

Claire Trinquet

Pepper as an assistant in the library: Identifying books using machine learning

Master's thesis in Applied Computer Science

Supervisor: Deepti Mishra

June 2023

Claire Trinquet

Pepper as an assistant in the library: Identifying books using machine learning

Master's thesis in Applied Computer Science
Supervisor: Deepti Mishra
June 2023

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



Norwegian University of
Science and Technology

Acknowledgement

I would like to express my deepest gratitude to my supervisor, Deepti Mishra for her guidance and support throughout the entire process of writing this master thesis. Her feedback and patience have been an invaluable help for this research, and I am very thankful for all I have learned from her.

I would also like to thank my family and my friends during these two years. I am profoundly grateful for your support and encouragement which have been a constant source of strength and motivation during these two years.

Claire Trinquet

Abstract

With the development of new technologies in the last decades, robots are more present than ever in our daily lives. Social robots are an example of this. They are humanoid robots, designed for social interactions with humans and other robots. Another example of this is robots being used in libraries, to help locate books and automate processes. However, these library robots don't have social interactions with people the way social robots do, and social robots themselves lack the sensors that enable other robots to detect and identify books. In order to explore the possibility of using a social robot in the library, this master thesis focuses on developing a way for a social robot, Pepper, to detect books in its environment. The objective is to combine Pepper's camera with Computer Vision techniques such as object detection and OCR in order to enable the robot to read the titles of books in front of it. Several models and parameters were compared in the experiment, and the findings show that EasyOCR is the most accurate model for this task. Although the results of the OCR do not fit perfectly the text written on the books, the error rate is low enough for it to be recognizable. This study presents a new way for Pepper to learn from its environment, and can serve as a basis for future work related to Pepper being used in the library.

Sammendrag

Roboter er mer til stedet enn på noe tidligere tidspunkt gitt den hyppige teknologiske utviklingen de siste tiårene. Sosiale roboter er et eksempel på dette. De er humanoide roboter, designet for sosial interaksjon med mennesker og andre roboter. Andre roboter, som bibliotekroboter, er designet for automatiserte prosesser og bokidentifisering. Biblioteksrobotene er derimot ikke designet for sosiale interaksjoner på likt vis som sosiale roboter. Denne masteroppgaven ser på muligheten for å bruke sosiale roboter, som mangler sensoren som muliggjør identifisering av bøker, i biblioteker. For å utforske muligheten for å bruke en sosial robot i biblioteket, fokuserer denne masteroppgaven på å utvikle en måte for en sosial robot, Pepper, å oppdage bøker i sitt miljø. Målet er å kombinere Peppers kamera med Computer Vision-teknikker som objekt-deteksjon og OCR. Dette for å gjøre det mulig for roboten å lese titlene på bøkene foran seg. Flere modeller og parametere ble sammenlignet i forsøket, og funnene viser at EasyOCR er den mest nøyaktige modellen. Selv om resultatene av OCR ikke passer perfekt til teksten som er skrevet på bøkene, er feilraten lav nok til at den er gjenkjennelig. Denne studien presenterer en ny måte for Pepper å lære av omgivelsene sine, og kan tjene som grunnlag for fremtidig arbeid knyttet til at Pepper brukes i biblioteket.

Contents

Acknowledgement	iii
Abstract	v
Sammendrag	vii
Contents	ix
Figures	xi
Tables	xiii
Code Listings	xv
1 Introduction	1
1.1 Keywords	2
1.2 Problem statement	2
1.3 Research Questions	2
1.4 Contributions	3
1.5 Outline of Chapters	3
2 Related Works	5
2.1 Robotics and social robotics	5
2.2 Robots in libraries	6
2.3 Object Detection and Optical Character Recognition	7
2.3.1 Object detection in real time	7
2.3.2 OCR or Optical Character Recognition	9
2.3.3 OCR accuracy	10
3 Methodology	11
3.1 The Pepper Robot	11
3.2 Design of the scenario	12
3.3 Experiments	13
3.3.1 Procedure	14
3.3.2 Parameters	15
3.3.3 Data collection	17
4 Implementation	21
4.1 Experiments	21
4.1.1 TakePictures	21
4.1.2 CropPictures	23
4.1.3 UseOCR	25
4.1.4 Accuracy	28
4.2 Scenario	29

5 Results	33
5.1 Object detection	33
5.2 OCR	35
5.2.1 Parameters of the experiment	36
5.2.2 OCR results across books	42
6 Discussion	45
6.1 Integration of machine learning with Pepper	45
6.2 Object detection and OCR	46
6.3 Scenario	46
6.4 Limitations	48
6.5 Implications	48
7 Conclusion and Future Work	49
Bibliography	51

