modell er store mengder sensordata, hvilket medfører kravene om tilstrekkelig lagringskapasitet og båndbredde over nettverket.
For å møte disse kostbare forbeholdene, vil det følgende arbeidet presentere en kostnadseffektiv metode for å oppdatere den digitale kopien av det fysiske miljøet uten behovet for uforholdsmessige mengder lagring, regnekraft eller båndbredde. Fremgangsmåten er å ha et kamera som overvåker en fysisk scene i sanntid hvor utvalgte objekter oppholder seg. I øyeblikekt et objekt beveger seg, vil bevegelsen bli fanget opp av kameraet, objektet bli lokalisert og gjenkjent i bildet, og den nye orienteringen av objektet blir estimert. Den estimerte endringen i orientering kan dermed bli brukt for å oppdatere den digitale tvillingen ved å rekonstruere den endrede scenen på etterspørsel. De muliggjørende teknologiene som blir brukt i denne fremgangsmåten er Dynamic Mode Decomposition for detektering av bevegelse, Yolo for objektgjennkjenning og 3D Maskinlæring for å estimere orientering. Videre vil også en alternativ metode for å utføre objektgjennkjenning som en del av det foreslåtte rammeverket, og som ikke er basert på dyp læring, bli undersøkt.
Innen landskapet for 3D-teknologi, er det kjent at 3D-modelering og Computer Aided Design er kraftige og tilgjengelige metoder som muliggjør presise representasjoner av fysiske objekter. Fremgang innen 3D-modelering åpner dermed for spørsmålet om hvordan denne teknologien kan overføres til den fysiske verdenen vi lever i. Denne tanken blir undersøkt i det følgende arbeidet, ved å lage automatiserte skript som samler inn et utvalg syntetisk data til bruk som treningsdata for ulike dyp læring- og ikke dyp læringsmodeller. Deretter blir modellene satt til å testes i den virkelige verden, og deres prestasjon undersøkt.

Chapter 1

Introduction

The concept of Digital Twin (DT) is rapidly transforming the landscape of industries (Tao et al. (2018)). In order to accurately mirror the physical environment, frequent updates are required. This implies processing large streams of data, resulting in expensive storage and bandwidth demands. Through motion detection, object detection and 3D machine learning, a pipeline for performing Change Detection to update the digital scene in a cost-effective way on demand is facilitated. Secondly, research on the potential value of using synthetic data for training, while testing the trained model in the real world is also undertaken. Lastly, investigations on non-Deep Learning methods such as Dynamic Mode Decomposition and Sparse Representation based Classification are exhibited.

1.1 Motivation

The proposed change detection method is based on the approach presented in Sundby et al. (2021). The following work aims to perform a more rigorous implementation, while examining some other directions within the scope of the change detection framework.

In its essence, a Digital Twin is a virtual model empowered by big data and the governing physics of a process, product or service (Wanasinghe et al. (2020)). Furthermore, to obtain an accurate digital representation of an asset, high-quality geometrical models describing its physics is required. This is typically enabled with Computer Aided Design (CAD) models, which is software for creating accurate 3D digital representations of geometry data consisting of parametrized object surfaces (Dugelay et al. (2008)). Further, a *descriptive* digital twin is comprised of CAD models that models the static and dynamic evolution of a physical asset (Zheng et al. (2019)).

Raw data collected in real-time through sensors and data transmission technologies can by mined and transformed into valuable information with the use of powerful storage, computing power of cloud computing, and big data analysis models and algorithms (Tao et al. (2019)). This continuous stream of new information is required to update the Digital Twin such that the state of the physical asset is close to real-time mirrored in its digital replica. Certainly, this will result in a large volume of structured, semi-structured and unstructured data (Qi and Tao (2018)). Central challenges DT technologies face are consequently related to storage as well as bandwidth requirements from the extensive transmission of raw sensor data (Rasheed et al. (2019)).

In an attempt to face these challenges, an approach for detecting changes in a physical environment for the purpose of updating its digital scene on demand with the detected changes is presented in this report. The approach is proposed as a lightweight alternative to continuously storing and processing sensor information from the physical scene. It is designed with emphasis towards minimizing computational and storage demands required for continuous and real-time updates of a Digital Twin.

The proposed change detection method is composed of three modules. Firstly, motion detection is performed by using Dynamic Mode Decomposition (DMD) for background subtraction to extract the moving foreground. Secondly, object detection to localize accurate bounding box estimates of the moving objects is facilitated by You Only Look Once (Yolo). Lastly, the orientation of the localized object is estimated through a Pose Estimation network. Specifically, this is a Convolutional Neural Network (CNN) that performs pose estimation using a particular feature extraction method.

Fundamental for the object detection module in the proposed change detection approach is the task of image recognition. The most powerful class of models that have emerged as the standard for performing Computer Vision and image classification are deep learning models based on Convolutional Neural Networks (CNNs) (LeCun et al. (2015)). Yet, in exchange for their high performance are high computational demands requiring computing power from high-end Graphics Processing Units (GPUs) (He and Sun (2015)). Moreover, CNNs are often described as "Black-Box"-models (Tzeng and Ma (2005)), in that the underlying decisions that lead to some output given some input is completely hidden from the observer. This trait deem them unfit for safety-critical applications such as autonomous driving where it is essential that the decisions made by the software are predictable and adhere to the governing ethics consensus. It has been shown that non-deep learning image recognition inspired by techniques from Compressed Sensing (CS) and dictionary learning, namely Sparse Representation based Classification, yield state of the art performance in the task of Face Recognition (Wright et al. (2008)). Feature-based methods such as Principal Component Analysis (PCA) is another non-deep learning method that has proven to be successful in image classification (Bajwa et al. (2009)). These methods are beneficial in terms of being less computationally heavy. SRC also embody greater transparency as the classifications stems from a simple optimization problem that is addressed in subsection 2.5.2.

Furthermore, a prerequisite for building a powerful image classifier using deep learning in particular is the data volume available for training (Shorten and Khoshgoftaar (2019)). Generating a high quality dataset of sufficient size may however be costly, time consuming and challenging depending on the data source. Methods such as data augmentation have been proposed to meet this requirement (Perez and Wang (2017); Mikołajczyk and Grochowski (2018)), yet more research on this field is sought.

Therefore, an investigation on whether we can leverage *synthetic data* for training image classifiers such as CNN and SRC with the aim of using the trained model in the real world, is conducted in this report. This could alleviate the challenges related to large-scale high quality data generation. Moreover, the requirement of training data volume is not necessarily present for SRC as it has even been shown to achieve high performance in addition to generalising well in the task of face recognition with a single training image per object category (Deng et al. (2012*a*)). With these remarks in mind, SRC is also revised as a competing candidate against CNNs in the task of image recognition.

1.2 Background and related work

Motion detection and DMD: Application areas in motion detection such as video surveillance, traffic analysis and robot navigation, demonstrates it to be a key topic in the field of computer vision. Traditional approaches of motion detection is generally comprised of the categories Background Subtraction, Frame differencing, Temporal Differencing and Optical Flow (Kulchandani and Dangarwala (2015)). Moreover, the leading and most reliable approach among these is considered to be Background Subtraction. The idea is based on estimating a model of the background to further subtract it from the raw video frame. Based on the pixel intensities of the resulting frame, moving foreground objects can be isolated (Shaikh et al. (2014)). The use of DMD for background subtraction and its variants is successfully proposed in Grosek and Kutz (2014), Kutz et al. (2017), Erichson et al. (2019), Kutz et al. (2016). In Kutz et al. (2017), methods for applying DMD to streaming data, thus enabling real-time background subtraction, along with measures for optimizing the DMD algorithm is proposed. A variant leveraging the storage optimization enabled by sparse sampling and Compressed Sensing is proposed in Erichson et al. (2019), namely compressed DMD or cDMD. Furthermore, an approach that enables detection of moving objects at different speeds, namely Multi-Resolution DMD or mrDMD, is proposed in Kutz et al. (2016), which is related to Wavelet theory.

Object detection: The task of object detection can be posed as both localizing object instances as well as assigning their true class from images (Liu et al. (2020)). Using SRC for image classification, and specifically achieving state-of-the-art Face Recognition was first presented in Wright et al. (2008). The proposed face recognition system is inspired by Compressed Sensing theory and sparse representation dictionary learning, leading to the Sparse Representation Classifier (SRC). It is motivated by the notion of natural sparsity in the face recognition problem, in trying to classify a single subject from a large database of faces. Thus, only a few training samples from a single category is needed in order to describe a query image. This implies that a sparse representation of a query image can be sought by posing the problem as an ℓ_1-minimisation problem.

Attempts at improving the SRC algorithm is done with the Kernel Sparse Representation Classifier (KSRC) proposed in Yin et al. (2012). A kernel is utilized for mapping features into a high-dimensional feature space prior to classification, which enable the extraction of non-linear features. The aim of KSRC is to find an appropriate kernel that projects the linearly inseparable samples in the original feature space into a linearly separable higher dimensional feature space. A test sample can consequently be more accurately described as a linear combination of training samples from the same class.

Yet another improvement of SRC that advances the KSRC algorithm is the one proposed in Xu et al. (2013). In SRC, all training samples are weighted equally as a linear combination to represent a new test sample. Therefore, it does not consider correlation between training samples from the same class. In kernel based weighted group sparse classifier (KWGSC), the Kernel trick is not only used for mapping the original feature space into a high dimensional feature space, but also as a measure to select or weight members of each group. The weight thus reflects the degree of importance of training samples in different groups.

With the recent years advent of powerful Deep Learning methods, the leading object detection frameworks are based on the following; RCNN (Girshick et al. (2014)), Fast RCNN (Girshick (2015)), Faster RCNN (Ren et al. (2016)), YOLO (Redmon et al. (2016)) and Single shot detector (SSD) (Liu et al. (2016)). Common for these frameworks are their back-bone architectures, which are powered by CNNs (Girshick et al. (2014)). CNNs facilitate the learning of complex features from images and seem to excel at producing the most relevant feature maps for images that both help in object localization as well as recognition (Wu et al. (2020)). Although this may be a computationally heavy process, recent innovations such as Yolov5 is a lightweight and fast Yolo model variant that is said to obtain an accuracy on par with its predecessor Yolov4 (Bochkovskiy et al. (2020); Jocher et al. (2020)). Additionally, the Yolo model family are shown to best qualify for real-time applications, as they only need to process an image once, hence the name "You Only Look Once" (Redmon et al. (2016)). Thus, with both speed and size in mind, the model choice for object detection in the following presented framework is Yolov5.

Pose estimation: When the object-shape is known, approaches for estimating the pose of a 3D object can be broadly categorized into feature-matching and template-matching methods (Xiao et al. (2019*a*)). Feature-matching methods commonly seek to extract local features of an image and match them to the a 3D model of the depicted object. Matching on the pixel-level rather than using features is a variant of this approach (Park et al. (2019)). A Perspective-n-Point (PnP) algorithm for extracting the camera 6D viewpoint comprised of both orientation and translation coordinates can further be applied to the estimated 2D-3D correspondences. These methods perform poorly on texture-less and low resolution images as the task of matching 2D-3D correspondences become increasingly difficult (Xiang et al. (2017)). Template-based methods have been developed to meet these challenges by matching the target object in an image to a similar template of the object (Lee and Hong (2012)). However, these methods are not robust when faced with occluded objects or objects in cluttered scenes. Recently, deep learning models, specifically CNNs, have been successfully employed for the task of Pose Estimation (Xiang et al. (2017); Peng et al. (2019); Li et al. (2018)). They have shown to be robust to the mentioned challenges while being able to perform Pose Estimation from a single RGB (Red Green Blue)-image. Common for these deep models are that they are category-specific in that they may only estimate poses of object categories known at the time of training. In the context of Digital Twins, the physical environment may alternate with the introduction of novel objects. The ability to generalize in the mapping from the physical to the virtual environment is a necessity if complete autonomy is to be realized for an autonomous Digital Twin. A method of performing Pose Estimation that is not restricted to specific categories is therefore deemed relevant for the Change Detection framework. Moreover, a recent deep learning method for performing Pose Estimation with the possible extension to novel objects is presented in Xiao et al. (2019*a*). With these remarks in mind, it is therefore deemed the appropriate model for performing Pose Estimation in the proposed framework.

Empowered by these technologies, a full change detection framework can come alive.

1.3 Research Objectives and research questions

1.3.1 Objectives

Primary Objective: To develop a cost-effective approach to detect geometrical changes in descriptive digital twins.
Secondary Objectives:

- Develop a virtual environment for synthesizing data for training and an experimental set up for evaluating the effectiveness of the change detection approach proposed in this work.
- Evaluate the potential of utilizing synthetic data for training various models to be later used in the real world.
- Evaluate the potential of algorithms with mathematically sound foundation like dynamic mode decomposition for motion detection, and compressed sensing as an alternative to Deep Learning methods for image classification.

1.3.2 Research Questions

To the best of our knowledge there is currently no published work on the following approach for conducting change detection in the context of Digital Twins. To this end, the guiding questions governing the research can be stated as:

- With pedagogical purpose in mind, what kind of experimental setup can be built for the purpose of testing a cost-effective change detection approach in a Digital Twin ?
- Can synthetic data programmatically acquired in a virtual environment be used for training image classifiers so that they can be used in the real world with confidence ?
- How effective are DMD for motion detection, and SRC for image classification in comparison to Deep Learning ?

1.4 Outline of Report

The thesis is comprised of the following sections and content: chapter 2 presents the theory which comprise the foundation for the technical methods used; chapter 3 dissect the concrete methods and implementation details as well as introducing the two experimental setups considered for the for the change detection application; section 3.1 presents more specifically how synthetic data acquisition is performed, while chapter 4 presents the results which are evaluated and discussed, and the thesis is concluded in chapter 5.

Chapter 2

Theory

2.1 Notation

The following practises are applied in this report:

- **Vectors and Matrices** are marked in bold. Matrices are capitalized ($\mathbf{\Phi}$, $\mathbf{A}$, $\mathbf{W}$), while vectors are in lower-case ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{c}$)

- A vector, $\mathbf{x}$, is given as a column vector, and its transpose, $\mathbf{x}^T$ denotes a row-vector

- **Scalar values** are written in lower-case letters with no formatting (a, b, c)

- The symbols $\mathbf{\Phi}$, $\mathbf{\Theta}$ and $\mathbf{\Psi}$, denotes matrices in the theory sections for DMD and Compressed Sensing in section 2.3 and section 2.5, and are not to be confused with the same lower-case scalar symbols denoting the Euler angles (ϕ, θ, ψ) in subsection 2.7.1. From this section and throughout the report, these lower-case symbols will refer to the Euler angles as defined in this section

2.2 Geometric modeling

Geometric modeling is the mathematical representation of an object's geometry and shape. The modeling of a geometric shape begins with outlining its surfaces. Surfaces can be mathematically described using curves that are essentially analytic functions or a set of points. These curves are what will ultimately result in a visual representation of a 3D object. Geometric modeling is a tool of great importance in mechanical engineering for visualising mechanical parts, compute mass properties, create programs to drive NC machine tools to cut out shapes from materials, etc. It is also used in architecture, geology and medical image processing (Gallier and Gallier (2000)).

Computer Aided Modeling or CAD is computer geometric modeling that uses software to store the data of the geometric properties of an object, thereby giving a digital 3D representation of the object.

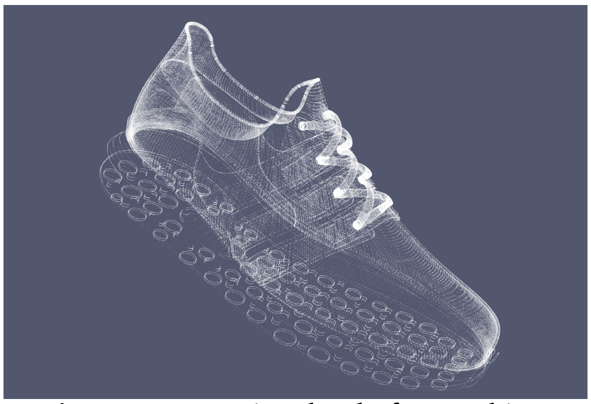


Figure 2.2.1: Point cloud of a 3D object

2.2.1 CAD Modeling

Computer aided design (CAD) is a powerful tool for creating 2D or 3D graphical models of objects. It consists of hardware and software to assist in design tasks such as creation, modification, analysis or optimisation of a design (Groover and Zimmers (1983)). CAD software makes it possible to build a model in an imaginary space, enabling visualization where design choices like size, color or material can be altered. The CAD model of an object can further be used for 3D printing, which is an innovation that enables the manufacturing of a synthetic object residing in the virtual world to take shape as a physical object in the real world Berman (2012).

To produce a CAD model of a physical object, point cloud data of the object can be generated. This is collected through laser scanning techniques that sample the surface geometry of the 3D object Hattab and Taubin (2015). Point clouds are datasets of X, Y and Z-geometric coordinates that represents the object surface in space. Each point in the cloud requires a single laser scan measurement. All points stitched together in a cloud will then form a geometric shape of either an object or a scene. This model can further be used to create a CAD model of the object by creating meshes that ultimately form the object surface.

3D file formats: STL, OBJ

A 3D file format stores information that can be processed by 3D software. The two most common file formats to use for 3D printing is STL and OBJ. STL files encodes the surface of a 3D object into a triangular mesh. For higher resolution, smaller triangles are used. The information contained in an STL file is restricted to that of the object shape and size. In order to store information like color and texture, another format must be used.

OBJ files on the other hand are equipped with more flexibility. They store additional geometry information, texture and the original mesh the model was created with. The surface encoding is not restricted to triangles as with STL, but can also use polygons or hexagons. This results in is a smoother mesh that simulates the original surface better. Yet, they are more complex and difficult to work with than STL, which is why STL remains the most popular file format overall (Iancu (2018)).

2.3 Dynamic Mode Decomposition

The following theory is partially or fully retrieved from the author's preproject that also utilized DMD for motion detection.

Dynamic Mode Decomposition (DMD) is an equation-free method that is capable of retrieving intrinsic behaviour in data, even when the underlying dynamics are nonlinear (Tu et al. (2013)). It is a purely data-driven technique, which is proving to be more and more important in the arising and existing age of Big Data.

DMD decomposes time series data into spatiotemporal coherent structures by approximating the dynamics to a linear system that describes how it evolves in time.

The linear operator, sometimes also referred to as the Koopman operator (Nathan Kutz et al. (2017)), that describes the data from one time step to the next is defined as follows

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t \tag{2.3.1}$$

Consider a set of sampled snapshots from the time series data. Each snapshot is vectored and structured as a column vector with dimensions $n \times 1$ in the following two matrices

$$\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_{m-1}\}, \quad \mathbf{X}' = \{\mathbf{x}_2, ..., \mathbf{x}_m\} \tag{2.3.2}$$

Where $\mathbf{X}'$ is the $\mathbf{X}$-matrix shifted one time step ahead, each of dimension $n \times (m-1)$.

Relating each snapshot to the next, Equation 2.3.1 can be rewritten more compactly as

$$\mathbf{X}^{'} = \mathbf{A}\mathbf{X} \tag{2.3.3}$$

The objective of DMD is to find an estimate of the linear operator $\mathbf{A}$ and obtain its leading eigenvectors and eigenvalues. This will result, respectively, in the modes and frequencies that describes the dynamics.

Computing the leading DMD modes of the linear operator proceeds as follows

Algorithm 1: Standard DMD

1. Structure data vectors into matrices $\mathbf{X}$ and $\mathbf{X}'$ as described in Equation 2.3.2

2. Compute the SVD of $\mathbf{X}$

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{2.3.4}$$

   Where $\mathbf{U}$ and $\mathbf{V}$ are square unitary matrices of sizes $n \times n$ and $m \times m$ respectively, and $\mathbf{U}\mathbf{U}^* = \mathbf{V}\mathbf{V}^* = \mathbf{I}$.

3. To reduce the order of the system, the following matrix is defined

$$\tilde{\mathbf{A}} = \mathbf{U}^*\mathbf{A}\mathbf{U} = \mathbf{U}^*\mathbf{X}'\mathbf{V}\mathbf{\Sigma}^{-1} \tag{2.3.5}$$

   Where $\tilde{\mathbf{A}}$ is projected onto the r leading modes of $\mathbf{U}$ as a result of a truncated SVD.

4. Compute the eigendecomposition of $\tilde{\mathbf{A}}$:

$$\tilde{\mathbf{A}}\mathbf{W} = \mathbf{W}\mathbf{\Lambda} \tag{2.3.6}$$

5. Which ultimately leads to the DMD *modes* $\mathbf{\Psi}$, where each column of $\mathbf{\Psi}$ represents a single mode of the solution.

$$\mathbf{\Psi} = \mathbf{X}'\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{W} \tag{2.3.7}$$

The predicted state at time points $t \in 1, ..., k$ is then expressed as a linear combination of the identified modes.

$$\tilde{\mathbf{x}}_{t+1} = \sum_i \lambda_i^t \boldsymbol{\psi}_i b_{i,0} = \mathbf{\Psi}\mathbf{\Lambda}^t \mathbf{b}_0 \tag{2.3.8}$$

The initial amplitudes, $b_{i,0}$, of the modes are obtained by setting $t = 0$ and solving for $\mathbf{b}_0$ in $\mathbf{\Psi}\mathbf{b}_0 = \mathbf{x}_1$.

Note that each DMD mode $\boldsymbol{\psi}_i$ is a vector that contains the spatial information of the decomposition, while each corresponding eigenvalue λ_i^t along the diagonal of $\mathbf{\Lambda}^t$ describes the time evolution of the respective mode. The part of the video frame, i.e. modes, that changes slowly in time must therefore have a stationary associated eigenvalue, i.e. $\mid \lambda \mid \approx 1$. Relating this to the frequency domain, we have that $\omega_i = \frac{ln(\lambda_i)}{\delta t}$. Thus, the slowly varying dynamics will have frequency content $\mid \omega_i \mid \approx 0$, i.e slowly varying energy content in time.

2.3.1 DMD for streaming data and background subtraction

Although Standard DMD is primarily viewed as a post-processing tool that requires a large amount of data, recent innovations of DMD for streaming and online-data have emerged. Among them are the two methods of Streaming DMD and Compressed DMD proposed in Nathan Kutz et al. (2017).

The computational cost of performing DMD is dominated by the expensive SVD calculation at each iteration. The Streaming DMD method utilises a less costly method of computing the SVD decomposition, namely *the method of snapshots*. This method is derived from performing the eigenvalue decomposition of the matrix product $\mathbf{X^T X}$.

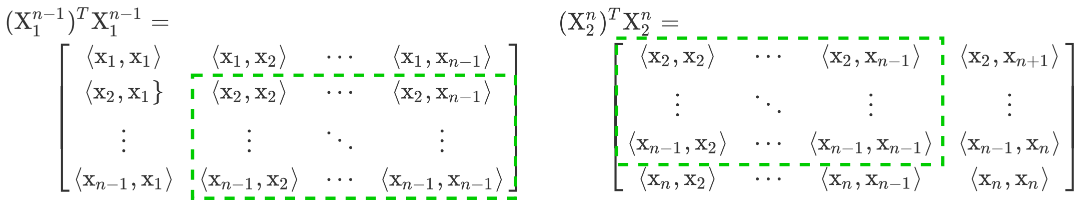
$$\mathbf{X^T X V} = \mathbf{V}\mathbf{\Sigma}^2 \tag{2.3.9}$$

This result can be used to obtain the $\mathbf{U}$-matrix

$$\mathbf{U} = \mathbf{X V}\mathbf{\Sigma}^{-1} \tag{2.3.10}$$

Now consider performing Standard DMD on an incoming data stream using the method of snapshots for computing the SVD at each iteration. As more data become available, there will be a shift of the $\mathbf{X}$ matrix as columns are appended to the right of the data matrix $\mathbf{X}$ with new sampled snapshots, and columns to the left with old data are discarded. This will result in repeated SVD computations where all the overlapping columns from one time step to the next yield redundant inner product computations ($\mathbf{X^T X}$). This is illustrated in Figure 2.3.1.

The Streaming DMD algorithm utilises this fact to only update the calculation of the last column appended to the data matrix $\mathbf{X}$ at the next time step. Thus reducing the computational complexity from $\mathcal{O}(n^2 m)$ to $\mathcal{O}(nm)$ (Nathan Kutz et al. (2017)),



$$(\mathrm{X}_1^{n-1})^T\mathrm{X}_1^{n-1} = \begin{bmatrix} \langle \mathrm{x}_1, \mathrm{x}_1\rangle & \langle \mathrm{x}_1, \mathrm{x}_2\rangle & \cdots & \langle \mathrm{x}_1, \mathrm{x}_{n-1}\rangle \\ \langle \mathrm{x}_2, \mathrm{x}_1\} & \langle \mathrm{x}_2, \mathrm{x}_2\rangle & \cdots & \langle \mathrm{x}_2, \mathrm{x}_{n-1}\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \mathrm{x}_{n-1}, \mathrm{x}_1\rangle & \langle \mathrm{x}_{n-1}, \mathrm{x}_2\rangle & \cdots & \langle \mathrm{x}_{n-1}, \mathrm{x}_{n-1}\rangle \end{bmatrix}$$

$$(\mathrm{X}_2^{n})^T\mathrm{X}_2^{n} = \begin{bmatrix} \langle \mathrm{x}_2, \mathrm{x}_2\rangle & \cdots & \langle \mathrm{x}_2, \mathrm{x}_{n-1}\rangle & \langle \mathrm{x}_2, \mathrm{x}_{n+1}\rangle \\ \vdots & \ddots & \vdots & \vdots \\ \langle \mathrm{x}_{n-1}, \mathrm{x}_2\rangle & \cdots & \langle \mathrm{x}_{n-1}, \mathrm{x}_{n-1}\rangle & \langle \mathrm{x}_{n-1}, \mathrm{x}_{n}\rangle \\ \langle \mathrm{x}_{n}, \mathrm{x}_2\rangle & \cdots & \langle \mathrm{x}_{n}, \mathrm{x}_{n-1}\rangle & \langle \mathrm{x}_{n}, \mathrm{x}_{n}\rangle \end{bmatrix}$$

Figure 2.3.1: Re-usable inner products from one time step to the next outlined in green

where m are the number of snapshots or video frames considered and n is the pixel dimension of the flattened frames.

The paper Zhang et al. (2019) proposes another method of dealing with online streaming data and DMD. It is an intuitive and simple approach called *Windowed DMD*, which in simple terms performs DMD repeatedly on a sliding window. The window is of equal size and enables the incorporation of new data and the discarding of old data for each iteration as new data become available. Note that this is simply the Streaming DMD method, without the optimisation of reusing the redundant inner products.

Another remark is that both methods (Streaming DMD and Windowed DMD) yield a better time resolution of the dynamics present in the video, rather than performing DMD once on the entire batch of video snapshots.

A relevant application of DMD is background modeling of video streams. As mentioned, DMD can be used to separate out the slowly varying modes related to slowly varying dynamics in the data. Based on the assumption that the nonlinear dynamics is a superposition of a low-dimensional component and a sparse component, Robust Principal Component Analysis (RPCA) have been shown to successfully separate the data $\mathbf{X}$ (Candès et al. (2011)).

$$\mathbf{X} = \mathbf{L} + \mathbf{S} \tag{2.3.11}$$

where $\mathbf{L}$ is the low-rank structure and $\mathbf{S}$ is sparse.

In terms of DMD, this separation can be based on the frequency contents of the modes. Considering again Equation 2.3.8 in terms of frequencies

$$\hat{\mathbf{x}}(t) = \sum_{i}^{r} exp(\omega_i t)\boldsymbol{\psi}_i b_{i,0} \tag{2.3.12}$$

Slowly varying video content, i.e low-rank features, will be related to small values of ω_i. Selecting a threshold ϵ, the separation of the stationary or nearly stationary dynamics, i.e the video background, with the time-varying foreground can be written as

$$\mathbf{L} \approx \underbrace{\sum_{|\omega_i| \le \epsilon} exp(\omega_i t) \boldsymbol{\psi}_i b_{i,0}}_{background}, \quad \mathbf{S} \approx \underbrace{\sum_{|\omega_i| > \epsilon} exp(\omega_i t) \boldsymbol{\psi}_i b_{i,0}}_{foreground} \tag{2.3.13}$$

The foreground can thus be estimated by subtracting the raw video frame with the reconstructed background, $\mathbf{L}$, from the DMD mode separation in Equation 2.3.13.

$$\hat{\mathrm{S}} = \mathbf{X}_{raw} - \mathbf{L} \tag{2.3.14}$$

where $\hat{\mathrm{S}}$ denotes the estimated foreground. This result can thus be used for localising moving objects against a stationary or slowly moving background Erichson et al. (2019).

Thresholding

To evaluate the accuracy of the detected pixels, a foreground mask based on thresholding the difference between the true raw frame and the reconstructed background can be computed. The Euclidean distance can then be used to transform the problem to a binary classification problem (Wang et al. (2014), Erichson et al. (2019)) as follows.

$$\chi_t(j) = \begin{cases} 1 & \textit{if } \|x_{j,t} - \hat{x}_j\| > \tau \\ 0 & \textit{otherwise} \end{cases}$$

where $x_{j,t}$ is the jth pixel of the tth video frame, and $\hat{x}_j$ is the corresponding pixel of the estimated background. The foreground pixels, i.e. pixels classified as 1 based on the threshold, can then be compared with ground truth pixel values to evaluate the localisation performance of the method.

Figure 2.3.2 illustrates background subtraction using DMD with thresholding.

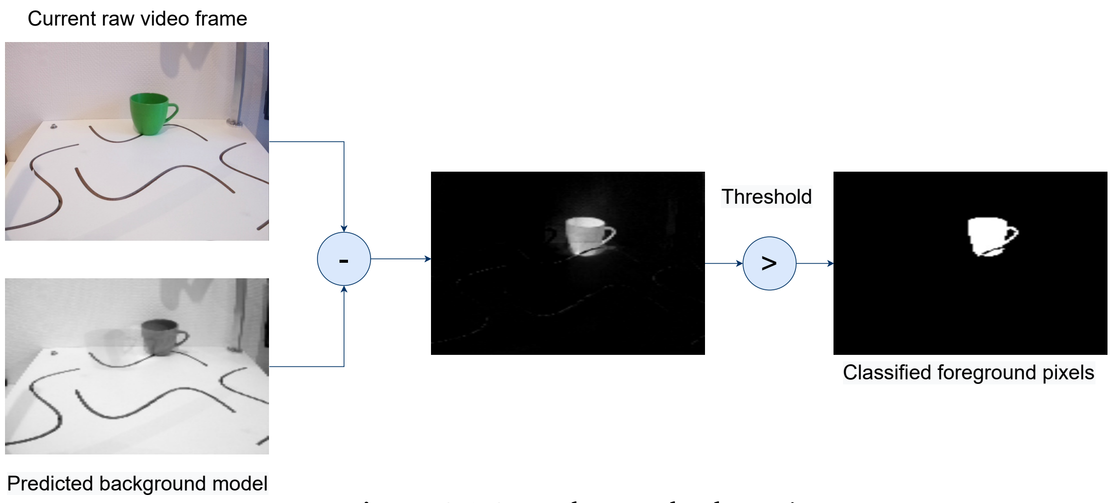


Figure 2.3.2: Background subtraction

2.4 Convolutional Neural Network

A Convolutional Neural Network (CNN) for image classification takes as input an image and outputs a predicted label among certain categories, e.g. chair, boat, cup. An input image is of size height $\times$ width $\times$ depth, where the depth can be interpreted as the color depth of the image. Here, $d = 1$ refers to a grayscale image, while $d = 3$ refers to a RGB (Red Green Blue)-image.

A CNN is typically composed of several layers that each serves a dedicated purpose. These layers are convolutional layers, pooling layers and fully connected layers where the classification eventually takes part.

2.4.1 Convolution layer

The convolution layers in combination with the pooling layers extract the most important features from the input images. A convolution operation in the sense of feature extraction is applying a kernel, which is a matrix, that is slid over the input image, multiplied with the overlapping pixels and finally adding them together as a measure of similarity between the kernel and the image. This operation will create an output that enhance some feature, depending on the kernel used, that is detected at the particular region in the image. A common kernel used in image feature extraction is one that detects vertical or horizontal edges in an image. In a CNN however, the kernel pixel values are learned by the network. This way it obtains the features it deems most important for the training data provided. Yet, the size of the kernel is specified by the user and is thus a hyper-parameter to consider when designing the convolutional layers. Another parameter that is user-specified is the stride step which is the number of pixels the kernel is shifted for each multiplication operation. Finally, the convolution operation must have a way of handling the borders of the input image if the overlap between the kernel and stride does not coincide with the image size. This is referred to as the padding. Options here are valid padding, which only performs the convolution operation on the pixels of the input image resulting in a feature map of smaller dimensions than the input image. While a padded or same convolution will add zeros around the borders of the image such that the output dimensions match those of the input dimensions.

The output of the convolution is referred to as a feature map (He and Sun (2015)). A pooling layer is then applied to downsample the feature map which also reduces the sensitivity to the position of the features in the map (O'Shea and Nash (2015)).

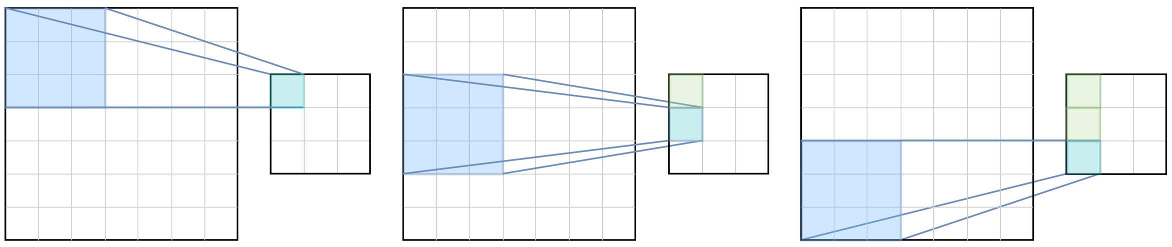


Figure 2.4.1: Convolution operation with a sliding kernel where stride step is two and padding is valid or zero

2.4.2 Pooling layer

Pooling layers may be applied to the feature maps produced by the convolutional layers. They serve the purpose of further downsampling the feature maps, as well as summarizing, or extracting the most prevalent features that are present in each section of the feature maps. This will decrease the number of parameters or weights in the final network. The two most common pooling types are average pooling and max pooling. The first taking the average of each patch and returning a single number, while the other simply returns the maximum number of a section. A section is here referred to as the window that is slid over the feature map, similarly to the filter that is slid over the image in a CNN layer.

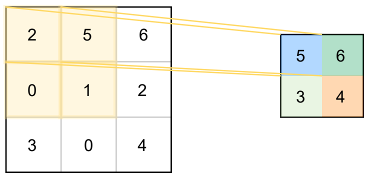


Figure 2.4.2: Max pooling operation with stride 1 and zero padding

2.4.3 Fully connected layer

The final part of the CNN network are the fully connected layers that have the same architecture as a regular feed-forward Artificial Neural Network (ANN). The input to this network is the set of flattened feature maps produced by previous parts. Each neuron receives inputs from connected neurons, with weights corresponding to them. The output of the neuron is further the result of applying an activation function to the weighted sum of the inputs multiplied with their corresponding weights and a potential added bias term. This operation is seen in Figure 2.4.3.

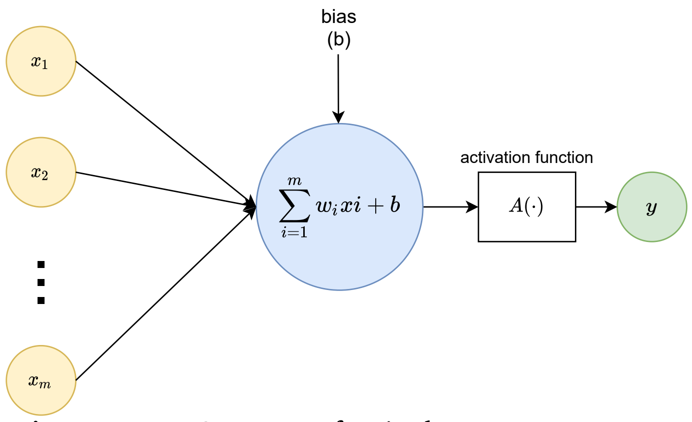


Figure 2.4.3: Output y of a single neuron

The purpose of an ANN is further to learn the weights related to each connecting neuron. This is in simple terms done by formulating an optimization problem that minimizes some loss function and updating the weights accordingly during training (Günther and Fritsch (2010)).

2.4.4 Activation functions

Following each node or neuron in a Neural network, is an activation function that outputs if the neuron should fire or not. Activation functions need to exhibit a non-linearity in order to learn the complex structures of the input data, which is the trait that ultimately renders neural networks to be universal approximators (Sharma (2017)). If the activation functions were to be linear, one would essentially be left with a linear regression model, which is far from a universal approximator.

The ReLU activation function is the most general and widely used in neural networks (Ramachandran et al. (2017)). Mathematically, it is expressed as in Equation 2.4.1.

$$y = max(0, x) \tag{2.4.1}$$

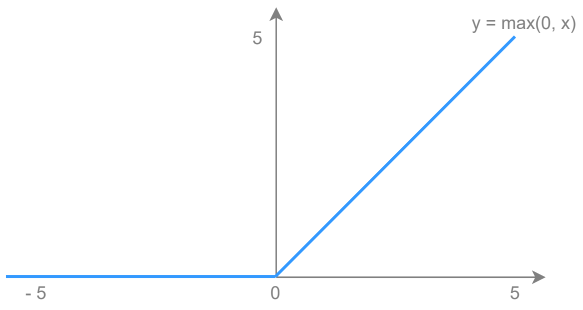


Figure 2.4.4: ReLU function

For classification tasks, such as image recognition, we require the network to output the recognized class in the image. It is however more interesting to know how certain the network is of the prediction. The Softmax activation function solves this by mapping the output of the final layer to a probability distribution, giving the probability of the input belonging to each of the possible classes. For instance, the final output after applying softmax, could predict that it is a 90% probability that the input image is of a cat, and a final 10% probability that the input image is of a dog, given that there are only these two possible classes. An illustration of this operation can be seen in Figure 2.4.5.

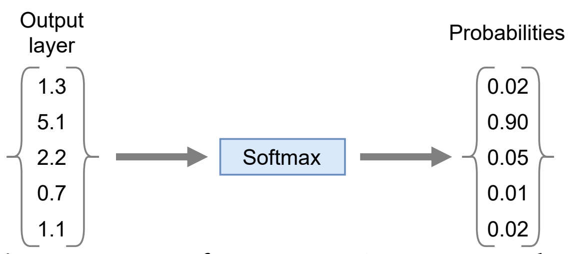


Figure 2.4.5: Softmax operation to output layer

The mathematical formulation of the softmax function is as in Equation 2.4.2.

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))} \tag{2.4.2}$$

2.5 Compressed Sensing

The following theory is partially or fully retrieved from the author's preproject that also utilized CS and SRC for image classification.

Compressed sensing (CS) is a method of signal compression that enables successful representation and reconstruction of a signal with far less samples than what the well-known Nyquist-Shannon sampling theorem requires. CS benefits from the fact that a signal's frequency content is highly sparse in some basis, which is the case for most natural signals.

A signal $\mathbf{z} \in \mathbb{R}^n$ is said to be *k-sparse* if it can be represented as a linear combination of only k basis vectors. Consider a signal $\mathbf{x}$ that is sparse in the basis $\mathbf{\Phi}$

$$\mathbf{z} = \mathbf{\Phi}\mathbf{s} \tag{2.5.1}$$

Suppose only m samples of the signal are taken. Then the measurement vector of $m \times 1$ dimensions is $\mathbf{y}$ where each element is a single measurement. Then $\mathbf{y}$ can be written as

$$\mathbf{y} = \mathbf{\Theta}\mathbf{z} = \mathbf{\Theta}\mathbf{\Phi}\mathbf{s} \tag{2.5.2}$$

Where $\mathbf{\Theta}$ is an $m \times n$ measurement matrix.

This yields a highly underdetermined system. The aim of CS is to find the sparsest solution $\hat{\mathrm{s}}$, that successfully recovers $\mathbf{z}$ from m measurements

$$\hat{\mathrm{s}} = \min_{\mathbf{s}'} \|\mathrm{s}'\|_0, \text{ subject to } \mathbf{y} = \mathbf{\Theta}\mathbf{\Phi}\mathrm{s}' \tag{2.5.3}$$

Solving Equation 2.5.3 is an NP hard problem and difficult to approximate. Yet, sparse representation and CS theory reveals that the sparsest solution of Equation 2.5.3 can be obtained by relaxing the norm to l_1 (Donoho (2006)). The problem can therefore be reformulated

$$\hat{\mathrm{s}} = \min_{\mathbf{s}'} \|\mathrm{s}'\|_1, \text{ subject to } \mathbf{y} = \mathbf{\Theta}\mathbf{\Phi}\mathrm{s}' \tag{2.5.4}$$

To account for corrupted or noisy data the equality constraint is relaxed with a residual error, leading to the quadratically constrained ℓ_1-minimisation problem which is the standard formulation of sparse reconstruction

$$\hat{\mathrm{s}} = \|\mathrm{s}'\|_1, \text{ subject to } \|\mathbf{y} - \mathbf{\Theta}\mathbf{\Phi}\mathrm{s}'\|_2 \leq \epsilon \tag{2.5.5}$$

2.5.1 Time complexity of ℓ_1-minimization

The standard formulation of ℓ_1-minimization is a linear program and can be solved with high accuracy using interior point-algorithms Ge et al. (2011). Unfortunately, traditional linear programming solvers solves the problem of ℓ_1-minimization in cubic time, which is unsuitable for large-scale applications. Therefore, attempts at developing more efficient solvers have been a central topic for CS advocates. Donoho and Tsaig (2008) propose a solver based on homotopy that is able to recover solutions with t non-zeros in $\mathcal{O}(t^3 + n)$ time, making it linear in the size of the training set.

2.5.2 Sparse Representation based Classification

Based on theory from CS and Sparse Representation, the application of Sparse Representation for Face Recognition is presented in Wright et al. (2008).

An image can be represented as a signal flattened into an $m \times 1$ vector $\in \mathbb{R}^m$. This signal can then be represented by a basis of orthogonal and unit length vectors $\mathbf{d}_i$, forming an $m \times n$ orthonormal basis matrix $\mathbf{D}$, which is also referred to as a dictionary matrix (Wright et al. (2008)).

When running the classifier, a new test image $\mathbf{y}$ belonging to class j, can be expressed as a linear combination of the column vectors that make up the training images of that same class as

$$\mathbf{y} \approx \mathbf{D}_j \mathbf{c}_j, \text{ or } \quad \mathbf{y} \approx \sum_{i=1}^{n_j} c_i \mathbf{d}_{j,i} \tag{2.5.6}$$

Where $\mathbf{D}_j \in \mathbb{R}^{m \times n_j}$ has columns $\mathbf{d}_{j,i}$, and n_j is the number of training samples for class j such that $\mathbf{c}_j \in \mathbb{R}^{n_j}$. The input image $\mathbf{y}$ in terms of *all* the training samples for L number of classes, where each $\mathbf{D}_j$-matrix is concatenated into what becomes the final dictionary matrix $\mathbf{D}$ is then

$$\mathbf{y} = \mathbf{D}\mathbf{c} = \begin{bmatrix} \mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_L \end{bmatrix} \mathbf{c} \tag{2.5.7}$$

where $\mathbf{D} \in \mathbb{R}^{m \times n}$, and $n = \sum_{j=1}^{L} n_j$. The coefficient vector is thus

$$\mathbf{c} = \begin{bmatrix} 0, ..., 0, c_{j,1}, c_{j,2}, ..., c_{j,n_j}, 0, ..., 0 \end{bmatrix}^T$$

such that coefficients corresponding to other classes than the one $\mathbf{y}$ belongs to, in this case j, is ideally zero. In practice however, there will be small non-zero coefficients associated with other class images as well due to noise or modelling errors (Carrillo et al. (2016)).

Each element $c_{j,i}$ of the coefficient vector $\mathbf{c}$ will denote a weighting coefficient for each column vector in the dictionary matrix, $\mathbf{D}$ for the new test image $\mathbf{y}$ belonging to class j. This relationship is illustrated in Figure 2.5.1.

For a valid test image $\mathbf{y}$, this $\mathbf{c}$-vector contains zero-elements for all other classes than the one in question, making it a highly sparse vector when the number of training images and classes are sufficiently large. The CS optimization problem Equation 2.5.4 can thus be adapted as follows

$$\min \|\mathbf{c}\|_1, \text{ subject to } \|\mathbf{y} - \mathbf{D}\mathbf{c}\|_2 \leq \epsilon \tag{2.5.8}$$

The representation error or the residual error of a class p is calculated by keeping the coefficients in $\mathbf{c}$ corresponding to class p, and setting the rest to zero. This is achieved by introducing a characteristic function η_j, that selects the jth class. The residual of a class p can then be represented as a function of the classes by

$$r_p(\mathbf{y}) = \|\mathbf{y} - \mathbf{D}\eta_p \mathbf{c}\|_2 \tag{2.5.9}$$

The recognized class of the input signal, $\mathbf{y}$ is retrieved as the class with the smallest residual error

$$\text{identity}(\mathbf{y}) = \min_j r_j(\mathbf{y}) \tag{2.5.10}$$

The full workflow of performing a single classification is comprised of the following steps

Algorithm 2: Sparse Representation Classifier (SRC)

1. The input to SRC is a matrix of training images that form a dictionary $\mathbf{D} = [\mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_L] \in \mathbb{R}^{m\times n}$ for L classes, a test sample $\mathbf{y} \in \mathbb{R}^m$ and an error tolerance $\epsilon > 0$.

2. Normalise the columns of D to have unit length

3. Solve the ℓ_1 minimisation problem in Equation 2.5.8

4. Compute the residuals from Equation 2.5.9 for each class $j = 1, 2, ..., L$

5. The given test sample is then classified as the class that provides the minimum representation error from Equation 2.5.10, given that the test sample is accepted as a valid input image, see subsubsection 2.5.2

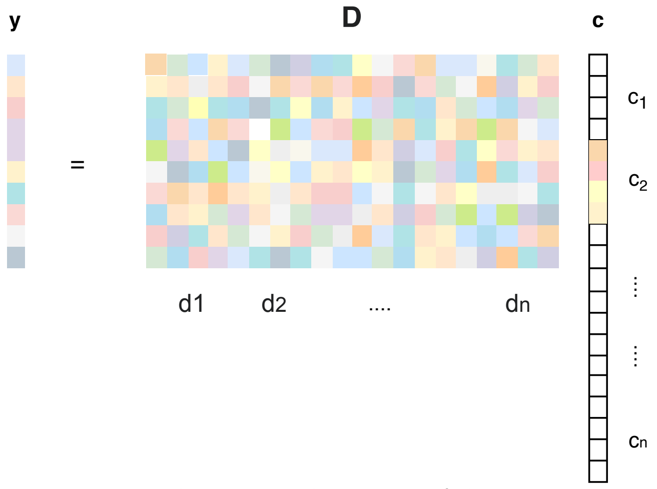


Figure 2.5.1: Query image $\mathbf{y}$ as a linear combination of the training image set $\mathbf{D}$ with corresponding weights given by $\mathbf{c}$. White entries signify zero-elements

Classification threshold

Prior to making a prediction based on the class residuals of an input image, the validity of the image itself and thus also the prediction must be assessed. This is done by evaluating the sparsity of the coefficients vector obtained by the minimization step in Equation 2.5.2. The idea is that a valid input image will have sparse coefficients more concentrated around a single class, while an invalid input image that could just as well be a random image not associated with any class, will have its coefficients spread across several classes. The predicted label for such an image should thus be discarded. The metric that captures this "sparsity"-score and is used in this report is the one presented in Wright et al. (2008), called the *SCI-score*. The score takes as input the coefficients vector obtained from Equation 2.5.5 and returns a score SCI $\in [0, 1]$ where 0 denotes coefficients spread out perfectly even across all classes, and 1 denotes an input image that can be described solely by images from a single class.