Figures

2.1	An example of industrial robots and humanoid robots	5
2.2	Results of object detection on a photo of a desk	8
3.1	The Pepper robot	12
3.2	The process to identify a book in the application with the Pepper robot	13
3.3	Example of the process explained previously: Taking a photo with Pepper and running the object detector model	14
3.4	The previous photo from figure 3.3, cropped to only show the book at the top of the pile	14
3.5	The books used for the experiments	15
3.6	A photo taken by the Pepper robot with its top camera and resolution of 4	16
3.7	Photos of the same pile of book, taken with resolution 3 (left) and resolution 6 (right)	17
5.1	The number of detection results	33
5.2	The number of OCR results	35
5.3	The average CER, separated according to OCR model	36
5.4	The average CER, separated according to resolution	37
5.5	The average CER, separated according to distance	38
5.6	The average CER, separated according to distance and resolution	39
5.7	The average CER, separated according to OCR model and distance	40
5.8	The average CER, separated according to OCR model and resolution	40
5.9	The average CER, separated according to OCR model, distance and resolution	42
5.10	The average CER, separated according to book	43

Tables

3.1	The four categories of detection results	19
3.2	The books and the associated expected result for each	20
5.1	The detection results	34
5.2	The detection results, separated according to resolution	34
5.3	The detection results, separated according to resolution and distance	35
5.4	The OCR results, separated according to OCR model	36
5.5	The OCR results, separated according to resolution	37
5.6	The OCR results, separated according to distance between the robot and the books	38
5.7	The OCR results, separated according to distance and resolution . .	38
5.8	The OCR results, separated according to OCR model and distance .	39
5.9	The OCR results, separated according to OCR model and resolution	40
5.10	The OCR results, separated according to OCR model, distance and resolution	41
5.11	The OCR results, separated according to book	43
6.1	The CER results from example 1	47

Code Listings

4.1	Connection to Pepper	21
4.2	Taking a picture with Pepper	22
4.3	Transferring the files to the computer	23
4.4	The main part of the <i>CropPictures</i> file	23
4.5	The <i>detect</i> function	24
4.6	The <i>cropPicture</i> function	24
4.7	The main loop of <i>UseOCR.py</i>	25
4.8	The <i>pytess</i> function	25
4.9	The <i>easyOCR</i> function	26
4.10	The <i>get_distance</i> , <i>distinguish_rows</i> and <i>sorting</i> functions	26
4.11	The <i>kerasOCR</i> function	28
4.12	The <i>Accuracy</i> file	28
4.13	Use of a subprocess in <i>scenarioMain.py</i> for the object detection . . .	30
4.14	The <i>detector.py</i> file	30
4.15	Counting books in <i>scenarioMain.py</i>	31
4.16	Displaying text on Pepper's tablet	31

Chapter 1

Introduction

Technology is becoming more and more present in our modern society. Digital devices are used everywhere, from big industrial robots in factories to smartphones and laptops in our daily lives. The field of robotics is an integral part of this progression of technology in our lives, as more types of robots are developed.

An example of this is social robots [1]. Social robots are robots made specifically with human interaction in mind. They have humanoid characteristics and are able to interact with their environment and mimic social interactions. They can provide service and companionship in many different situations, for example in education as assistants to teachers. They can also be companions for the elderly, giving them a social connection while also having the same functionalities as digital devices which enable them to give reminders, contact family or caregivers and so on. Social robots can also be receptionists, helping to automate some tedious tasks.

Another area influenced by the expansion of the the robotics field is libraries. More and more studies are exploring ways to integrate robots in libraries. Many tasks performed by librarians can be automated and given to robots. These library robot are able to navigate the library they are in and identify books, usually through specific markers put on the books, like radio frequency tags. They can then manipulate these books, retrieving them for patrons, sorting them to be in the correct order or simply registering their position. However, contrary to social robots, these robots helping in libraries are usually not interacting much with library patrons. This study focuses on filling this gap by having a social robot perform the tasks given to robots in library.

The integration of a social robot as assistant in the library would be beneficial by bringing in the advantages of the social robots, automation and interaction, in the environment and tasks of the library. However, a social robot faces issues that other robots might not, namely it is not made to instantly recognize books. While other robots may have specific sensors, the social robot used in this study, Pepper, lacks them, relying instead on microphones and cameras to perceive its environment. Thus, this research focuses on developing a program that enables Pepper to identify books in front of it.

This study aims to apply machine learning solutions for computer vision problems to the specific situation of this social robot identifying books in the library.

1.1 Keywords

Social robotics; Computer vision; Object detection; OCR

1.2 Problem statement

The main goal of this master thesis is to evaluate the possibility to use a social robot, Pepper, as an assistant in the library. The main scenario would be to have Pepper talk to people and when they are looking for a specific book, they would give its title to Pepper and it would look for it in the library.

Pepper is well capable of social interaction with humans, being equipped with modules to talk, recognize faces and even emotions. However, it is lacking the sensors that enable most robots currently used in libraries to recognize books. The robots developed for libraries are made for this environment and specifically to detect, identify and sometimes even move books. As will be seen later in the Background chapter, most of them use RFID technology to find and identify books, and have gripper arms mounted on a mobile platform that allow them to take the books and move them in the library. The sensor that is most suited to the task of detecting the objects around the robot is the camera. However, Pepper doesn't have specific modules that would allow it to detect specific objects, or to read text on an image.