A user specified threshold, τ, is selected as the value the SCI-score must pass in order for the predicted label to be accepted.

$$\text{SCI} \geq \tau \tag{2.5.11}$$

2.6 Object detection using Yolo

Object detection is a subdivision of computer vision tasks that aims at both localising and classifying objects in a frame. When an object is localised, it is identified as belonging to one of the classes in the provided set of defined classes.

You only look once (Yolo) is a family of state-of-the-art object detection algorithms first introduced by Redmon et al. (2016) in 2016. Yolo became a game-changer in the computer vision community as it proved itself capable of providing one of the fastest object detection algorithms at the time. This made it suitable for real-time object detection applications. In the following years, newer versions of Yolo models have been released, with most iterations implemented and maintained in the open-source Darknet framework Redmon (2013). In 2020, Yolov5 was released, this time based on the PyTorch framework (Jocher et al. (2020)). PyTorch is an open-source optimised deep learning library compatible for running on CPU and GPU (Paszke et al. (2019)). This newest version of Yolo have outperformed all previous versions in addition to being even more suitable for smaller mobile devices as its size have been reduced with 90% compared to its predecessor Yolov4.

The Yolo algorithm is simpler than its predecessors in that it uses a single convolutional network to simultaneously predict bounding boxes, and class probabilities for those boxes in an image Redmon et al. (2016). The algorithm "looks" at the image once, hence the name, and extracts all this information in a single run. The problem of object detection is thus reformulated as a regression problem; relating image pixels to bounding box coordinates and class probabilities. This results in a high-speed model which is well suited for processing a video-stream or other real-time applications.

2.6.1 Evaluation metrics

Typical metrics for evaluating the performance of a classification model is Precision, Recall and F1-score. These are defined in terms of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) of the predictions on a test set. True positives and true negatives are classifications that match the ground truth, while false positives and false negatives are predictions that are not equal to the true label. The F1-score can be considered as the harmonic mean of the precision recall scores, and as an overall measure of the model's accuracy. A high F1-score indicates that both Precision and Recall is high, which reflects a good model.

$$Precision = \frac{TP}{TP + TN} \quad Recall = \frac{TP}{TP + FN} F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

A high precision typically yields a higher confidence of a correct prediction, but might in turn yield a lower recall or sensitivity to a true label. Due to the desirable trait of both a high precision and a high recall, a precision-recall curve can be plotted that shows the trade-off between the two for different confidence thresholds. Furthermore, from the precision recall curve, the Average Precision (AP) can be calculated as a means of summarizing the precision-recall curve into a single value. The AP is calculated as the sum over all precisions at every threshold value, multiplied by the change in recall.

$$AP = \sum_{k=1}^{N} Precision(k) \Delta Recall(k)$$

Here, N signifies the number of threshold values considered.

Object detection models are usually evaluated at different threshold for the predicted bounding boxes against the ground truth bounding boxes. At each threshold, different predictions may occur which results in different Precision Recall values. Another important metric used for evaluating object detection models is thus the *mean Average Precision*, or mAP, which is the mean of the APs calculated for each class at different thresholds.

2.7 3D machine learning

3D machine learning is an interdisciplinary field that integrates machine learning, computer vision, and computer graphics to enhance 3D understanding. It has gained traction over the last couple of years due to its broad application areas in Robotics, Autonomous driving, Augmented or Virtual reality and Medical Image processing. If computers are to succeed at these tasks, they require a thorough 3D understanding of the world. This understanding need in addition to be robust, fast and lightweight as most of these applications require real-time capabilities (Cunico et al. (2019)).

2.7.1 3D Pose estimation

3D pose estimation is the task of determining an object's translation and orientation relative to some reference coordinate system (Xiang et al. (2017),). Thereby recovering its full six degree of freedom (6DoF) coordinates stemming from three rotational angle coordinates, and three spatial coordinates.

The 3D pose of an object can in general be recovered either by localizing a set of keypoints that describe the object shape, or by estimating the camera viewpoint relative to a depicted object (Tulsiani and Malik (2015)). State of the Art approaches for methods of pose estimation using either RBG, RBG-D (depth), or point cloud information (i.e. 3D information) as input (Cunico et al. (2019)). These methods can be based on feature-matching, which essentially extracts 3D features from RGB images, and use them to recover a full 6DoF pose. This is done by either matching the features with known objects in a feature database (Lowe (2004)). Or by matching coherent feature keypoints between 2D images and 3D point clouds(Nadeem et al. (2020)). Recent methods are particularly focused around using CNNs for extracting relevant 3D information (Kendall et al. (2015), Xiang et al. (2017), Su et al. (2015)).

Translation and orientation

The rotational angles are expressed as the Euler angles azimuth (ϕ), elevation (θ) and in-plane rotation (ψ) that can be used to describe the orientation of a rigid body.

If a rigid body is rotated, each basis vector of the coordinate system fixed to its body, will experience a rotation. The total of this operation is contained in a rotational matrix $\mathbf{R}$. This matrix can further be decomposed into three rotational matrices - one for each rotation of either ϕ, θ or ψ about one of the basis vectors which are referred to as elementary rotations. These elementary rotations are defined as follows

$$\mathbf{R_x}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{bmatrix} \quad \mathbf{R_y}(\theta) = \begin{bmatrix} cos(\theta) & 0 & -sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & cos(\theta) \end{bmatrix}$$

$$\mathbf{R_z}(\psi) = \begin{bmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

While $\mathbf{R}$ exhibits the rotational change of the body, the translation $\mathbf{T} = [T_x, T_y, T_z]^T$ refers to the coordinate displacement of the body frame resulting from moving the frame through space.

The results of applying a translation and a rotation to a rigid body frame is illustrated in Figure 2.7.1.

Referring back to our application, this simple transformation can be used for updating the orientation of the objects considered in our Digital Twin. By using Pose estimation for estimating the camera viewpoint, i.e. the Euler angles for which the camera is oriented in relation to the object, the camera rotation matrix can be retrieved and applied to the previous camera viewpoint coordinates to obtain the updated scene.

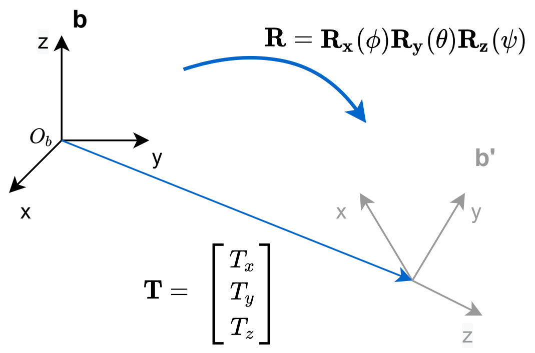

b
z
O_b
y
x
$\mathbf{R} = \mathbf{R_x}(\phi)\mathbf{R_y}(\theta)\mathbf{R_z}(\psi)$
b'
x
y
z
$\mathbf{T} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$

Figure 2.7.1: Movement of a rigid body through 3D space will produce a translation and a rotation component to the body frame from $\mathbf{b}$ to $\mathbf{b}'$

Chapter 3

Method and Setup

The full workflow of detecting a positional change of a 3D-object consists of three modules. The proposed method of performing change detection is tested using an experimental setup that is built and designed to fit this purpose. This chapter presents the experimental setup used for testing the framework as well as the implementation details for each individual module.

Furthermore, with research question two in mind, namely *Can synthetic data programmatically acquired be used for training image classifiers to further use the trained model in the real world?*, a virtual experimental setup is considered. This virtual experimental setup is used for the programmatic acquisition of synthetic data that will be used to train the various image classifiers CNN, SRC and yolo. The results following this experiment will be evaluated with discussions around their performance as well as which model is best equipped for performing object detection in the proposed framework.

Reverting back to the Change Detection application, each part of the workflow constitute a preprocessing step for the subsequent module. The procedure begins with real-time motion detection performed by DMD. This is described in subsection 3.3.1. The output of this part is further passed on to the next module, namely object detection performed by yolov5 described in subsection 3.3.2. Lastly, the localised objects with their bounding box estimates are processed by a Pose estimation algorithm that estimates the object orientation with respect to the camera reference frame. This is explained in subsection 3.3.3. It should be noted that the proposed framework is constrained to detecting rotational changes only. This is partly due to the pose estimation model used, as well as the absence of the necessary apparatus needed to extend the approach to estimate a translation component of the movement as well. This is also mentioned as part of future works in chapter 5.

The full workflow is finally summarised to give an overview of the working pipeline in subsection 3.3.4.

3.0.1 CAD models

CAD models of the 3D objects that were used in this analysis were selected from the website `https://grabcad.com`, each depicted in Figure 3.0.1 and downloaded in .stl format. These particular objects were selected for their different shapes, with some being more oblong, their difference in symmetries and textures. Apart from the cup, the objects exhibit non-symmetry, which is a feature most pose estimation algorithms are dependent on Labbé et al. (2020). The cup, although exhibiting symmetry except for its handle, is thus included for comparison purposes.

Each CAD model was 3D printed in three colors; black, white and green. The color variations serve the purpose of testing for performance variations in the different modules presented later in the report. The white objects are in particular considered in the motion detection module performed by DMD.

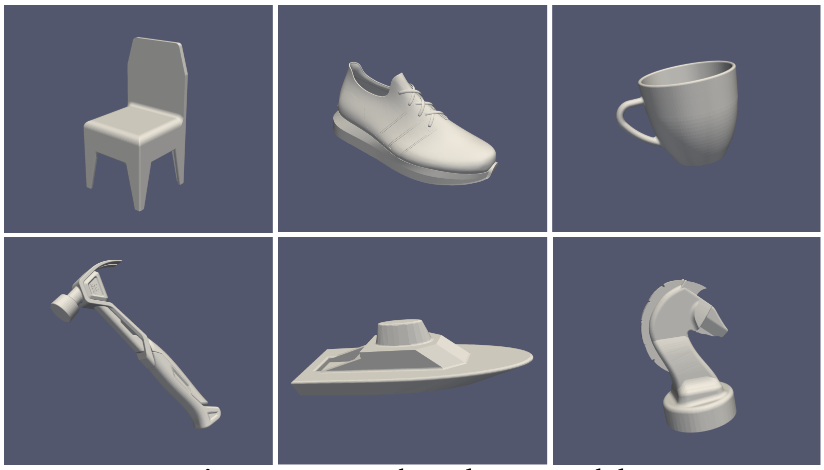


Figure 3.0.1: Selected CAD models

3.1 Virtual experimental setup

The virtual experimental setup is created in ParaView (Ahrens et al. (2005); Ayachit (2015)), an open-source data analysis and visualization application, and an excellent tool for exploring 3D-data. Several options are available for customizing the exploration of 3D-data in ParaView. Specifically, the rendering background can be changed as well as the texture, color and lighting options for the 3D models. For the purpose of synthetic data acquisition in the following experiments, the virtual scene was altered to mirror that of the real-world scene where the 3D-printed objects would reside. An image of the appropriate background was therefore captured and uploaded as the rendering background in ParaView. Further, the color and lighting of each 3D model was modified to reflect that of the physical object. This was done for the black and green objects. Since the physical background in the experiments were white, the white objects were omitted for the image recognition parts. This decision was made after performing some initial tests with motion detection, and observing that these objects were hard to detect and separate from the white board in the scene. In addition, it was recognised that the the white objects against a white background would likely cause problems for the Pose estimation model as the 3D information may be hard to extract when the distinction between the object and the background is poor (Choi and Christensen (2012)).

The result of these operations can be seen in Figure 3.1.1.

ParaView comes with a seamless Python integration that enables the creation of scripts for rendering views from different angles and taking snapshots of the rendered views. This feature is utilized for acquiring the synthetic datasets of the selected CAD models.

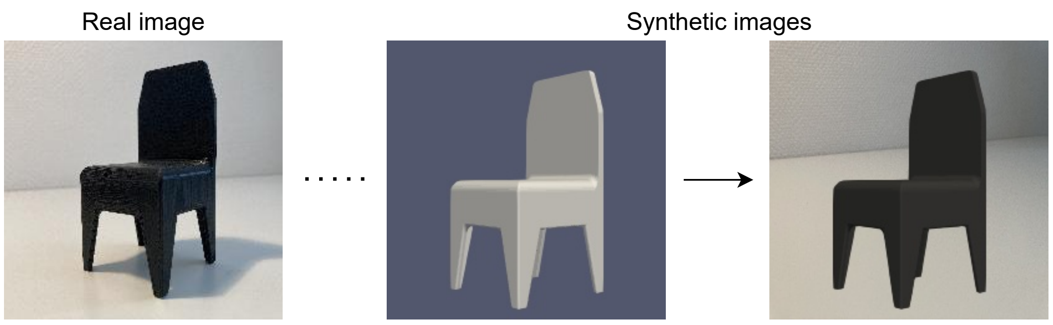


Figure 3.1.1: Adapting the synthetic environment to match the physical one

The first view of each object is rendered from the side in ParaView. The camera reference frame or the rendered viewpoint with respect to the object frame, and the angles for which the object is rotated is illustrated in Figure 3.1.2. The angles are defined as the Euler angles where ϕ refers to the azimuth-angle, θ refers to elevation and ψ denotes the in-plane rotation angle. The object is further rotated about its center with an increasing ϕ angle offset for each sampled image, as illustrated in Figure 3.1.3.

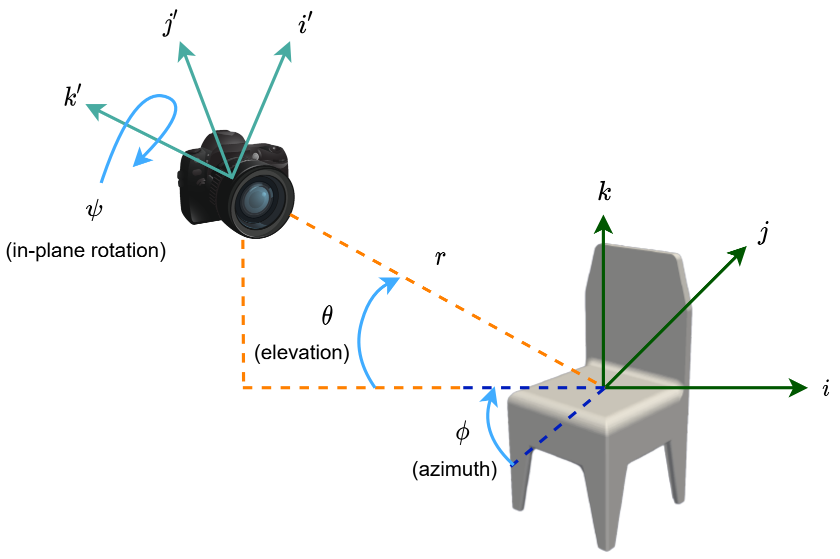


Figure 3.1.2: Angles for which the object is rotated w.r.t the camera

3.1.1 Synthetic data acquisition

The sampling angles and setup for which the synthetic datasets were acquired for either the CNN, SRC or yolo varied slighty. The following sections present the procedure for each of them.

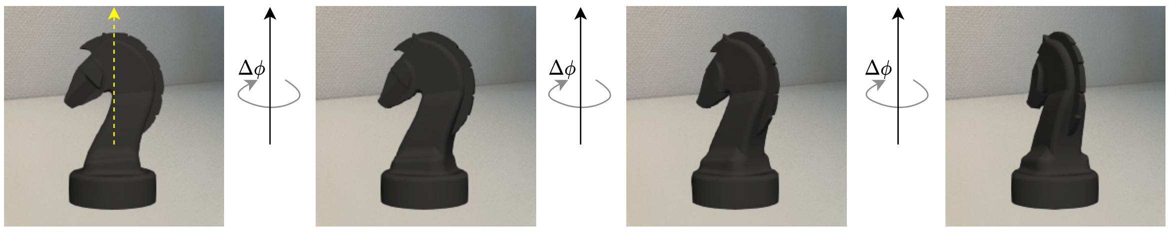


Figure 3.1.3: Images sampled for an increasing Azimuth angle

CNN

The sampling angle offsets for acquiring data for CNN are summarized in Table 3.2.3. An offset azimuth angle of $20°$, consequently yields $\frac{360°}{20°} = 18$ images for a full rotation of the object. The procedure is further repeated at three elevation levels. This is to reflect the test images that were captured from either a level or elevated perspective.

For the CNN dataset, the objects were in addition applied an in-plane rotation of $30°$, before repeating the above procedure but with two elevation levels at level and $15°$ elevation. As the in-plane rotation in addition to a large elevation resulted in a poor representation of the object, the last elevation level was omitted.

Table 3.1.1: Sampling angle offsets for CNN

$\Delta\phi$ (azimuth)	$20°$
$\Delta\theta$ (elevation)	$15°$
$\Delta\psi$ (in-plane rot.)	$30°$

SRC

The SRC algorithm is known to be sensitive to pose variations at test time, when the training and testing images of the objects are misaligned (Wright et al. (2008), Zhang et al. (2013)). An intuitive approach for dealing with this challenge is simply to enrich the training data dictionary to contain more information. For instance, providing enough training samples with pose variations, thus expanding the feature set and increasing robustness to these variations (Zhang et al. (2013)).

Even so, for large pose variations of each object that spans a whole $360°$ rotation, the dictionary matrix would have to contain enough training samples to encompass this, and it might still not lead to sufficient mitigation of this effect. In addition, the optimization problem in Equation 2.5.8 would become much larger, which would result in increased computation time. Therefore, the synthetic dataset acquired for SRC was more conservative regarding the sampling angles, than what was the case for the CNN.

The resulting sampling angles used for SRC are summarized in Table 3.1.2. For SRC the following azimuth angles were omitted to limit large pose variations; $\{60°, 75°, 90°, 105°, 120°, 135°, 240°, 255°, 270°, 285°, 300°\}$.

Table 3.1.2: Sampling angle offsets for SRC

$\Delta\phi$ (azimuth)	15°
$\Delta\theta$ (elevation)	15°
$\Delta\psi$ (in-plane rot.)	0°

Yolo

The virtual setup considered for yolo is designed to mirror the physical experimental setup to be presented in section 3.2. Example renders of the 3D-objects in the virtual experimental setup in ParaView can be seen in Figure 3.1.4.

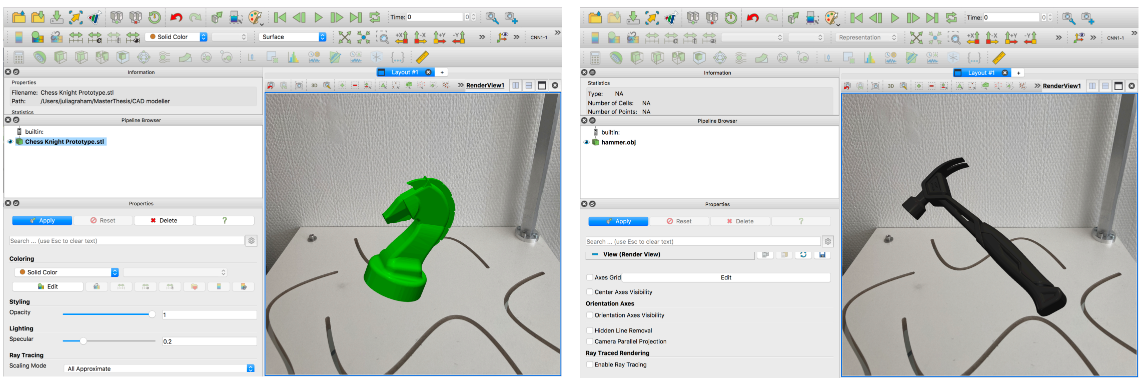
The sampling procedure for yolo is quite similar to that of CNN. The difference is the in-plane rotation which was omitted this time. This was to limit the amount of data that needed to be annotated. Additionally, it was recognised that several of the images with in-plane rotations yielded a poor representation of the object and could possibly lead to outlier data. The final selection of sampling angles are summarised in Table 3.1.3.

Table 3.1.3: Sampling angle offsets used in data acquisition scripts

$\Delta\phi$ (azimuth)	20°
$\Delta\theta$ (elevation)	15°
$\Delta\psi$ (in-plane rot.)	0°

Three elevation levels were used, i.e. 0°, 15° and 30°. Thus, the number of images per object amounted to $3 \times \frac{360}{20} = 54$.

Thus, the final dataset consisted of synthetic images of six green and black objects, each depicted in 54 images. This resulted in a total of 648 images.


Filename: Chess Knight Prototype.stl
Path: /Users/juliagraham/MasterThesis/CAD modeller

Coloring: Solid Color
Opacity: 1
Specular: 0.2
Scaling Mode: All Approximate

Type: NA
Number of Cells: NA
Number of Points: NA

Figure 3.1.4: Examples of renders in ParaView

3.2 Experimental set-up

The setup was built such that a camera is monitoring a scene where the objects reside. This scene was constructed with a white wooden plate with laser-cut trenches where the 3D-printed objects can move.

The scene is monitored and captured by a high quality (HQ) Camera and a 6 mm CS-mountlens compatible with a Rapberry Pi4 model B. The Raspberry Pi is installed with the Raspberry Pi OS which is GNU/Linux-based. The camera is mounted on a 3D-printed tripod stand depicted in, and further secured on an adjustable steel arm. The Raspberry Pi is secured on a steel plate that is positioned underneath the camera. This installation is depicted in Figure 3.2.1. Both the camera and the Raspberry Pi can be elevated along a vertical steel pole, and the camera can further be adjusted sideways and up an down. Furthermore, a light source in the form of a torch was installed to a steel arm overlooking the scene to provide additional lighting with the option to alter the light intensity. The full setup can be viewed in Figure 3.2.2.

The Raspberry Pi runs the DMD-algorithm for motion detection in real-time. This is further explained in subsection 3.3.1.

Table 3.2.1: Raspberry Pi4 model B technical specifications

System-on-a-chip	Broadcom BCM2711
Processor	Quad-core 1.5GHz Arm Cortex-A72 based processor
Memory	8GB LPDDR4 RAM
Power	5V/3A via USB-C
GPU	Broadcom VideoCore VI @ 500 MHz
OS	Raspberry Pi OS

Table 3.2.2: Raspberry Pi4 HQ camera specifications

Sensor	Sony IMX466
Sensor resolution	4056×3040 pixels
Sensor image area	$6.287mm \times 4.712mm$
Pixel size	$1.55\mu \times 1.55\mu$
Focal length	$6mm$
Resolution	3 MegaPixel

3.2.1 Real data acquisition

To test the trained models in the real world, real-world images of the 3D-printed objects were acquired and used for testing. Test sets for the black and green objects were acquired separately. The real environment considered was both against a plain white background, and with the experimental setup presented in section 3.2. Example images from each test set is depicted in Figure 3a, Figure 4a and Figure 3.2.5. For SRC, some of the test samples with large pose variations in line with those angles that were omitted in the training data were discarded. This was due to the arguments presented earlier regarding SRC's proneness to pose sensitivity. As no measures were taken to mitigate this effect by for instance creating a large training set including large pose variations, and a more frequent azimuth sampling rate, these particular images were not considered relevant at test time.

With regards to Yolo, each object was depicted from an elevated perspective, sampling 7 to 9 images per object category for different azimuth angles. This resulted in a total of 104 images in the real test set.

Furthermore, the number of samples of each object in the synthetic datasets and real test sets for black and green objects respectively, as well as for all three models are summarised in Table 3.2.3 and Table 3.2.4.

Table 3.2.3: Number of samples per object category

	Black objects						
CNN	boat	chair	hammer	knight	shoe	cup	**total**
Synthetic dataset	90	90	90	90	90	90	**540**
Real test set	8	7	8	7	9	7	**46**
SRC							
Synthetic dataset	26	26	26	26	26	26	**156**
Real test set	7	8	8	8	6	7	**44**
Yolo							
Synthetic dataset	54	54	54	54	54	54	**648**
Real test set	9	9	7	8	8	8	**49**

Table 3.2.4: Number of samples per object category

	Green objects						
CNN	boat	chair	hammer	knight	shoe	cup	**total**
Synthetic dataset	90	90	90	90	90	90	**540**
Real test set	12	15	13	11	16	8	**75**
SRC							
Synthetic dataset	26	26	26	26	26	26	**156**
Real test set	10	11	10	8	11	8	**58**
Yolo							
Synthetic dataset	54	54	54	54	54	54	**648**
Real test set	8	8	8	8	8	8	**48**

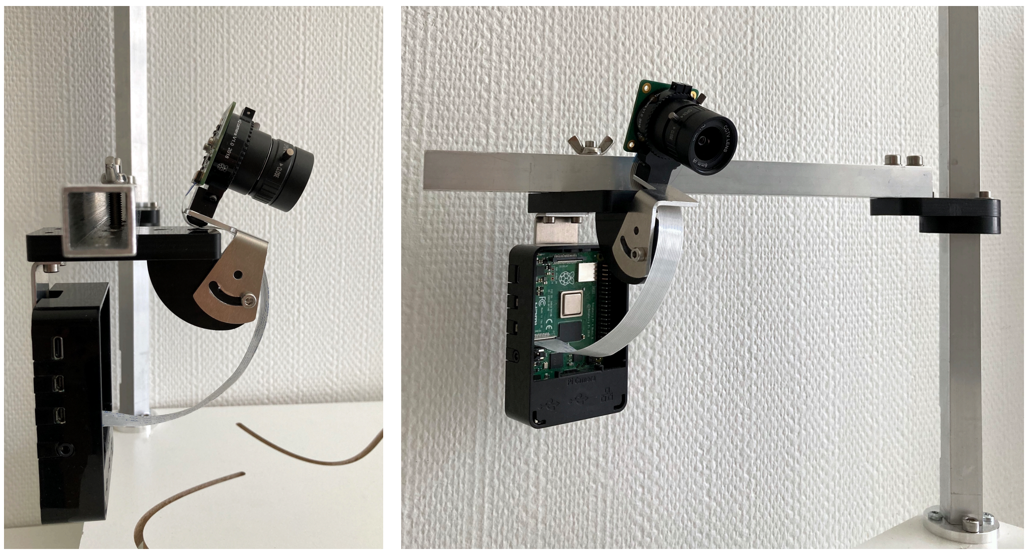


Figure 3.2.1: Raspberry Pi Camera mounted on a 3D-printed stand and secured on a steel arm

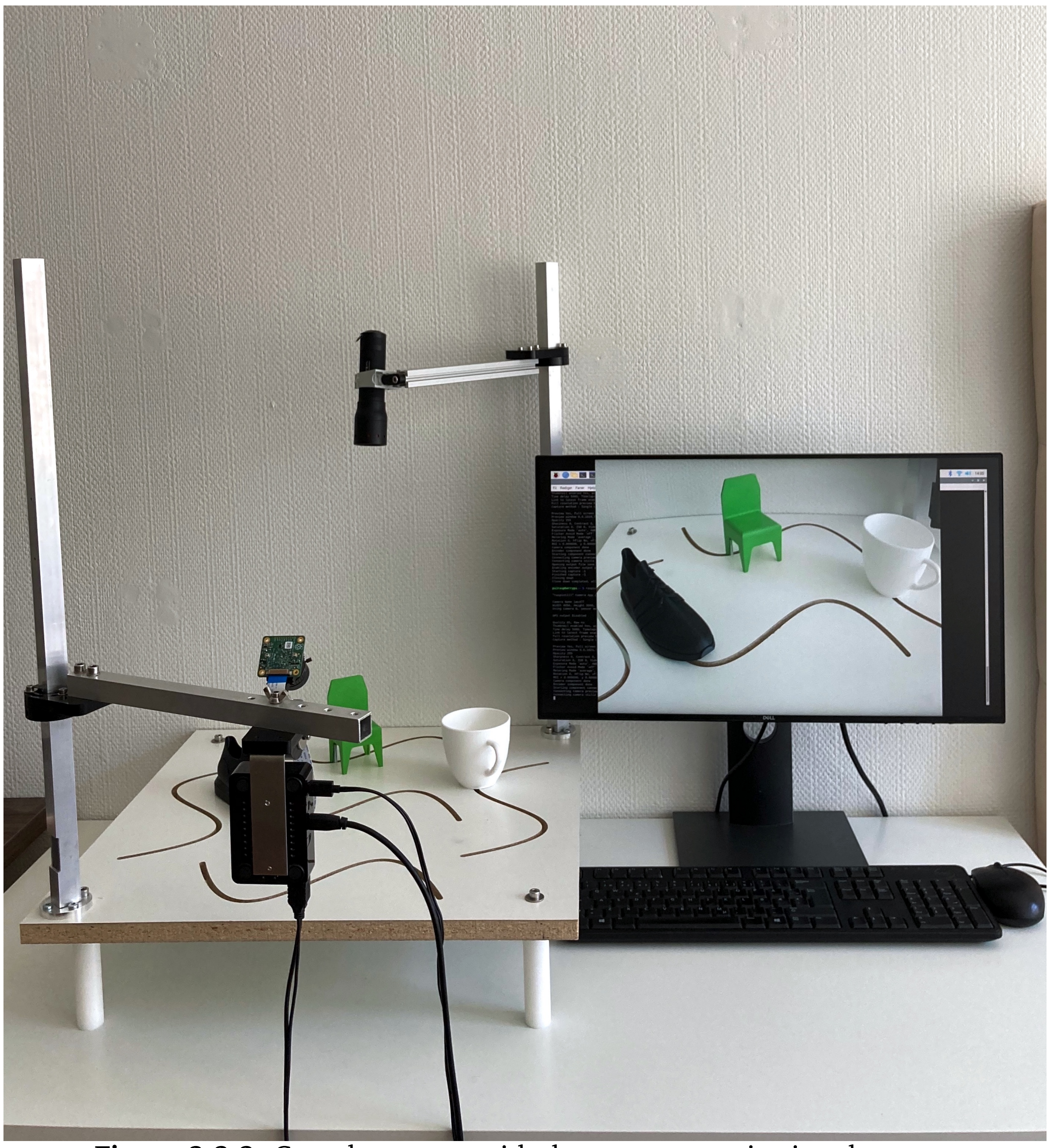
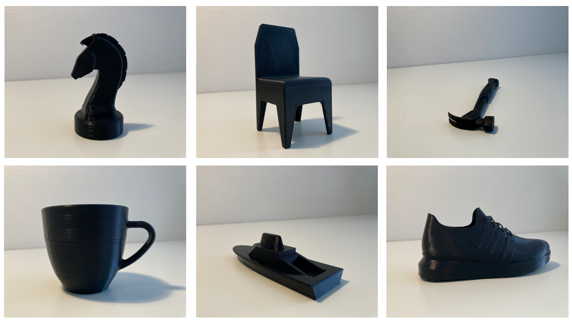


Figure 3.2.2: Complete setup with the camera monitoring the scene


(3a) Images from the real test set

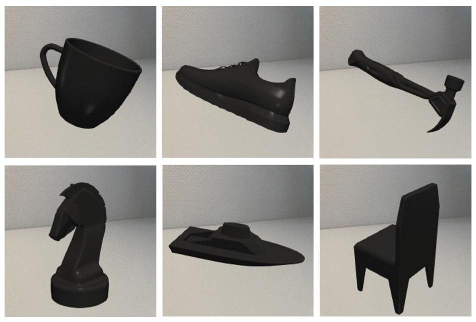
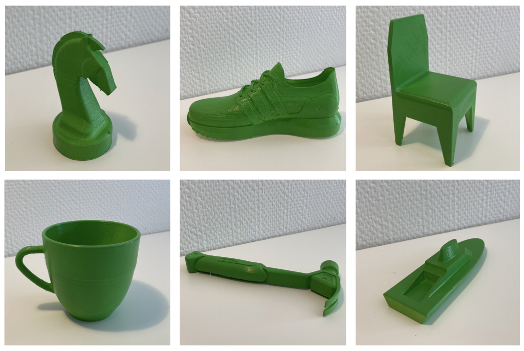

(3b) Synthetic data

Figure 3.2.3: Example images of black objects from the real test set


(4a) Real data

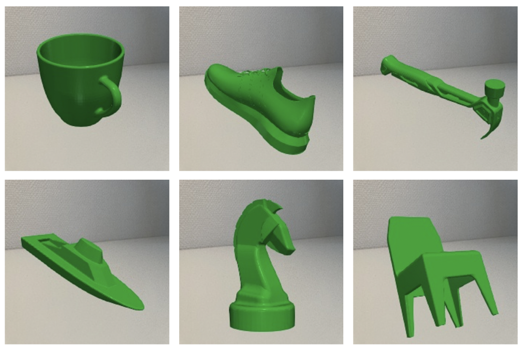

(4b) Synthetic data

Figure 3.2.4: Example images of green objects from the real test set

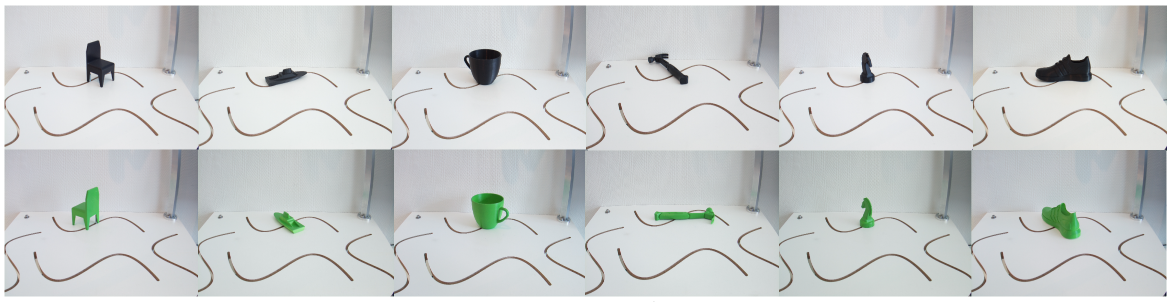


Figure 3.2.5: Example images from real yolo test set

3.3 Method

3.3.1 Motion detection using DMD

The DMD framework used for motion detection was implemented in Python. The algorithm used for processing a real-time stream of video was the algorithm presented in Nathan Kutz et al. (2017), namely Streaming DMD. For details refer to subsection 2.3.1. The size of the sliding window was set to $N = 10$, meaning that 10 subsequent video frames were evaluated at a time. The Δt parameter that shifts the window Δt timesteps for each iteration was set to $\frac{1}{3}$ seconds. The frame rate (fps), or the sampling rate of the video stream was 15 fps.

The camera is monitoring the scene while the Raspberry Pi is continuously running the Streaming DMD algorithm, computing the DMD modes from a fixed stream of video frames and performing background subtraction in real-time as illustrated in Figure 2.3.2.

The foreground model of the video frames will be the raw frame with the background subtracted. The foreground model is further thresholded by using the Euclidean distance in Equation 2.3.14, with the threshold set to $\tau = 0.3$. Thus, the pixels are either classified as foreground (white = 1) or background (black = 0). Based on this, motion is detected if enough white pixels are lighting up in the frame, signalling the presence of movement. The motion detection problem can thus be formulated as a binary classification problem, where the proportion of foreground pixels decides if motion is detected or not. Moreover, this proportion needs to exceed yet another threshold that is set by the user, and will be referred to as the *motion threshold*. This threshold was decided by trial and error, after visual inspection of the modeled foreground. The selection fell on a threshold deemed sensible in terms of not being too sensitive, resulting in false positives, yet not too strict and unable to detect motion, resulting in false negatives.

Prior to calculating the DMD modes, the raw frame is downsampled and converted to grayscale as steps to decrease the computational cost related to the SVD-decomposition in the DMD algorithm.

Upon detecting movement, the running script starts to sample the incoming video frames. At the time the scene is deemed stationary again, the algorithm will merely store the *last* frame where movement was detected, and delete the rest. This is to minimize storage, which is one of the motivations for designing this framework. This last moving frame can then be compared with the original position of the object, to estimate the rotational change. This is further explained in the upcoming sections. The process of motion detection is depicted in Figure 3.3.1.

3.3.2 Object detection and Image classification

The models were trained and tested on each color separately. This decision was made to compare if color variations had any effect on the performance of the classifiers. The implementation details and training procedure of each image classifier and yolo is further presented in this section.

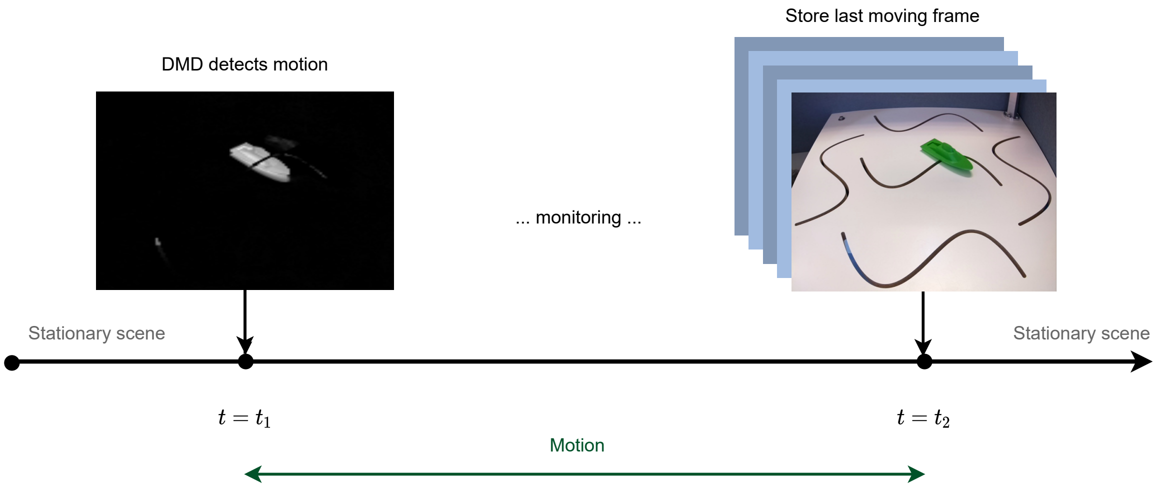


Figure 3.3.1: Motion detection timeline

CNN

The CNN is implemented in Python using the Keras API which has a Tensorflow backend (Chollet et al. (2015)). The network architecture is comprised of two convolutional layers with kernel size 3×3 followed by a ReLu activation function as in Equation 2.4.1. Further, a pooling layer with filter size 2×2 is added after each convolutional layer followed by a dropout layer with a dropout rate of 0.6. After these feature extraction steps, the remaining feature map is flattened before it is passed in to a dense layer with 256 neurons followed by a ReLU activation and a dropout layer with rate 0.7. The final output layer is a dense layer with the number of neurons equal to the number of output classes, in this case six. The final output or predicted label is the result of a softmax activation in Equation 2.4.2. The final architecture used is illustrated in Figure 3.3.2.

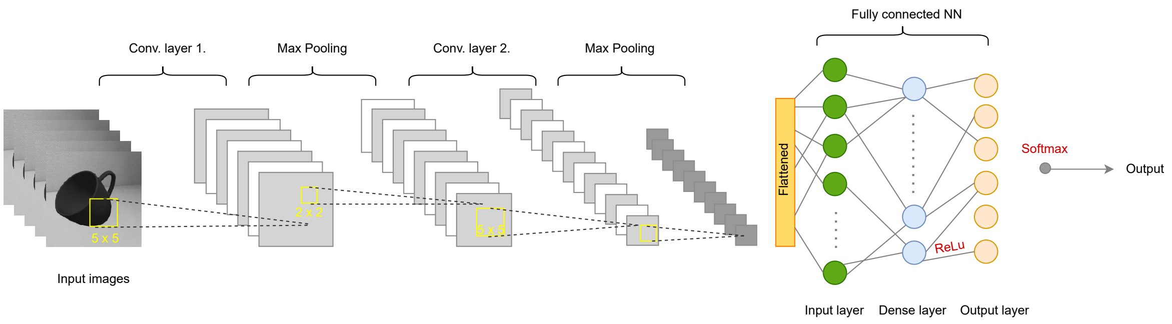


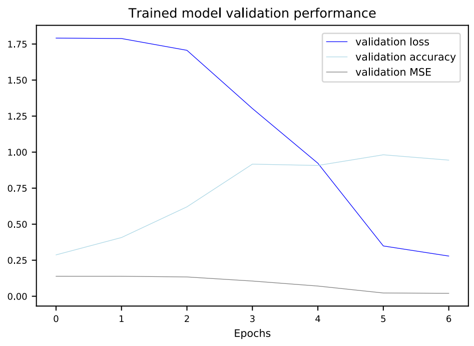
Figure 3.3.2: Architecture of the CNN used

For the CNN, a training and validation set was created from a 20% split in the data. Furthermore, the images were downsampled to 200×200 and converted to grayscale.

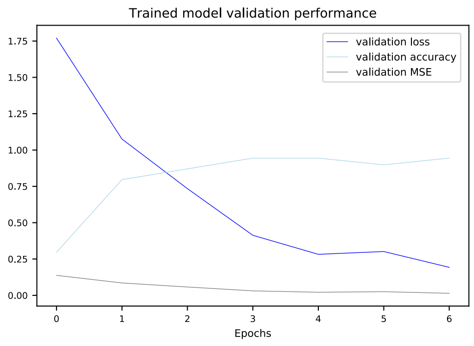
The network was trained and tested on both RGB and Gray-scale images for both color sets. For the black objects, this did not improve the performance of the network significantly, and thus the grayscale-setting was kept to avoid unnecessary model complexity. For the green objects however, a slight improvement was observed, which is why this network was trained and tested with RGB-images.

Table 3.3.1: CNN training parameters

Image size	200×200
Epochs	7
Optimizer	RMSprop
Learning rate	0.001
Kernel size	3
Pooling size	2
Batch size	32
Color ch. black objects	GRAY
Color ch. green objects	RGB



(3a) Black objects



(3b) Green objects

Figure 3.3.3: Validation performance during training of the two models

The network was trained with the parameters summarized in Table 3.3.1. It was trained for six epochs as a means of regularising the network against overfitting towards the synthetic data, in addition to observing that the validation performance stagnating from that point on.

The evolution of the two models' performance on the validation with respectively black and green objects can be seen in Figure 3.3.3. The same validation set was used when producing the confusion matrices with predictions from the final trained models on synthetic images in Figure 4.2.1 and Figure 4.2.4. Note that this is purely for comparison purposes against the real test set as it is a highly prohibited practice to use validation data at test time. Yet, as it is the real world images that are the actual test sets in this trial, it is for this reason deemed acceptable.

SRC

The SRC-algorithm is implemented in Python. For solving the ℓ_1-minimisation problem in Equation 2.5.8, the Python library CVXPY is used. The choice of solver is the open source ECOSsolver, which is an interior point solver for second order cone programming (Domahidi et al. (2013)).

Contrary to a CNN, the SRC algorithm does not require training in the sense of storing the learned knowledge in the model itself. The knowledge rather lies directly accessible in the dictionary matrix comprised of the training images. The sparsity that is present in most natural signals and images encourages the use of feature extraction for removing redundancy without the loss of information (Mairal et al.

(2007)). This is an important step towards a minimal and discriminate representation of the data, which ultimately leads to a more simplistic pattern recognition system (Zhang et al. (2016)). Depending on the number of training samples, and the choice of feature extraction for reducing the image size, the dictionary matrix will differ in shape. For this report, both downsampling and Principal Component Analysis (PCA) were tested for feature extraction. As PCA enabled better model performance, this was the final choice. The number of components for representing the data such that a $\sim 95\%$ explained variance in the data is obtain is a common choice when using PCA for feature extraction (Jolliffe and Cadima (2016)). For this trial however, it was observed that choosing $n_{components} = 25$ for the black objects which yielded an explained variance of 87%, and $n_{components} = 50$ with an explained variance of 92% for the green objects resulted in better model performance than the general rule of $\sim 95\%$ explained variance. The results presented are therefore with this choice of principal components.

PCA was applied to the total dictionary matrix after stacking each training image horizontally. With a total of 156 training images as reported in Table 3.2.3 and Table 3.2.4, the final dictionary matrices had dimensions 156×25 for the black objects and 156×50 for the green objects. The choice of threshold for validating the prediction using the SCI-score in Equation 2.5.11, was set to 0.5 which is a common threshold value for classification tasks (Burger and Gowen (2015)).

Yolov5

The Yolov5 model is implemented with the PyTorch framework, as mentioned in section 2.6. The training parameters selected for training Yolov5 are listed in Table 3.3.2. The network was implemented using the data platform provided by Kaggle, and the cloud computing services that comes with it. Kaggle is a platform and community for data science and machine learning, that provides datasets, hosts data science competitions which also recently recently released a cloud computing platform for free development of data science and Artificial Intelligence projects [1]. Kaggle provides free access to NVidia K80 GPUs, which was the accelerator used for training the network.

Table 3.3.2: Yolov5 Training parameters

Image size	416×416
Batch size	16
Epochs	180
Optimizer	Adam
Learning rate	10^{-2}
Device	NVidia Tesla K80 GPU

Furthermore, the images used for training and testing Yolo was annotatated for correct labeling of the bounding boxes. This was done in Roboflow Annotate [2], a tool for creating bounding boxes with labels for image datasets. Examples of the annotation process can be seen in Figure 3.3.4.

[1]https://www.kaggle.com
[2]https://roboflow.com/

For the Yolo trial, both green and black objects were considered together in both the training and test sets, as the conditions for data sampling were the same for both colors.

In terms of preprocessing, all images, both test set and synthetic data were down-sampled to 416×416. The synthetic training images were also applied augmentations of a 90% rotation as well as a crop, leading to a zoomed in version of the image. This led to a final training data size of 1554 images.

8% of the synthetic dataset was saved for validation during training, and 4% for testing. The *synthetic* test set was thus comprised of 65 images.

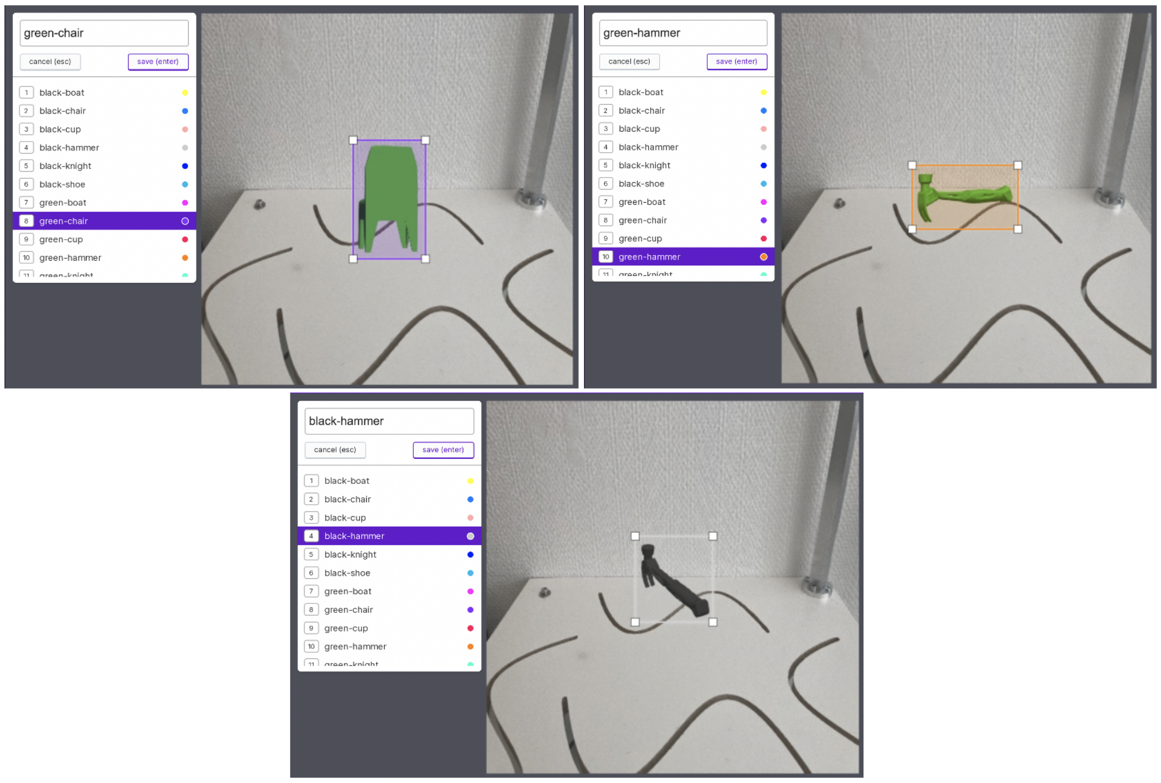


Figure 3.3.4: Annotating the synthetic data set in Roboflow

3.3.3 Pose estimation

The method for performing Pose Estimation is the one presented in Xiao et al. (2019*b*). A detailed description of the method is written in our published paper; Sundby et al. (2021). The following section is thus a direct citation from the paper, describing the method of Pose Estimation.

The essence of the method is to combine image and shape information in the form of 3D models to estimate the pose of a depicted object. The information that relates a depicted object to its 3D shape stems from the feature extraction part of the method. Two separate feature extractions are performed in parallel. One for the images themselves by using a CNN (ReHe et al. (2016)), and one for the corresponding 3D shapes. The 3D shape features are extracted either by feeding the 3D models as point clouds to the point set embedding network PointNet (Qi et al. (2017)), or by representing the 3D shape through a set of rendered image views that circumvents the object at different orientations, and further passes these images into a CNN.

The outputs of the two feature extraction branches are then concatenated into a global feature vector before initiating the pose estimation part. This part consists of a fully connected multi-layer perceptron (MLP) with three hidden layers of size 800-400-200, where each layer is followed by batch normalization and a ReLU activation. The network outputs the three Euler angles of the camera; azimuth, elevation, and in-plane rotation concerning the shape reference frame. Each angle is estimated as belonging to a certain angular bin $l \in \{0, L_\alpha - 1\}$*, for a uniform split of* L_α *bins. This is done through a softmax activation in the output layer which yields the bin-probabilities. Along with each predicted angle belonging to an angular bin, an angle offset,* $\hat{\delta}_{\alpha,l} \in [-1, 1]$ *relative to the center of an angle bin is estimated to obtain an exact predicted angle. For this report, the 3D model features are extracted by using the method of rendering different views of the 3D shape as mentioned above and using them as input to a CNN. This is done with the help of the Blender module for Python (Community (2018)). For the image features, YOLOv5 is first applied to create bounding boxes to localize each object, before being fed to the ResNet-18. The final MLP network was trained on the ObjectNet3D dataset, as this provided better system performance than the network trained on Pascal3D.*

The full pose estimation workflow is illustrated in Figure 3.3.5. The output of the network are thus the three estimated Euler angles (azimuth, elevation, in-plane rotation) describing the camera orientation relative to the object reference frame. This relation is illustrated in Figure 3.1.2.

The parameters used for training this part of the network are presented in Table 3.3.3.

Table 3.3.3: Pose Estimation network training parameters

Batch size	16
Epochs	200
Optimizer	Adam
Device	NVidia Tesla P100-PCIE-16GB GPU
Learning rate	$10^{-4}/10^{-5}$

As mentioned the network was trained on ObjectNet3D; a large scale database consisting of 100 class categories, 90,127 images depicting 201,888 objects, and their corresponding 44,147 3D shapes (Xiang et al. (2016)). Due to its large variety of object classes, it is suited as training data when the aim is to test the model on novel classes due to the regularisation that results from this fact. This became evident at test time as the model performed better when using ObjectNet3D for training rather than Pascal3D, which is another popular benchmark 3D dataset, however containing only 12 object categories (Xiang et al. (2014)).

Each object was rendered at three elevation levels; $0°$, $30°$ and $60°$, with an image sampled every $5°$ offset in azimuth. This amounted to $3 \times (\frac{360}{5}) = 216$ images per object. Examples of rendered object views are depicted in Figure 3.3.6. The first view of the object is with the camera facing its side. The camera is further shifted with an increasing azimuth angle as depicted in Figure 3.3.7. These are the 3D-images that are used as input to the shape encoder for feature extraction as depicted in Figure 3.3.5.

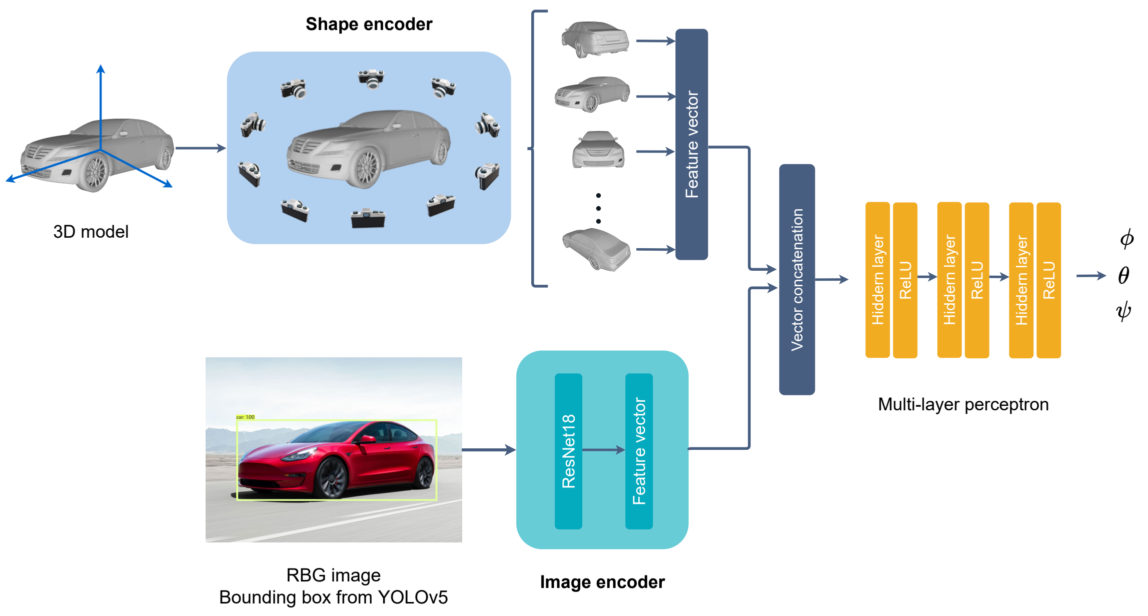


Figure 3.3.5: Pose estimation pipeline

Hardware used for training

The pose estimation network was trained and tested on the professionally administered compute platform for the Norwegian University of Science and Technology (NTNU), namely the Idun cluster (Själander et al. (2019)). The Idun cluster is part of a project between the Department of Computer Science and the IT Division at NTNU. It is a research cluster providing high-performance computing (HPC) infrastructure for rapid testing and prototyping of HPC software. It exhibits 73 nodes with and 90 general purpose graphics GPUs (GPGPUs). The Idun topology is comprised of four EPIC research infrastructure groups, either serving a distinct purpose. Each EPIC consists of at least three nodes with either NVIDIA Tesla P100 or NVIDIA Tesla V100 GPUs with respectively 16 and 32 GiB. Furthermore, each EPIC has two Intel Xeon cores and at least 128GiB of memory.

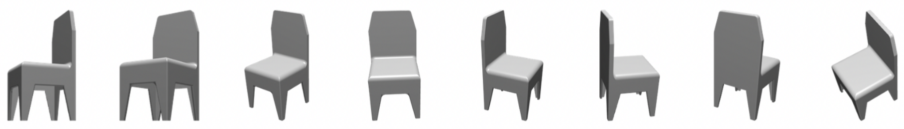


Figure 3.3.6: Example images of rendered views by Blender

Evaluation metrics

The accuracy of the predicted angles, (ϕ, θ, ψ), is based on which angle bin the output angle is associated with. As mentioned, the network estimates each angle as belonging to a certain angular bin $l \in \{0, L_\alpha - 1\}$, for a uniform split of L_α bins. Selecting a

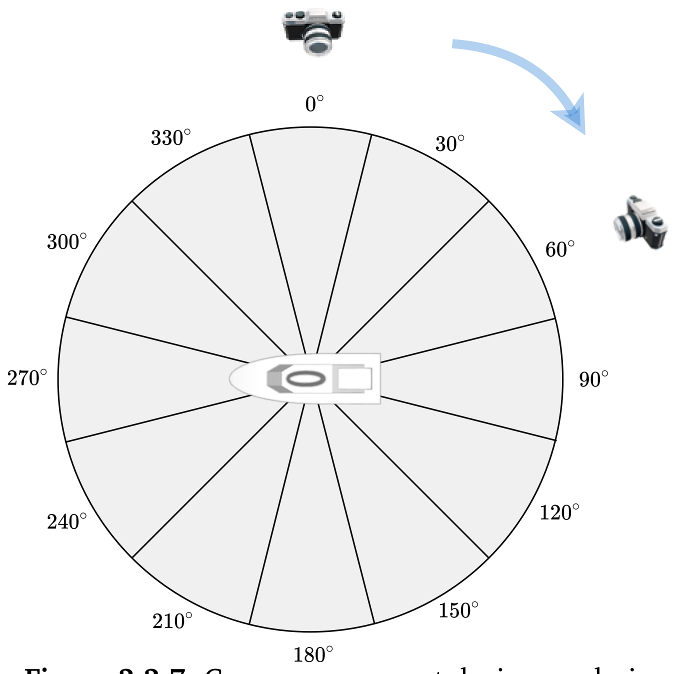


Figure 3.3.7: Camera movement during renderings

higher bin-number, L_α yields higher precision demands when determining the accuracy. A common metric used for determining accuracy is $Acc_{\frac{\pi}{6}}$ (Li et al. (2019), Xiao et al. (2019*b*)). This implies a bin-number of 12, and $\alpha = \frac{\pi}{6}$. Hence, $Acc_{\frac{\pi}{6}}$ is the percentage of all predicted angle differences smaller than $\frac{\pi}{6}$, i.e. 30° within the correct angle-bin. Thus, if the true camera angle is 0°, a predicted angle deemed correct can lie within $\langle -15^\circ, 15^\circ \rangle$. This is illustrated in Figure 3.3.8.

Another common metric used for evaluating Pose estimation models is Med_{ERR}, denoting the median of all viewpoint differences across all test samples.

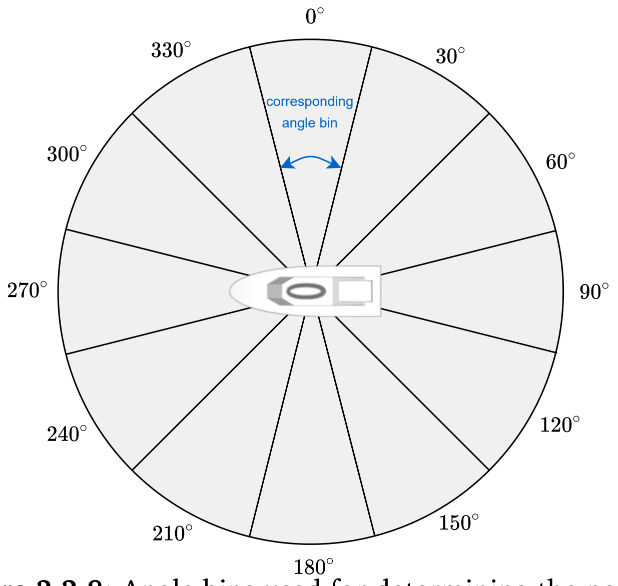


Figure 3.3.8: Angle bins used for determining the pose accuracy

3.3.4 Full workflow overview

The result from piecing together all three modules is the full workflow architecture illustrated in Figure 3.3.9. Consider the physical environment our Digital Twin is going to model. From an initial state of the physical scene, a change occurs and the workflow is initiated. From the knowledge of the full initial state, the object(s) pose will be known. The moment DMD detects motion in the frame, the monitoring initiates and images of the scene are sampled. As soon as the scene is deemed stationary again by the absence of foreground objects, only the last moving frame is stored while the rest are discarded for minimizing storage. This frame is further passed on to the Object Detection module, where Yolov5 outputs a bounding box estimate along with the predicted object class. An accurate bounding box estimate of the object is essential for providing the Pose Estimation module with an apt representation of the object. The Pose Estimation network will output three rotational angles that describes the camera viewpoint with respect to the object reference frame. This can however be translated into the object orientation in relation to the stationary camera by constructing a rotational matrix and taking its inverse. Thus, given an initial pose of the object and the rotational change it has undertaken, this information can be utilised by the Digital Twin to reconstruct the new scene on demand. Thereby circumventing the need for real-time sensor data updates, and the storage and bandwidth requirements that follow.

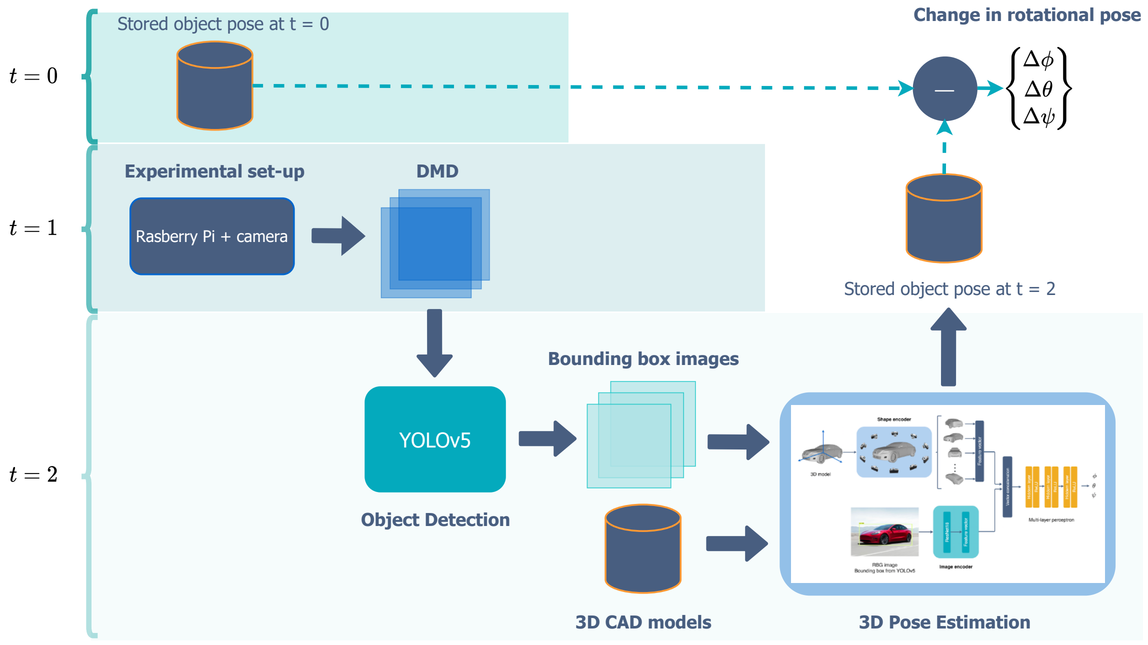


Figure 3.3.9: Architecture of change detection method

Chapter 4

Results and Discussions

4.1 Motion detection using DMD

The performance of DMD as a motion detector was inspected by moving objects of different shapes and colors in the monitored scene. For this, two different movements were considered as to enable a valid comparison of object color and object type which is only possible when the movements they undertake are similar. The first experiment is a pure translation, i.e. no rotation, along the middle trench on the board as can be seen in Figure 4.1.1. The images show the raw video frame to the out-most left, the reconstructed background by DMD in the middle and lastly the estimated foreground that results from background subtraction. The second experiment is a pure $360°$ rotation of each object about its center.

Based on thresholding the predicted foreground as in subsubsection 2.3.1, and comparing with the *motion threshold* as described in subsection 3.3.1, the results from performing these experiments are summarised in Table 4.1.1.

When the movement of the objects was slowed down, the DMD performance degrades notably. The predicted foreground becomes weaker and less foreground pixels pass the thresholded prediction. This is however expected of background subtraction approaches, as the pixel intensity between the background and moving foreground objects decrease with a slower speed and may fall short of the threshold in subsubsection 2.3.1 (Shaikh et al. (2014)). Secondly, this observation could also be related to the sliding window size used in the Streaming DMD algorithm. Recalling that the nature of DMD is to decompose the spatial components and their corresponding frequency content or temporal evolution in the data, parameters like sampling frequency and window size used can be adjusted to capture dynamics at different time resolutions (Kutz et al. (2016)). This is comparable to selecting the window size in Wavelet theory. Thus, for slower dynamics, a larger window size can be used to capture full oscillations of a slowly varying signal while high frequent signals will require a finer time resolution, i.e. a more narrow window. In effect, for a persistent signal with dynamics at different timescales, the frequency components at each timescale can be retrieved by an appropriate choice of window size in the Sliding DMD method (Shu et al. (2020)). For this application however, the window size was selected such that the algorithm succeeded to both detect rapid and moderately slow movements of the objects adequately. Depending on the application, for instance detecting motion in traffic i.e. fast moving objects, this parameter of Streaming or Sliding DMD can be adjusted accordingly.

Overall, the DMD algorithm proves to be successful in detecting moving objects when the pixel contrast between the object and the background is large, which is the case for the black and green objects. In terms of the white objects, the reconstructed foreground is hardly showing any movement which results in negative predictions for the white objects. An attempt at increasing the motion sensitivity by adjusting the detection parameter to a lower threshold enabled detection of the white chair, but not the white boat. This is depicted in Figure 4.1.3 while the fail detection of the white boat is seen in Figure 4.1.4. Even so, increasing motion sensitivity by lowering the motion threshold is not a robust setting as it could make the algorithm more susceptible to noise such as light changes which should not be confused with the movement of an object. Another remark that may also explain why the boat was not detected with this setting is simply due to its smaller size compared to the white chair. Even with a lower motion threshold for the foreground pixels, a smaller object will have less pixels lighting up, and may therefore not be detected compared to a larger object on the same setting. This can be regarded as a criticism of the threshold used for detecting motion, as it is constrained to objects of fairly equal size, and not generalizable to all objects. A small object could for instance undertake a large movement, which could go undetected due to the low percentage of white foreground pixels. On the other hand, a large object, taking up more space in the frame, could make a larger impact with just a small movement.

Furthermore, an investigation of a better motion threshold to guard the binary classification problem of detecting motion, while being robust to noise is recommended. One option that does not suffer with the problem of object size above, and is more linked to the DMD theory, could be to threshold the DMD modes and their frequency content themselves. This approach is essentially what is proposed in Kutz et al. (2016) and enables motion detection of objects moving at different speeds.

For the change detection application presented in this report however, DMD does a more than adequate job in detecting moving objects that will trigger the sampling of the last moving frame, which is further passed on to the next step in the change detection pipeline, namely object detection.

Table 4.1.1: Motion detection using Streaming DMD in real time. Results with * are with a lower motion threshold

Detected motion (y/n)	**pure translation**	**pure 360° rotation**
black boat	y	y
green boat	y	y
white boat	n	n
black chair	y	y
green chair	y	y
white chair	n	n
white chair*	y	y
white boat*	n	n

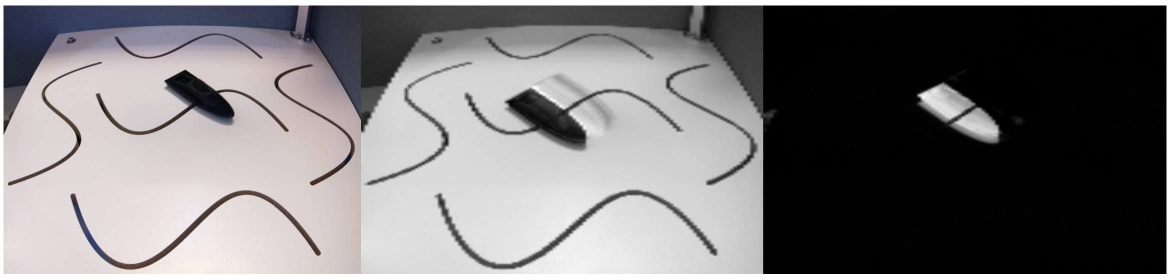


Figure 4.1.1: DMD motion detection of black boat detected

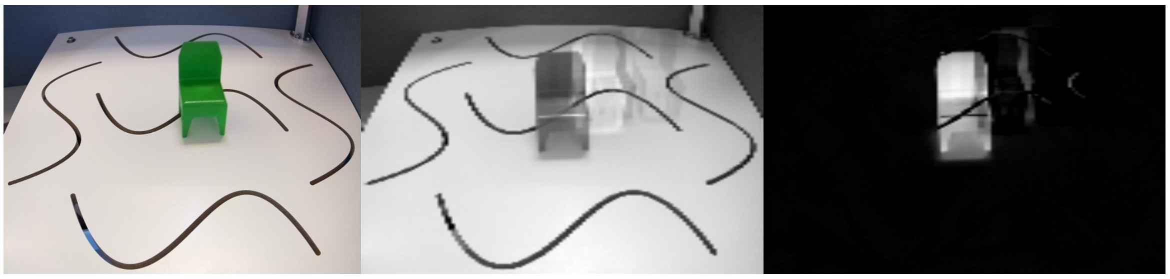


Figure 4.1.2: DMD motion detection of green chair

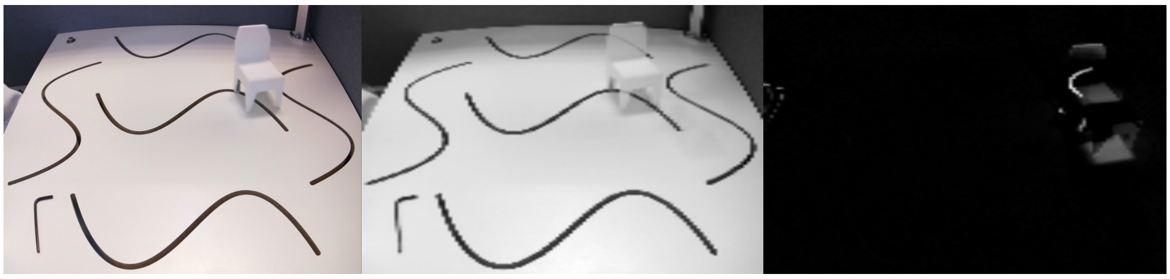


Figure 4.1.3: DMD motion detection of white chair with lower detection threshold

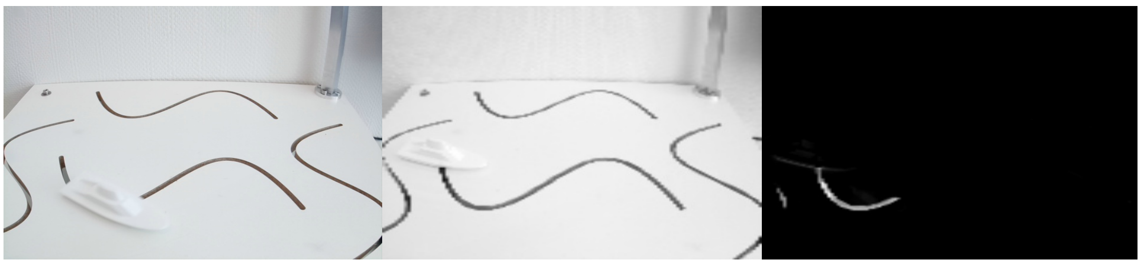


Figure 4.1.4: Images of white boat not detected even with lower detection threshold

4.2 Object detection and Image recognition

The preceding sections present the results obtained by the various image classifiers on the synthetically acquired data described in section 3.1. Final discussions on their respective performance and model nature are undertaken with the second research question presented in subsection 1.3.2 in mind:

- *Can synthetic data programmatically acquired in a virtual environment be used for training image classifiers so that they can be used in the real world with confidence ?*

4.2.1 CNN

Black figures

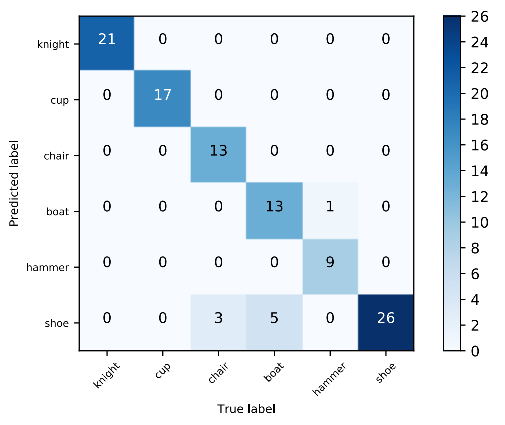
The confusion matrix for the real test set in Figure 4.2.1 reveals that the classifier performance is somewhat dependent on the object class. The boat is not recognised in any of its test samples, and is confused with the hammer and knight class. Both the hammer and boat exhibit an elongated shape, which is a probable factor for the confusion. This is also seen for the hammer, which is incorrectly classified as the boat in four of the samples.

Other similar objects in shape are the knight and chair, which is observed to be confused with each other in some of the samples as well. The object with the most unique shape in this selection is the cup, which could be an indication of why it is the only object that is recognised in all its samples in addition to not being subject to any false positives. The precision recall curves for each object, and the average over all classes in the bold blue line can be seen in Figure 4.2.2. Here, the confidence threshold for the prediction values is set to 0.5, which produces a single precision and recall point for each class.

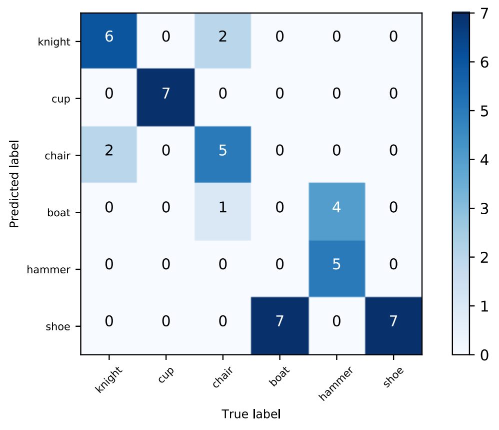
Furthermore, the performance metrics are summarised in Table 4.2.1 where it is seen that the obtained F1-score of 0.652 is below the desired level of a ideal classifier. Even so, as a preliminary result with no attempts for optimising the hyperparameters or architecture of the network, it is an indication that a model trained solely on synthetic data can exhibit expertise that is transferable to the real world. Especially when considering the plentiful possibilities of adapting the synthetic training data that is realisable through Python scripts. This is a topic for further research, as this trial merely concentrated on sampling synthetic images of the objects from different viewpoints.

Table 4.2.1: CNN performance on black test set

Classification accuracy (%)	65.20
Precision	0.65
Recall	0.65
Specificity	0.93
F1 score	0.65



(1a) Synthetic test set



(1b) Real test set

Figure 4.2.1: Confusion matrices for **black** figures on real images (right) and synthetic test set for comparison (left)

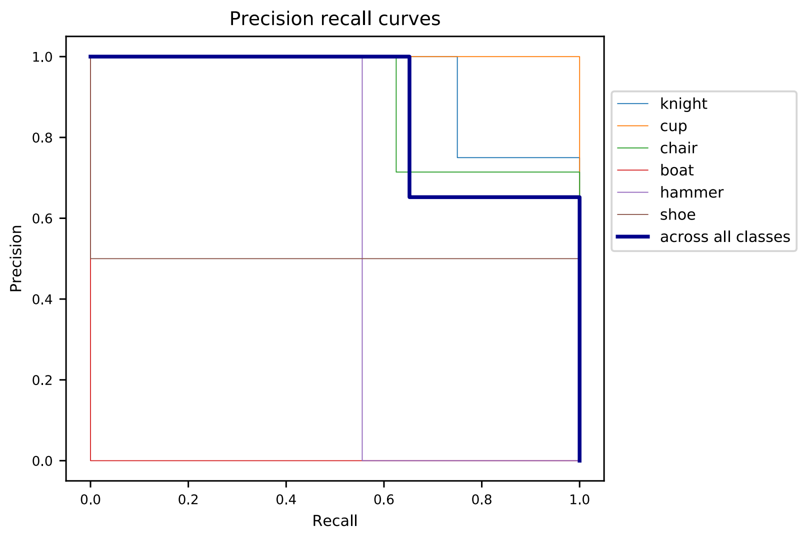


Figure 4.2.2: Precision recall curves for black test set

Green figures

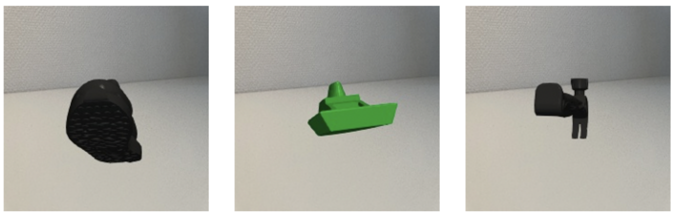
The results for the green figures are summarised in Table 4.2.2. Compared to the black figures, a slightly higher F1-score of 0.76 was obtained. A possible reason for this is that the test images captured of the green figures had more natural lighting, so no external light source was used. This resulted in less noise from the shadow associated with each object that the black objects were increasingly subject to. Another aspect could be that the test images of the green objects were acquired such that the objects stayed true to size compared to the synthetic data. This resulted in less cropping and resizing that will decrease the pixel granularity, producing an image of lower-quality and with less distinctive image features. The synthetic data did not contain zoomed in or out versions of the objects. This will likely result in a network that is sensitive to the object size in the test images, and the reason for the improved performance on the green test set, being more similar to the synthetic training data than the black objects.

Another remark that is connected to the black objects as well is regarding the

Table 4.2.2: CNN performance on green test set

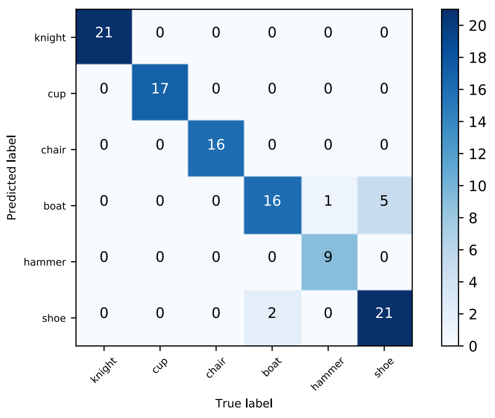
Classification accuracy (%)	76.00
Precision	0.76
Recall	0.76
Specificity	0.95
F1 score	0.76

sampling angles the synthetic data was acquired with. When the objects are rotated about the k-axis, as seen in Figure 3.1.2, and images are sampled at regular intervals specified by the $\Delta\phi$-angle, there will be some sampled images that provide a poor representation of the objects. Thus a good portion of the training data is missing out on important features describing the particular object, and may even present themselves as outliers disrupting the quality of the training data. This is mainly the case for the elongated objects as can be seen in Figure 4.2.3. As mentioned, this trial was primarily meant to be a preliminary study to examine the potential value of acquiring synthetic data using scripting techniques. The sampling angles was probably chosen a bit to rigorous, and examining the effects of removing some of these outliers could therefore be an interesting path to pursue.

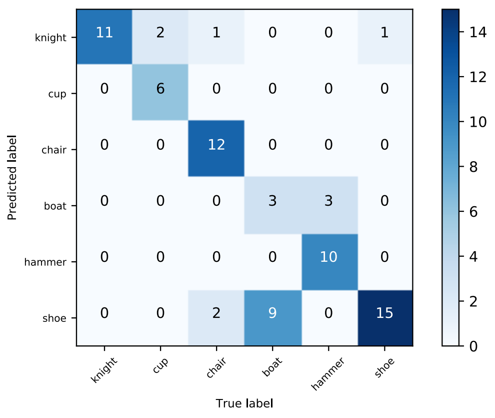


black shoe green boat black hammer

Figure 4.2.3: Examples of poor sampling angles for oblong objects



(4a) Synthetic test set



(4b) Real test set

Figure 4.2.4: Confusion matrices for **green** figures on real images (right) and synthetic test set for comparison (left)

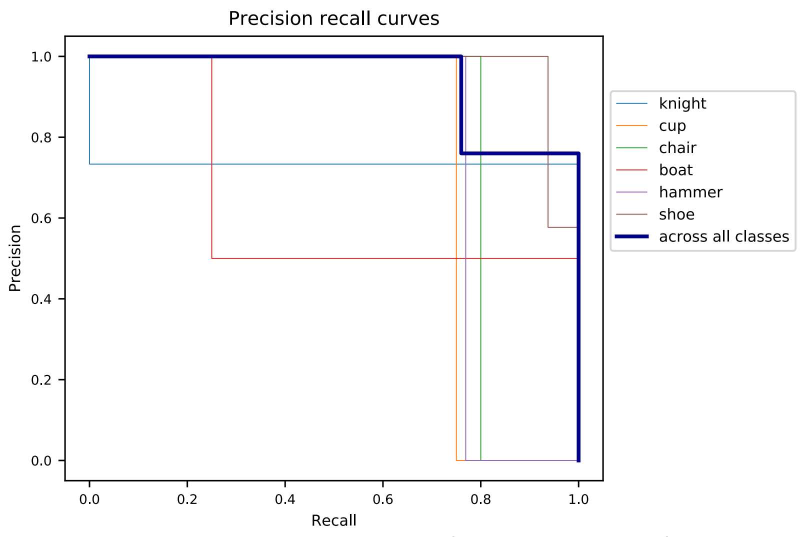


Figure 4.2.5: Precision recall curves for real test set of green objects

4.2.2 SRC

Black figures

The performance of SRC on the black objects are summarised in Table 4.2.3 and Table 4.2.4. With a prediction threshold of 0.5 and 0, respectively. When the threshold was set to 0.5, the SRC-classifier was able to obtain an accuracy of 75.0%. This lead to five "None"-predictions, or predictions with a too low SCI-score to be accepted as valid. Even so, six false positives resulted from this threshold. An investigation on finding the optimal threshold that obtains the fewest false negatives while achieving a sufficient classification accuracy could be relevant here. Even so, it should be noted that using a threshold based on the sparsity of the class coefficients is not a completely robust method of validating the predictions. As similar classes will share much of the same information or feature set, they become harder to discriminate. This can lead to sparsity in other classes than the true class, leading to a high SCI-score for the similar class and thus a false prediction. Methods that exploit this similarity in features or intra-class variability's like illumination, shape or occlusions for enriching the dictionary information, especially when training data is limited, is proposed in Deng et al. (2012*b*) and Peng et al. (2018).

Referring back to Table 4.2.4, the average SCI-score for all test samples was 0.73. This is a relatively high score which would indicate that the "certainty" or at least the discriminate nature of the predictions can be considered quite high. But with six false positives, it is likely that there have been false predictions with a high SCI-score, and that a threshold based on sparsity cannot effectively say anything for certain about the correctness of the prediction.

The distributions of class residuals and the corresponding sparse vector coefficients ($\mathbf{c}$) obtained for a correct prediction and a "None"-prediction can be seen in Figure 4.2.6. For the correct prediction, one of the training images in the dictionary obtains a very large weight. Since the weights from other class coefficients become negligible in comparison, there is a prominent smaller residual for the knight class and a high SCI-score. The "None"-prediction on the other hand illustrates the distribution of coefficients among several classes, which leads to a low SCI-score and high residuals across all classes. The query image for this prediction belongs to the boat class which is the class SRC seems to struggle the most with in this trial.

Similarly to the CNN, SRC interchanges some of the oblong objects as can be seen in Figure 4.2.8, likely due to their similarity in shape. As mentioned, it performs worst on the boat class, which is predicted as the shoe in five of its seven test samples, while the hammer is observed to be confused with the boat in three of its samples, and the shoe in one.

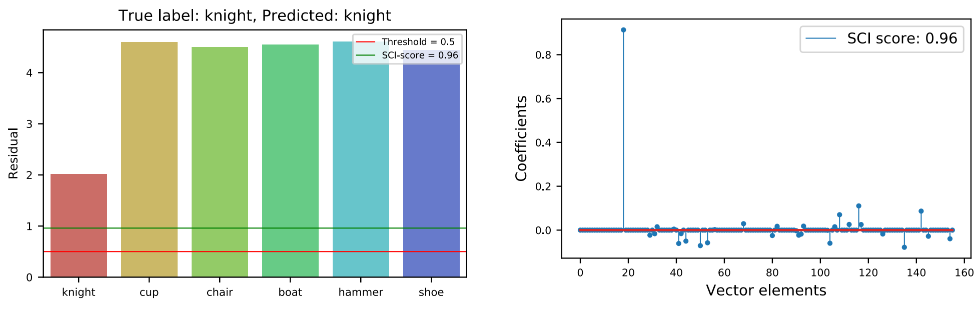
Results from a second run of the model with the threshold set to 0 is presented in Table 4.2.4. The same evaluation metrics have been used here as for the CNN as a threshold of 0 is in line with how a CNN performs predictions (the array index with the largest value in the softmax output). It is observed that the accuracy increases slightly, but at the expence of increased false positives of 10 predictions, which is expected.

Table 4.2.3: SRC performance on black test set and threshold set to 0.5

Classification accuracy (%)	75.0
False positives	6
True positives	33
None predictions	5
Avg. SCI-score	0.73

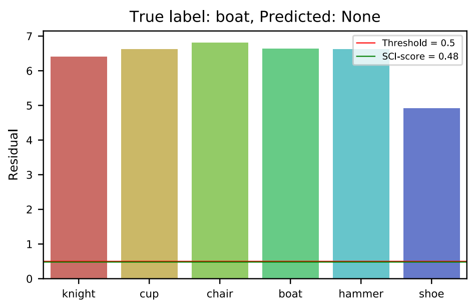
Table 4.2.4: SRC performance on black test set and threshold set to 0

Classification accuracy (%)	77.30
False positives	10
Precision	0.77
Recall	0.77
Specificity	0.95
F1 score	0.77

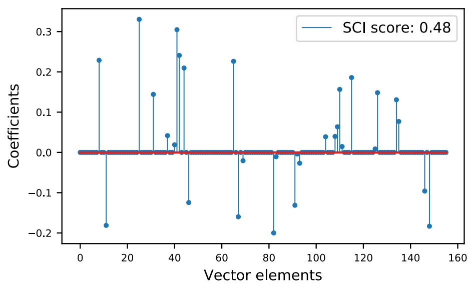


(6a) Class residuals **(6b)** Sparse coefficients

Figure 4.2.6: Example residuals and coefficients from black objects of a successful prediction

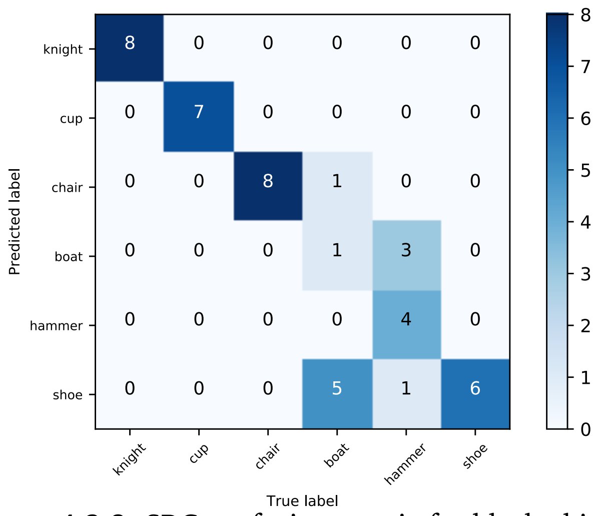


(7a) Class residuals



(7b) Non-sparse coefficients

Figure 4.2.7: Non-sparse coefficients leading to no predicted class due to threshold



Predicted \ True	knight	cup	chair	boat	hammer	shoe
knight	8	0	0	0	0	0
cup	0	7	0	0	0	0
chair	0	0	8	1	0	0
boat	0	0	0	1	3	0
hammer	0	0	0	0	4	0
shoe	0	0	0	5	1	6

Figure 4.2.8: SRC confusion matrix for black objects

Green figures

The table in 4.2.5 shows that with a threshold of 0.5 an accuracy of 68.96% was reached for the green objects, which is slightly poorer than the black objects. Another remark is that the average SCI-score is a bit lower than what was obtained for the black objects. An example of this is seen in Figure 4.2.9, where the coefficients looks to be somewhat clustered around three or more classes. This trend was seen for all test samples as well, indicating that the dictionary matrix for the green objects was less discriminative for predicting real world samples than the black objects. Another reason for why the coefficients may be more distributed is the fact that more principal components was used for down-sizing the data than what what used for the black objects. The size of the dictionary matrix, or rather how underdetermined the equation set in Equation 2.5.8, the more sparsity in the solution is encouraged. Thus, with more principal components, the dictionary matrix becomes less underdetermined, which softens this sparsity characteristic. Additionally, with a larger feature set, as a result of using more components, there will be more shared features across the classes as there will be more features to "choose" from.

Another remark that may explain the slightly poorer performance is that the background image used for acquiring the dataset in ParaView was taken with the exact same illumination conditions as the black test set. Even though the difference is slight, SRC is known to be sensitive to illumination variations (Cao et al. (2016), Hu et al. (2018)). It is therefore likely that this trait presents itself for the green objects, creating a more challenging classification problem.

By looking at Table 4.2.6, both the accuracy and the number of false positives increase with the threshold set to 0 while still falling slightly short of the black objects. This emphasises that it is not a too strict threshold value that is causing a slightly poorer accuracy, but rather the model, i.e. the dictionary matrix comprised of the training images.

Once again, the three oblong objects hammer, shoe and boat are interchanged as seen in Figure 4.2.4, with the poorest performance on the hammer class.

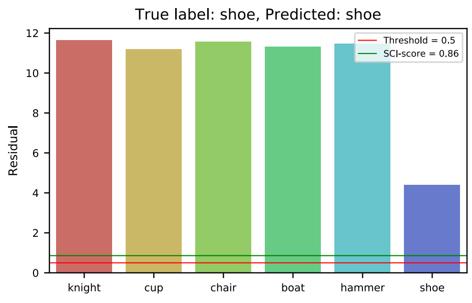
To summarize, the results could indicate that the SRC models were able to extract some valuable information from the synthetically acquired data that was transferable to the real world. Considering again that this was a rigorous trial where the approach of sampling the images was not fully examined. This could for instance imply experimenting with the synthetic illuminations to better fit the real world or creating an even larger image database with a more thorough representation of each class as well as experimenting with applying some filter to the training images to simulate noise present in the real world images. Yet, the results show that the SRC classifiers does an adequate job when introduced to the real world, and are able to distinguish classes to a certain extent.

Table 4.2.5: SRC performance on green test set and threshold set to 0.5

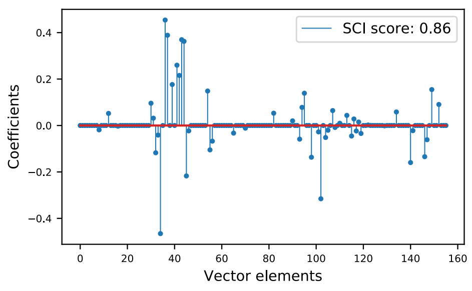
Classification accuracy (%)	68.96
False positives	11
True positives	40
None predictions	7
Avg. SCI-score	0.64

Table 4.2.6: SRC performance on green test set and threshold set to 0

Classification accuracy (%)	75.86
False positives	14
Precision	0.76
Recall	0.76
Specificity	0.95
F1 score	0.76



(9a) Class residuals



(9b) Sparse coefficients

Figure 4.2.9: Example residuals and coefficients for green objects of a successful prediction

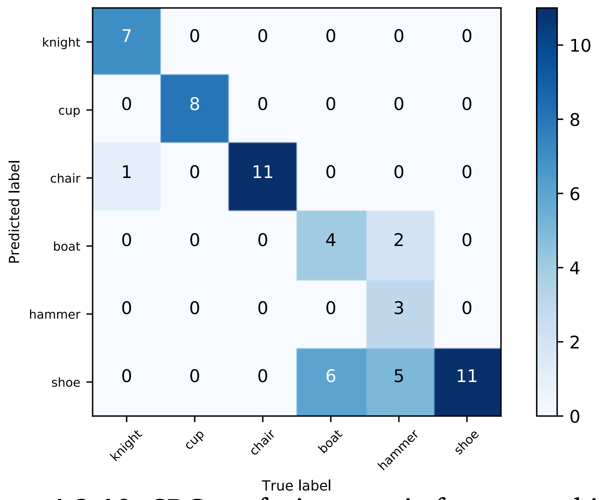


Figure 4.2.10: SRC confusion matrix for green objects

4.2.3 Yolo

The precision-recall curves from the test runs with synthetic and real data respectively can be seen in Figure 4.2.13. Interestingly, it is observed that the model performance on the real data has reached a satisfactory accuracy, with a mAP of 0.962. The green hammer and the black boat are the two object categories that yield the lowest performance here.

The highest F1-score of 0.94 on the synthetic data set is obtained already at confidence level 0.0, while for the real test set the F1-score peaks at confidence level 0.711 with a value of 0.93. Thus, the model is observed to be most accurate or perform best at this confidence level. By inspecting Figure 4.2.15 the precision curves of the green chair and boat produce the weakest scores, decreasing the precision across all classes until it reaches 1.0 at 0.939. This is the reason for the low F1-score seen for the green chair in Figure 4.2.13 for the real test data. As the training data is synthetic, there will be no natural variations in the data. This may explain why the F1-score for the synthetic test set is at its peak already at confidence level 0.0, as the model will not encounter sources leading to uncertainties other than which object is depicted. It is therefore sensible that the precision and recall values of the model are high already at low confidence levels because the environment of the test set is exactly the same as the training data.

By viewing the confusion matrix for the real test set in Figure 4.2.12 which is with a confidence of 0.25, it is observed that in 70% of the false positive predictions that are truly of the background, it is the green chair that is predicted. It therefore seems to be something in the background that yolo confuses with the green chair. This could possibly be the steel pole in the left corner of the scene which is a bit similar in shape and perhaps in color, and can be observed in Figure 4.2.11. This background object, could explain why the results for the green chair stands out from the rest of the classes in the precision plot.

In light of the results obtained by Yolov5 on the real test set, it can be concluded that the model performance is satisfactory and at a desirable level for an object detec-

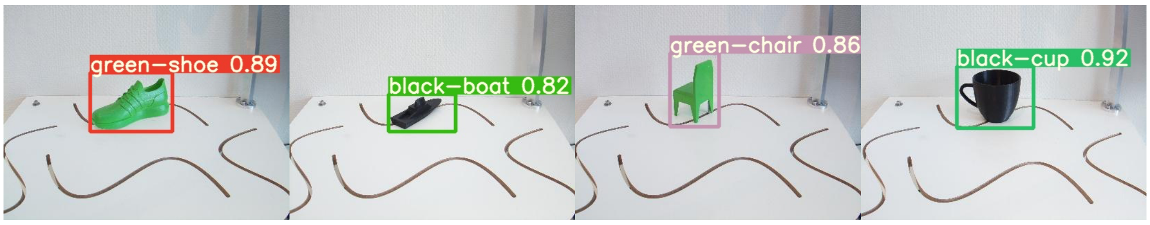


Figure 4.2.11: Examples of detected objects in the experimental setup

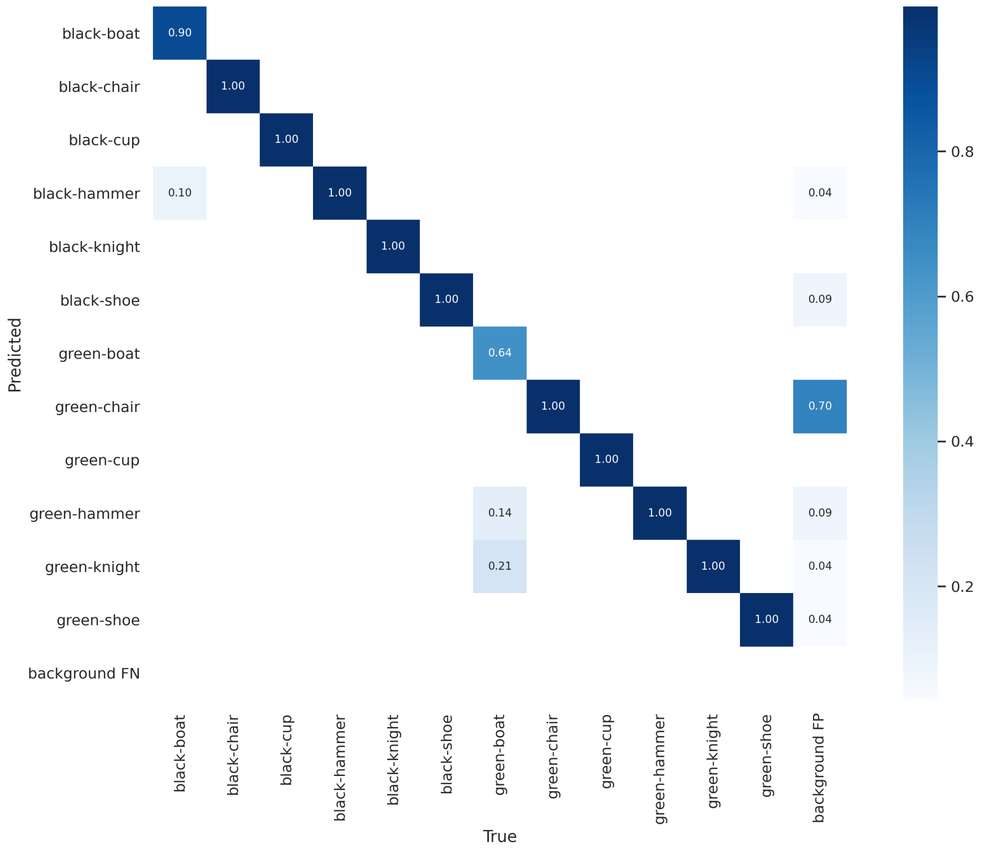
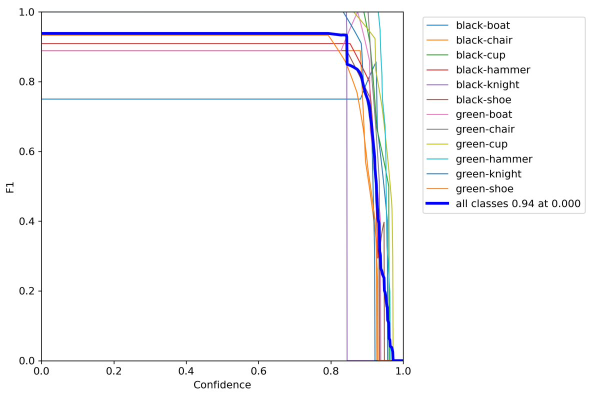
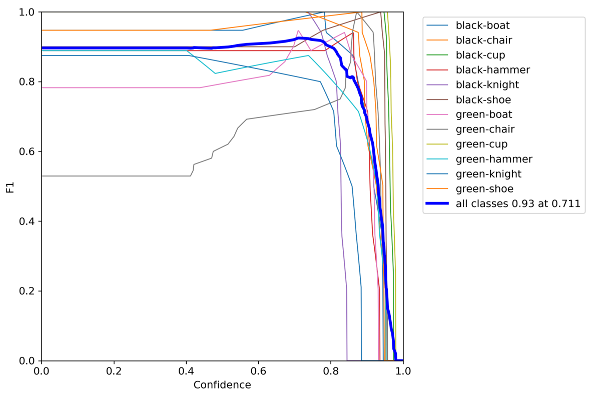


Figure 4.2.12: Confusion matrix for Yolo with confidence 0.25 of real test set

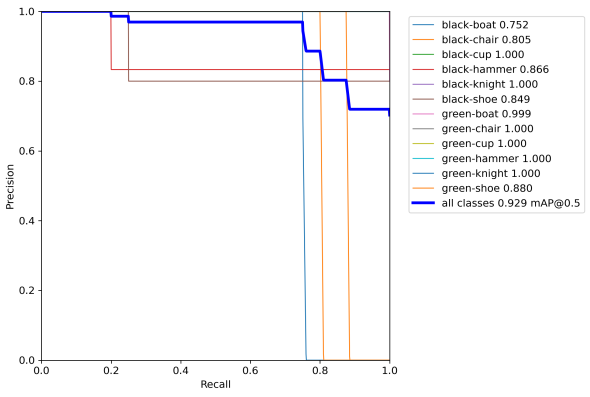


(13a) Test set with synthetic data

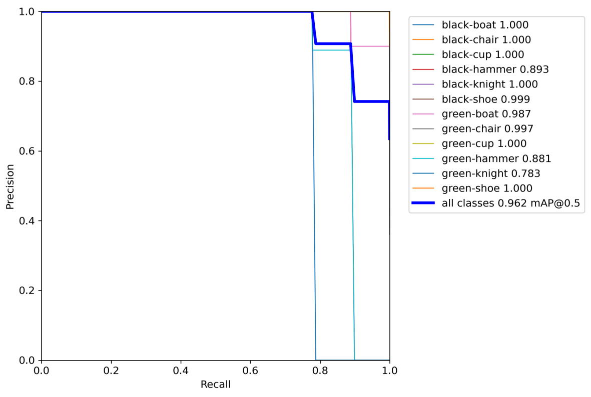


(13b) Test set with real data

Figure 4.2.13: F1-curves of test set from same distribution as training data, i.e. synthetic data, and test set of real data only



(14a) Test set with synthetic data



(14b) Test set with real data

Figure 4.2.14: Precision-Recall-curves of test set from same distribution as training data, i.e. synthetic data, and test set of real data only

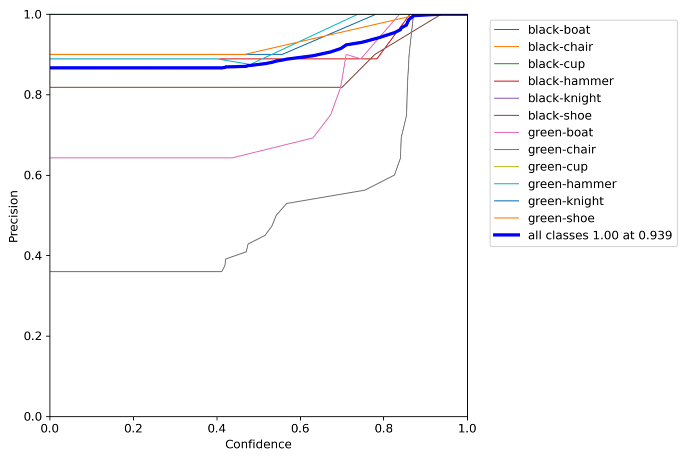


Figure 4.2.15: Green chair contributing to a low precision at lower confidence levels

4.2.4 Summary of all models

The initial findings from CNN and SRC point in a direction where both synthetically trained models were able to adequately classify the test samples acquired from a real world setting. Beyond the fact that both models seem to struggle with separating objects that are similar in shape, their performances are not really comparable as their respective synthetic datasets are too dissimilar as a result of the differing sampling strategies. With more conservative object rotations in the SRC dataset, it is likely that SRC had an easier job of recognising the query images than the CNN. Particularly based on the arguments regarding outlier data as previously discussed. Therefore, it cannot be established which model is more suited for use with synthetic training data, and real world test data.

Both models showed indications of being sensitive to the settings used at the time of data collection. Thus, the usefulness of leveraging synthetic data in training image classifiers as CNN, SRC as well as Yolo is not yet fully examined. Based on the observations in terms of outlier data for CNN, and illumination sensitivities for SRC, the following possible alterations to the synthetic data acquisition is proposed:

- Customize sampling angles to each object or objects similar in shape to avoid outlier data
- Sample images at different zoom-levels to avoid overfitting towards a fixed object size
- Depending on the application or real world scene, adapt the background the images are rendered in to several backgrounds as a means of regularizing the network to avoid overfitting towards a single background
- Alternate the object localization in the images such that they are not always centered in the middle
- Apply different illumination and texturing settings to the rendered objects

As for the Yolo model, the use of synthetic data for training proved to achieve a sufficiently powerful model also able to extract accurate bounding box estimates of the objects. Due to the requirement of tightly cropped images of each object as input to the Pose Estimation model, the model selection for performing this task of object recognition and localization in the proposed Change detection application was more suited for Yolo. Although SRC offer strengths in terms of interpretability, less computation load and no training necessary, it is not directly suited for the proposed change detection application. The reasons being sensitivity to large pose variations in addition to merely performing the task of image recognition.

It should be notet that Yolov5, belonging to the Yolo-family, is a powerful state-of-the-art object detection model exhibiting the robustness that is required for real-time object detection (Bochkovskiy et al. (2020)).This is likely to make it less sensitive to variations in data stemming from the real-world environment, as opposed to the synthetic environment in the training data. The discrepancy in performance seen between Yolo and the two other image classifiers might therefore be expected. Secondly, the synthetic dataset used for training Yolo contained a larger volume of images with more training samples for each class. Additionally, some image augmentation was applied to the training data as measures for mitigating overfitting as well as increasing model robustness. Lastly, the image size used to downsize the images was greater than for the CNN and SRC. This would lead to higher quality images, possibly enabling added or better image features learned by the network.

Furthermore, it is fair to say that the data volume used for training Yolo in this trial would have been significantly more tedious to produce if the images were to be acquired manually. By simply running a script, a sufficiently large dataset, completely customised to the application was produced. For the task of detecting positional changes of monitored objects, it is relevant to know the angles for which the objects are rotated in the dataset. This could for instance help with correctly annotating the dataset with the correct angles for use in the Pose Estimation model as this is exactly known in a synthetic world. If the same images were to be acquired manually, and with the same sampling angles, this would rely on human measurements. A process highly susceptible to uncertainty which would consequently be introduced into the application.

An interesting examination would be to see how the models perform when *combining* synthetic and real world data during training. If training data is scarce, this would be a valuable option for generating the necessary data to build a complete model. This could also make the model more robust when faced with for instance illumination changes in the test set that was seen for the green objects with SRC and CNN. As synthetic data can be adapted to fit any variation, the possibilities for enriching the training data with the features desired for some application are extensive.

A problem SRC and other image classifiers are faced with is imbalanced training data that results in a skewed representation of classes. This can make the classifier favor some classes over others simply because they are over-represented in the training data (Zou et al. (2018), Shu et al. (2020)). A highly useful application of synthetic training data, could therefore be to balance a skewed distribution of training images with synthetic data for the poorly represented classes.

With the rapid advancement of technology like virtual and augmented reality seeking to accurately mirror the real world, software for visualizing 3D-models and generating synthetic data will likely improve. Thus, the relevance of synthetic data applied to the real world is growing, and might not yet be fully exploited.

4.3 Pose Estimation

Proceeding with the change detection pipeline depicted in Figure 3.3.9, the resulting image crops given by the bounding box estimates produced by Yolov5 are further passed as input to the Pose Estimation module. Accompanying the cropped images of each object is its corresponding CAD model in .obj file format used for rendering views of the object as shown in subsection 3.3.3.

Once again, the object categories are isolated in terms of color such that experiments on the black and green objects are reported separately for comparison.

The metrics used for evaluating the Pose Estimation performance are the ones presented in subsection 3.3.3, namely $Acc_{\frac{\pi}{6}}$ and Med_{ERR}. The evaluated results are seen in Table 4.3.1 and Table 4.3.2. Additionally, the offset between the three predicted Euler angles and the true angles for each experiment is shown in Figure 4.3.1, Figure 4.3.2 and Figure 4.3.3.

With regard to the $Acc_{\frac{\pi}{6}}$ evaluation metric, there is only a small deviation between the green and black objects. Even so, the result for the green shoe category is seen to be slightly poorer than that of the black shoe, which results in a moderate decrease in the mean $Acc_{\frac{\pi}{6}}$ for the azimuth angle across all green classes. For both the elevation and in-plane rotation angle, the mean $Acc_{\frac{\pi}{6}}$ are fairly equal. The only significant deviation between the two colors is the the Med_{ERR} of the azimuth angle with the larger difference of $39°$ reported for the black objects. The deviation here seems to originate from mainly the black boat class and the black knight.

Overall, the results reported in Table 4.3.1 and Table 4.3.2, report poor accuracies and angle deviations for the elevation angles and azimuth angles in particular. The results for in-plane rotation on the other hand, are observed as obtaining a high $Acc_{\frac{\pi}{6}}$. The metric $Acc_{\frac{\pi}{6}}$ allows in general for angle deviations up to $30°$ within the same angle bin. The results presented here however, are based on angles centered in the angle bins seen in Figure 3.3.8 (e.g. $0°, 30°, 60°$). Thus, the results are rather based on angle deviations up to $15°$ in either direction from the ground truth angle. Yet, as even small in-plane rotation deviations can result in a completely altered pose, it may be more relevant to look at the Med_{ERR} for this angle estimate. With a result of 7.2 and 7.02 for the green and black objects respectively, and the application of Change detection for detecting even small changes in the objects in mind, this result may not suffice.

Revisiting the origin of this Pose Estimation model, namely the one presented in Xiao et al. (2019*a*), the results on novel categories with ObjectNet3D used for training reported in the paper can be compared with those obtained in this trial. They report a mean $Acc_{\frac{\pi}{6}}$ for novel categories of 62%. This is the mean accuracy across all three Euler angles. In this trial, the novel black and green objects obtain a mean accuracy across all three Euler angles of 61.3% and 59.3% respectively. This remark should only verify that the approach for using the model in this trial is in line with that of the origin of the model. Further, a high accuracy on novel categories might not be plausible unless measures are taken to adapt the model to fit the application. For this trial, these measures should have been to train the model on a custom dataset of the object categories used in this report. The original plan was to extend the idea of synthetic data acquisition for this part of the training data as well. Unfortunately, due to shortage in time as a result of recurring delays with finalizing the experimental setup the experiments were dependent on, this investigation had to be omitted from the scope of this report. It is however left as a note on a plausible measure for improving the performance of the Pose Estimation model used in this framework.

The resulting offset between the true and predicted angle for each experiment is seen in Figure 4.3.1. Every object undergoes 7-9 experiments, depending on the number of accurate bounding box estimates delivered by Yolov5. From the figure, it appears that around half of the predictions for several object categories are approaching a $170°$ azimuth angle offset. This seems to particularly affect the oblong objects like the hammer, boat and shoe. Upon creating a rendered view of the predicted poses for these experiments, it is observed that the Pose Estimation model flips the oblong objects such that they are turning the opposite way. In other words, with an addition to the azimuth angle of nearly $180°$. This effect can be seen in Figure 4.3.5. This suggests that the model might fail to extract the more finely tuned features of these objects that separate the front from the back. Consequently, it is somewhat random which orientation along their major principal axes the prediction lands on.

Table 4.3.1: Pose estimation results for black objects

	Black objects					
$Acc_{\frac{\pi}{6}}$	boat	chair	hammer	knight	shoe	**mean**
azi. (ϕ)	0.44	0.22	0.29	0.13	0.5	**0.32**
elev. (θ)	0.67	0.56	0.14	0.38	1.0	**0.55**
in-plane rot. (ψ)	1.0	1.0	0.86	1.0	1.0	**0.97**
Med_{ERR} ($\circ$)						
azi. (ϕ)	170.51	80.60	156.27	96.56	89.89	**118.77**
elev. (θ)	9.32	9.87	23.50	22.67	7.34	**14.54**
in-plane rot. (ψ)	6.91	6.78	7.39	6.50	7.54	**7.02**

Table 4.3.2: Pose estimation results for green objects

	Green objects					
$Acc_{\frac{\pi}{6}}$	boat	chair	hammer	knight	shoe	**mean**
azi. (ϕ)	0.38	0.25	0.25	0.13	0.38	**0.28**
elev. (θ)	0.88	0.5	0.13	0.25	1.0	**0.55**
in-plane rot. (ψ)	1.0	1.0	0.75	1.0	1.0	**0.95**
Med_{ERR} ($\circ$)						
azi. (ϕ)	96.53	82.19	74.65	52.55	90.67	**79.32**
elev. (θ)	8.41	14.16	23.81	22.67	7.45	**15.3**
in-plane rot. (ψ)	6.69	7.81	7.29	6.40	7.81	**7.2**

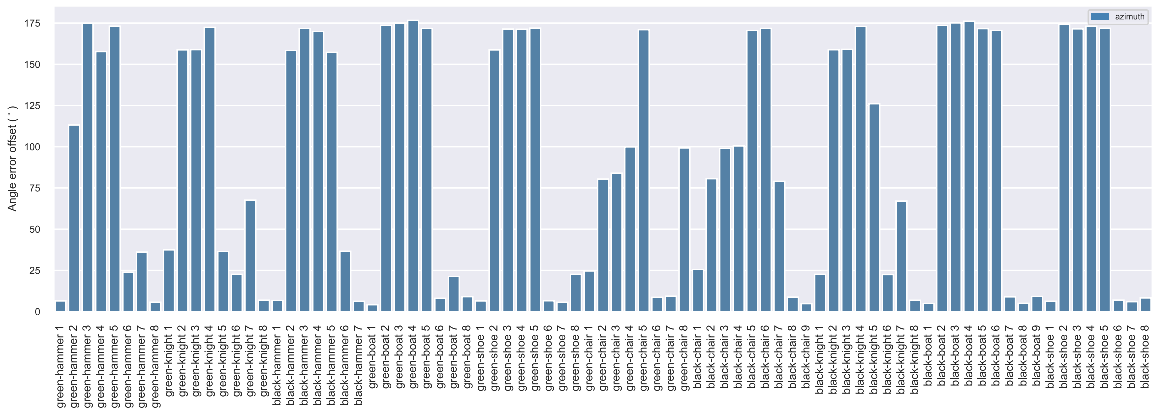


Figure 4.3.1: Offset of predicted azimuth angle and real angle

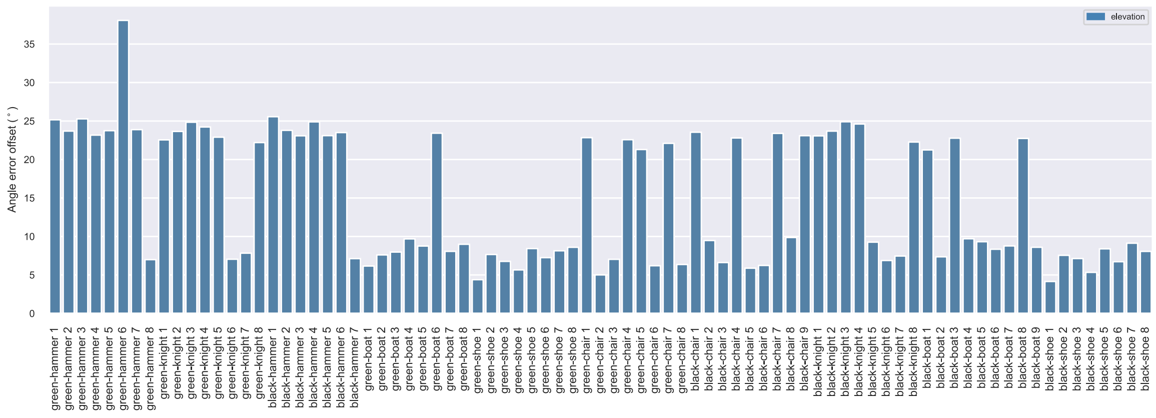


Figure 4.3.2: Offset of predicted elevation angle and real angle

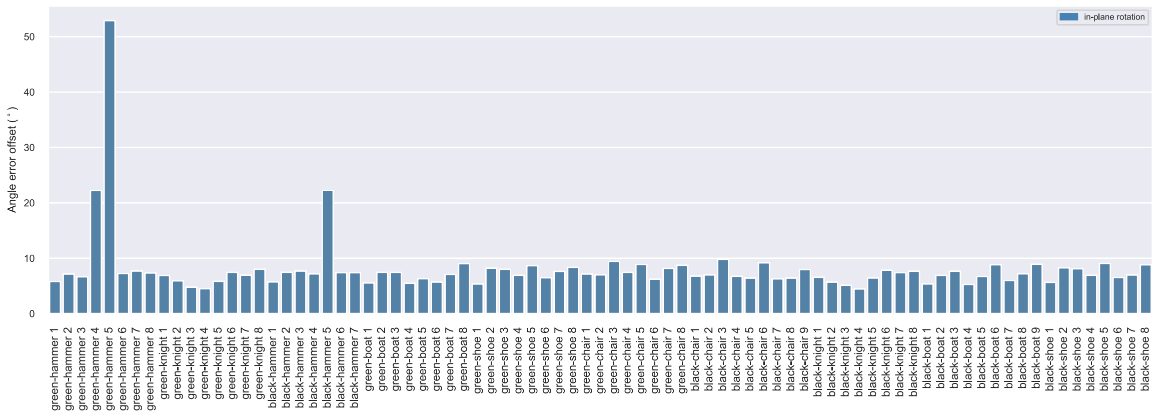


Figure 4.3.3: Offset of predicted in-plane rotation angle and real angle

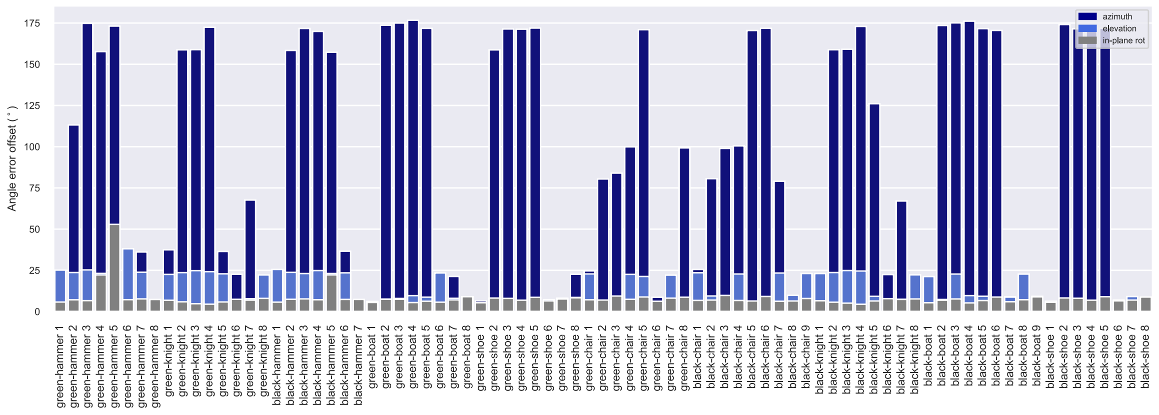


Figure 4.3.4: Offset of predicted angles and real angles

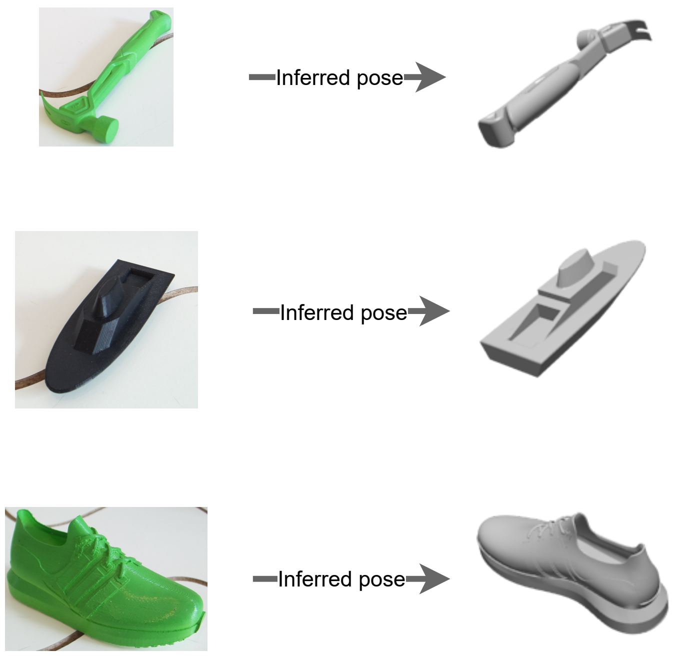


Figure 4.3.5: Inferred poses of elongated objects rendered in Blender

Chapter 5

Conclusion and future work

To reiterate, the research questions posed in the introduction to this report were the following

- With pedagogical purpose in mind, what kind of experimental setup can be built for the purpose of testing a cost-effective change detection approach in a Digital Twin ?
- Can synthetic data programmatically acquired in a virtual environment be used for training image classifiers so that they can be used in the real world with confidence ?
- How effective are DMD for motion detection, and SRC for image classification in comparison to Deep Learning ?

The first research question was answered by showing that the pipeline of change detection and its three modules could be properly tested with the experimental setup built in this project. Motion detection using DMD can be run in real-time on cheap hardware and obtained satisfactory results. The potential of using synthetic data for training YOLO showed another way of enhancing cost-effectiveness of the proposed approach since one can easily avoid the and time and labour intensive process of capturing and labelling real data. Although YOLO required training accelerated by GPU, once trained, inference was run with merely laptop-class computing power. Transferring the physically detected changes to reconstruct the digital scene on demand in a Digital Twin is dependent on an accurate Pose Estimation model. However, the pose estimation module, despite their ability to generalize to novel objects, still requires further improvements before they can be put to use in real use.

However, the overall change detection pipeline proved to be successful in terms of the cooperation between the modules which could be put together with the experimental setup. Approaches for improving the accuracy of the pose estimation model could for instance be to train the model on a custom dataset with the selected 3D-models, under the assumption that these are known, or simply try a more powerful model for performing pose estimation. A model that also estimates the translation part of the object movement could be relevant here. In terms of training the pose estimation model, a custom dataset could easily be generated and annotated with synthetic data, which would open for the investigation on the model performance in the real world.

The second research question regarding synthetic data acquisition was thoroughly tested on the two image classifiers CNN and SRC, as well as the object detection model Yolo. The two image classifiers obtained overall a performance on the same level, although with the remark that SRC had less pose variations in its training and test data. Even so, the concept of using synthetic data proved to produce two adequate models, and proposals for altering the data acquisition to possibly improve their performance was discussed. As for Yolo, the resulting model from using synthetic data proved to obtain a high mAP of 0.962 which is both satisfactory and impressive. The remark about Yolo being a more powerful model along with the added data augmentation and larger data volume used for training were listed as likely contributors to this promising result as compared to the two image classifiers.

With reference to the third research question, this report exemplifies possible applications of DMD and SRC. SRC has earlier proved to obtain state-of-the-art face recognition, and was seen to perform on the same level as CNN as an image classifier in this trial. Both DMD and SRC draw benefit from being cost-effective with sound mathematical foundations. DMD could be run on cheap hardware and managed to efficiently detect movement in the experimental setup. As opposed to Deep Learning models, SRC allows for complete transparency in the prediction as the prediction stems from a single optimization problem. Consequently, the computation load is decided by the size of the optimization problem, which for a moderate sized dictionary matrix, will result in a fast computation without the need for expensive GPUs. Additionally, SRC does not require the timely process of training. This is especially useful if an application requires an easy method of adding a new class to the image database, which would require expensive retraining of the model in the case of Deep Learning. For the Change Detection application, an introduction of a new object to the scene when using SRC, would merely require that training images of the new object class were appended to the dictionary matrix. Lastly, SRC has shown to perform adequately without the need of a large training data volume (Hu et al. (2018)) which could make it a valid candidate for applications where data is scarce. As the proposed change detection application required object localization as well as recognition for valid inputs to the pose estimation model, as well as Yolo obtaining a higher recognition accuracy than SRC, the latter model was not deemed fit this time. An idea could be to create bounding box estimates via the smallest rectangle circumventing all predicted foreground pixels from DMD. Further, the accuracy of SRC could possibly be improved by increasing the dictionary matrix to extend the representation of each object class, or the synthetic training data could be adapted to better coincide with the real world. These suggestions are however left out of the scope of this report.

5.1 Conclusions

The major conclusions of the thesis are:

- The experimental setup built proved to realize the proposed framework of detecting changes in a physical scene using a cost effective approach of motion detection with DMD, cheap hardware, lightweight Yolov5 for object detection and synthetic data acquisition for generating training data.

- Using synthetic data for training, and further use the trained model in the real world proved to obtain adequate models of CNN and SRC, while a satisfactory model with Yolo. The possibilities of adapting the scripting process and virtual scene are extensive, and the software for performing 3D-visualization and data acquisition may only improve with the innovations in augmented and virtual reality. Leveraging synthetic data as has been done in this report, is therefore yet to be exploited to its full extent.

- DMD and SRC have both shown to be computationally efficient methods with sound mathematical foundations. In this report, they have both shown to perform adequately in the application of motion detection and image recognition and could be recognized as competing methods with their respective strengths against state-of-the-art methods which are generally based on Deep Learning.

In doing so we answer all the research questions mentioned in 1.3.2 thereby realizing all the secondary and primary objectives

5.2 Future Work

Although the proposed approach of change detection was able to show the concept, several challenges were revealed and could light a spark for further investigations. Furthermore, findings and possible directions related to synthetic data acquisition emerged. Thus, the following remarks are left out as future work

- Alter the Pose Estimation module by either choosing a more robust and powerful model, or train the existing model with custom data acquired from the experimental setup

- Extend the change detection application to obtain an estimate of the translation component of each object in addition to the orientation. This could imply installing a RGB-D camera to the experimental setup. With the advert of sensor technology that have enabled the development of RGB-D cameras, pose estimation models have been proposed that exploit this additional dimension of depth, showing that there exists a community for this type of pose estimation models as well.

- Consider adjusting the process of synthetic data acquisition with different zoom levels, placements of the object, synthetic illuminations, colors and background variations. Furthermore, examine the effect of combining synthetic and real data when training to avoid overfitting towards the synthetic environment.

Bibliography

Ahrens, J., Geveci, B. and Law, C. (2005), ‘Paraview: An end-user tool for large data visualization’, *The visualization handbook* **717**(8).

Ayachit, U. (2015), *The paraview guide: a parallel visualization application*, Kitware, Inc.

Bajwa, I. S., Naweed, M., Asif, M. N. and Hyder, S. I. (2009), ‘Feature based image classification by using principal component analysis’, *ICGST Int. J. Graph. Vis. Image Process. GVIP* **9**, 11–17.

Berman, B. (2012), ‘3-d printing: The new industrial revolution’, *Business horizons* **55**(2), 155–162.

Bochkovskiy, A., Wang, C.-Y. and Liao, H.-Y. M. (2020), ‘Yolov4: Optimal speed and accuracy of object detection’, *arXiv preprint arXiv:2004.10934* .

Burger, J. E. and Gowen, A. A. (2015), Classification and prediction methods, *in* ‘Hyperspectral Imaging Technology in Food and Agriculture’, Springer, pp. 103–124.

Candès, E. J., Li, X., Ma, Y. and Wright, J. (2011), ‘Robust principal component analysis?’, *Journal of the ACM (JACM)* **58**(3), 1–37.

Cao, F., Hu, H., Lu, J., Zhao, J., Zhou, Z. and Wu, J. (2016), ‘Pose and illumination variable face recognition via sparse representation and illumination dictionary’, *Knowledge-Based Systems* **107**, 117–128.

Carrillo, R. E., Ramirez, A. B., Arce, G. R., Barner, K. E. and Sadler, B. M. (2016), ‘Robust compressive sensing of sparse signals: a review’, *EURASIP Journal on Advances in Signal Processing* **2016**(1), 108.

Choi, C. and Christensen, H. I. (2012), 3d pose estimation of daily objects using an rgb-d camera, *in* ‘2012 IEEE/RSJ International Conference on Intelligent Robots and Systems’, IEEE, pp. 3342–3349.

Chollet, F. et al. (2015), ‘Keras’, `https://keras.io`.

Community, B. O. (2018), *Blender - a 3D modelling and rendering package*, Blender Foundation, Stichting Blender Foundation, Amsterdam.
URL: *http://www.blender.org*

Cunico, F., Carletti, M., Cristani, M., Masci, F. and Conigliaro, D. (2019), 6d pose estimation for industrial applications, *in* M. Cristani, A. Prati, O. Lanz, S. Messelodi and N. Sebe, eds, 'New Trends in Image Analysis and Processing – ICIAP 2019', Springer International Publishing.

Deng, W., Hu, J. and Guo, J. (2012*a*), 'Extended src: Undersampled face recognition via intraclass variant dictionary', *IEEE Transactions on Pattern Analysis and Machine Intelligence* .

Deng, W., Hu, J. and Guo, J. (2012*b*), 'Extended src: Undersampled face recognition via intraclass variant dictionary', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **34**(9), 1864–1870.

Domahidi, A., Chu, E. and Boyd, S. (2013), ECOS: An SOCP solver for embedded systems, *in* 'European Control Conference (ECC)', pp. 3071–3076.

Donoho, D. L. (2006), 'For most large underdetermined systems of linear equations the minimal 1-norm solution is also the sparsest solution', *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences* **59**(6), 797–829.

Donoho, D. L. and Tsaig, Y. (2008), 'Fast solution of ℓ_1-norm minimization problems when the solution may be sparse', *IEEE Transactions on Information Theory* **54**(11), 4789–4812.

Dugelay, J.-L., Baskurt, A. and Daoudi, M. (2008), *3D object processing: compression, indexing and watermarking*, John Wiley & Sons.

Erichson, N. B., Brunton, S. L. and Kutz, J. N. (2019), 'Compressed dynamic mode decomposition for background modeling', *Journal of Real-Time Image Processing* **16**(5), 1479–1492.

Gallier, J. and Gallier, J. H. (2000), *Curves and surfaces in geometric modeling: theory and algorithms*, Morgan Kaufmann.

Ge, D., Jiang, X. and Ye, Y. (2011), 'A note on the complexity of l p minimization', *Mathematical programming* **129**(2), 285–299.

Girshick, R. (2015), Fast r-cnn, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 1440–1448.

Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2014), Rich feature hierarchies for accurate object detection and semantic segmentation, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 580–587.

Groover, M. and Zimmers, E. (1983), *CAD/CAM: computer-aided design and manufacturing*, Pearson Education.

Grosek, J. and Kutz, J. N. (2014), 'Dynamic mode decomposition for real-time background/foreground separation in video'.

Günther, F. and Fritsch, S. (2010), 'neuralnet: Training of neural networks', *The R journal* **2**(1), 30–38.

Hattab, A. and Taubin, G. (2015), 3d modeling by scanning physical modifications, *in* '2015 28th SIBGRAPI Conference on Graphics, Patterns and Images', IEEE, pp. 25–32.

He, K. and Sun, J. (2015), Convolutional neural networks at constrained time cost, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 5353–5360.

He, K., Zhang, X., Ren, S. and Sun, J. (2016), Deep residual learning for image recognition, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 770–778.

Hu, C.-H., Lu, X.-B., Liu, P., Jing, X.-Y. and Yue, D. (2018), 'Single sample face recognition under varying illumination via qrcp decomposition', *IEEE Transactions on Image Processing* **28**(5), 2624–2638.

Iancu, C. (2018), 'About 3d printing file formats.', *Annals of the Constantin Brancusi University of Targu Jiu-Letters & Social Sciences Series* .

Jocher, G., Stoken, A., Borovec, J., NanoCode012, ChristopherSTAN, Changyu, L., Laughing, tkianai, Hogan, A., lorenzomammana, yxNONG, AlexWang1900, Diaconu, L., Marc, wanghaoyang0106, ml5ah, Doug, Ingham, F., Frederik, Guilhen, Hatovix, Poznanski, J., Fang, J., , L. Y., changyu98, Wang, M., Gupta, N., Akhtar, O., PetrDvoracek and Rai, P. (2020), 'ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements'.
URL: *https://doi.org/10.5281/zenodo.4154370*

Jolliffe, I. T. and Cadima, J. (2016), 'Principal component analysis: a review and recent developments', *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **374**(2065), 20150202.

Kendall, A., Grimes, M. and Cipolla, R. (2015), Posenet: A convolutional network for real-time 6-dof camera relocalization, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 2938–2946.

Kulchandani, J. S. and Dangarwala, K. J. (2015), Moving object detection: Review of recent research trends, *in* '2015 International Conference on Pervasive Computing (ICPC)', pp. 1–5.

Kutz, J., Erichson, N., Askham, T., Pendergrass, S. and Brunton, S. (2017), Dynamic mode decomposition for background modeling, *in* '16th IEEE International Conference on Computer Vision Workshops, ICCVW 2017', Institute of Electrical and Electronics Engineers Inc., pp. 1862–1870.

Kutz, J. N., Fu, X. and Brunton, S. L. (2016), 'Multiresolution dynamic mode decomposition', *SIAM Journal on Applied Dynamical Systems* **15**(2), 713–735.

Labbé, Y., Carpentier, J., Aubry, M. and Sivic, J. (2020), Cosypose: Consistent multi-view multi-object 6d pose estimation, *in* 'European Conference on Computer Vision', Springer, pp. 574–591.

LeCun, Y., Bengio, Y. and Hinton, G. (2015), 'Deep learning', *Nature* **521**(7553), 436–444.

Lee, H.-J. and Hong, K.-S. (2012), Class-specific weighted dominant orientation templates for object detection, *in* 'Asian Conference on Computer Vision', Springer, pp. 97–110.

Li, Y., Wang, G., Ji, X., Xiang, Y. and Fox, D. (2018), Deepim: Deep iterative matching for 6d pose estimation, *in* 'Proceedings of the European Conference on Computer Vision (ECCV)', pp. 683–698.

Li, Z., Wang, Y. and Ji, X. (2019), 'Monocular viewpoints estimation for generic objects in the wild', *IEEE Access* **7**, 94321–94331.

Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X. and Pietikäinen, M. (2020), 'Deep learning for generic object detection: A survey', *International journal of computer vision* **128**(2), 261–318.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y. and Berg, A. C. (2016), Ssd: Single shot multibox detector, *in* 'European conference on computer vision', Springer, pp. 21–37.

Lowe, D. G. (2004), 'Distinctive image features from scale-invariant keypoints', *International journal of computer vision* **60**(2), 91–110.

Mairal, J., Elad, M. and Sapiro, G. (2007), 'Sparse representation for color image restoration', *IEEE Transactions on image processing* **17**(1), 53–69.

Mikołajczyk, A. and Grochowski, M. (2018), Data augmentation for improving deep learning in image classification problem, *in* '2018 international interdisciplinary PhD workshop (IIPhDW)', IEEE, pp. 117–122.

Nadeem, U., Bennamoun, M., Togneri, R. and Sohel, F. (2020), 'Unconstrained matching of 2d and 3d descriptors for 6-dof pose estimation', *arXiv preprint arXiv:2005.14502* .

Nathan Kutz, J., Benjamin Erichson, N., Askham, T., Pendergrass, S. and Brunton, S. L. (2017), Dynamic mode decomposition for background modeling, *in* 'Proceedings of the IEEE International Conference on Computer Vision Workshops', pp. 1862–1870.

O'Shea, K. and Nash, R. (2015), 'An introduction to convolutional neural networks'.

Park, K., Patten, T. and Vincze, M. (2019), Pix2pose: Pixel-wise coordinate regression of objects for 6d pose estimation, *in* 'Proceedings of the IEEE/CVF International Conference on Computer Vision', pp. 7668–7677.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. and Chintala, S. (2019), Pytorch: An imperative style, high-performance deep learning library, *in* H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, eds, 'Advances in Neural Information Processing Systems 32', Curran Associates, Inc., pp. 8024–8035.
URL: *http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf*

Peng, S., Liu, Y., Huang, Q., Zhou, X. and Bao, H. (2019), Pvnet: Pixel-wise voting network for 6dof pose estimation, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition', pp. 4561–4570.

Peng, Y., Li, L., Liu, S., Li, J. and Wang, X. (2018), 'Extended sparse representation-based classification method for face recognition', *Machine Vision and Applications* **29**(6), 991–1007.

Perez, L. and Wang, J. (2017), 'The effectiveness of data augmentation in image classification using deep learning', *arXiv preprint arXiv:1712.04621* .

Qi, C. R., Su, H., Mo, K. and Guibas, L. J. (2017), Pointnet: Deep learning on point sets for 3d classification and segmentation, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 652–660.

Qi, Q. and Tao, F. (2018), 'Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison', *Ieee Access* **6**, 3585–3593.

Ramachandran, P., Zoph, B. and Le, Q. V. (2017), 'Searching for activation functions', *arXiv preprint arXiv:1710.05941* .

Rasheed, A., San, O. and Kvamsdal, T. (2019), 'Digital twin: Values, challenges and enablers'.

Redmon, J. (2013), 'Darknet: Open source neural networks in c'.

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2016), You only look once: Unified, real-time object detection, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 779–788.

Ren, S., He, K., Girshick, R. and Sun, J. (2016), 'Faster r-cnn: towards real-time object detection with region proposal networks', *IEEE transactions on pattern analysis and machine intelligence* **39**(6), 1137–1149.

Shaikh, S. H., Saeed, K. and Chaki, N. (2014), Moving object detection using background subtraction, *in* 'Moving Object Detection Using Background Subtraction', Springer, pp. 15–23.

Sharma, S. (2017), 'Activation functions in neural networks', *towards data science* **6**.

Shorten, C. and Khoshgoftaar, T. M. (2019), 'A survey on image data augmentation for deep learning', *Journal of Big Data* **6**(1), 60.

Shu, T., Zhang, B. and Tang, Y. (2020), 'Sparse supervised representation-based classifier for uncontrolled and imbalanced classification', *IEEE Transactions on Neural Networks and Learning Systems* **31**, 2847–2856.

Själander, M., Jahre, M., Tufte, G. and Reissmann, N. (2019), 'EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure'.

Su, H., Qi, C. R., Li, Y. and Guibas, L. J. (2015), Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views, *in* 'Proceedings of the IEEE International Conference on Computer Vision', pp. 2686–2694.

Sundby, T., Graham, J. M., Rasheed, A., Tabib, M. and San, O. (2021), 'Geometric change detection in digital twins', *Digital* **1**(2), 111–129.

Tao, F., Zhang, H., Liu, A. and Nee, A. Y. (2018), 'Digital twin in industry: State-of-the-art', *IEEE Transactions on Industrial Informatics* **15**(4), 2405–2415.

Tao, F., Zhang, H., Liu, A. and Nee, A. Y. C. (2019), 'Digital twin in industry: State-of-the-art', *IEEE Transactions on Industrial Informatics* **15**, 2405–2415.

Tu, J. H., Rowley, C. W., Luchtenburg, D. M., Brunton, S. L. and Kutz, J. N. (2013), 'On dynamic mode decomposition: Theory and applications', *arXiv preprint arXiv:1312.0041* .

Tulsiani, S. and Malik, J. (2015), Viewpoints and keypoints, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 1510–1519.

Tzeng, F.-Y. and Ma, K.-L. (2005), *Opening the black box-data driven visualization of neural networks*, IEEE.

Wanasinghe, T. R., Wroblewski, L., Petersen, B. K., Gosine, R. G., James, L. A., De Silva, O., Mann, G. K. I. and Warrian, P. J. (2020), 'Digital twin for the oil and gas industry: Overview, research trends, opportunities, and challenges', *IEEE Access* **8**, 104175–104197.

Wang, Y., Jodoin, P.-M., Porikli, F., Konrad, J., Benezeth, Y. and Ishwar, P. (2014), Cdnet 2014: An expanded change detection benchmark dataset, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition workshops', pp. 387–394.

Wright, J., Yang, A. Y., Ganesh, A., Sastry, S. S. and Ma, Y. (2008), 'Robust face recognition via sparse representation', *IEEE transactions on pattern analysis and machine intelligence* **31**(2), 210–227.

Wu, X., Sahoo, D. and Hoi, S. C. (2020), 'Recent advances in deep learning for object detection', *Neurocomputing* **396**, 39–64.

Xiang, Y., Kim, W., Chen, W., Ji, J., Choy, C., Su, H., Mottaghi, R., Guibas, L. and Savarese, S. (2016), Objectnet3d: A large scale database for 3d object recognition, *in* 'European conference on computer vision', Springer, pp. 160–176.

Xiang, Y., Mottaghi, R. and Savarese, S. (2014), Beyond pascal: A benchmark for 3d object detection in the wild, *in* ‘IEEE Winter Conference on Applications of Computer Vision’, pp. 75–82.

Xiang, Y., Schmidt, T., Narayanan, V. and Fox, D. (2017), ‘Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes’, *arXiv preprint arXiv:1711.00199* .

Xiao, Y., Qiu, X., Langlois, P., Aubry, M. and Marlet, R. (2019*a*), Pose from shape: Deep pose estimation for arbitrary 3D objects, *in* ‘British Machine Vision Conference (BMVC)’.

Xiao, Y., Qiu, X., Langlois, P., Aubry, M. and Marlet, R. (2019*b*), ‘Pose from shape: Deep pose estimation for arbitrary 3d objects’, *CoRR* **abs/1906.05105**.
URL: *http://arxiv.org/abs/1906.05105*

Xu, B., Guo, P. and Chen, C. P. (2013), Kernel based weighted group sparse representation classifier, *in* ‘International Conference on Human-Computer Interaction’, Springer, pp. 236–245.

Yin, J., Liu, Z., Jin, Z. and Yang, W. (2012), ‘Kernel sparse representation based classification’, *Neurocomputing* **77**(1), 120–128.

Zhang, B., Ji, S., Li, L., Zhang, S. and Yang, W. (2016), ‘Sparsity analysis versus sparse representation classifier’, *Neurocomputing* **171**, 387–393.
URL: *https://www.sciencedirect.com/science/article/pii/S092523121500898X*

Zhang, H., Rowley, C. W., Deem, E. A. and Cattafesta, L. N. (2019), ‘Online dynamic mode decomposition for time-varying systems’, *SIAM Journal on Applied Dynamical Systems* **18**(3), 1586–1609.

Zhang, H., Zhang, Y. and Huang, T. S. (2013), ‘Pose-robust face recognition via sparse representation’, *Pattern Recognition* **46**(5), 1511–1521.

Zheng, Y., Yang, S. and Cheng, H. (2019), ‘An application framework of digital twin and its case study’, *Journal of Ambient Intelligence and Humanized Computing* **10**(3), 1141–1153.

Zou, X., Feng, Y., Li, H. and Jiang, S. (2018), ‘Improved over-sampling techniques based on sparse representation for imbalance problem’, *Intelligent Data Analysis* **22**(5), 939–958.