This immediately poses the problem which is at the center of this study: How can Pepper detect and identify books in its proximity? Or how can it be made to detect and identify books?

The main emphasis of this study will be put on developing the robot's perception of books in order to identify them.

In order to analyse the environment of the robot, some computer vision techniques will be required. Computer vision is a field of Artificial Intelligence that focuses on the problem of deriving meaningful information from images. In this case, two specific subtasks will be needed: object detection, which centers on detecting objects in images or video, and Optical Character Recognition (OCR) which focuses on extracting text from images.

This study will focus on these two subtasks applied to Pepper's camera for the goal of identifying books.

1.3 Research Questions

The combination of object detection and OCR has been explored in previous studies, but this study focuses more specifically on the social robot Pepper, and on book titles as a target. This goal is to explore the use of object detection and OCR

on Pepper's camera to read the title of books in front of Pepper. This objective is relayed by the following research questions:

- *How can we integrate machine learning with Pepper to enable it to read book titles using its camera?*
- *Which machine learning algorithms and models related to object detection and OCR can be used and are most efficient for this goal considering different parameters?*
- *To what extent can this integration of machine learning with Pepper be successfully used in the library?*

1.4 Contributions

This research study is an important first step in using Pepper in the library. Having a way to read the titles of books and thus to identify them opens many opportunities for uses of Pepper in the library.

The introduction of social robots in libraries serves several interests for the library itself. The automation of repetitive and time-consuming tasks is an important goal to improve the efficiency of library management. This eases the work of librarians while improving the quality of service to patrons. Using a social robot for this task is also crucial for said quality of service, because social robots are designed for human interaction.

More generally, this study contributes to the field of social robotics by trying to integrate a social robot in a new context. It focuses on the robot's perception of its environment and tries to expand it to books. The use of Computer Vision combined with Pepper's camera enables it to interact with its environment in other ways, which can be useful in many different contexts outside the library as well. This allows the robot to recognize and learn from its environment.

1.5 Outline of Chapters

In order to give some context to this research, as well as some introduction to the fields of social robotics and computer vision, chapter 2 presents some related works which give some background to this study. Chapter 3 presents the robot used for this study, Pepper, and introduces the scenario that will be implemented, along with the methodology followed by the experiments, with the procedure and choices made. Chapter 4 explains the implementation of both the experiment and the scenario in python, with details on technical solutions employed. Chapter 5 gives an analysis of the results of the experiment in terms of both object detection and OCR, while chapter 6 reflects on those results and discusses their applications to the scenario, their implications and their limitations. Finally, chapter 7 concludes on the findings of this study and examines some ideas for potential future work related to this topic.

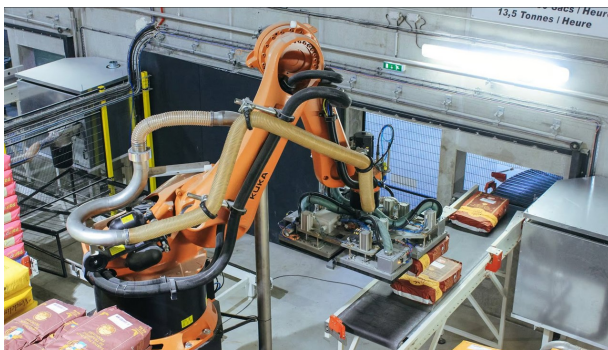
Chapter 2

Related Works

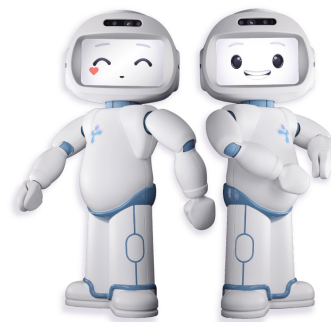
This chapter will present some related work in the field of Robotics and Machine Learning.

2.1 Robotics and social robotics

The field of robotics is constantly evolving and has seen in the last decades strong technological advancements. Different kinds of robots have emerged, suited to different fields and different tasks. Industrial robots have been used in manufacturing for decades, increasing precision and speed, and replacing humans to perform tasks which would be highly repetitive or dangerous. Robots are now also coming into other areas in our lives, as humanoid and social robots become more common.



(a) An industrial robot, palletizing sacks of flour in a Moulins Bourgeois factory in Verdelot, France
Source: Kuka, Kuka.com



(b) QRobot, a humanoid robot tasked to teach kids with autism
Source: LuxAI, luxai.com

Figure 2.1: An example of industrial robots and humanoid robots

Social robotics is a domain of robotics that has become bigger and more attractive recently. There are different ways to define social robots but they tend

to be defined as autonomous robots which can have complex social interactions with their environments, including other robots and humans [2]. They incorporate human characteristics and abilities in order to mimic social interactions. These social interactions can be expressed through face recognition, physical touch, dialog, emotion recognition... Social robots are deployed for different uses, being used as companions and educational tools [1]. They are used in elderly care and in hospitals, to interact with patients and assist them [3]. They are also present in education, where they can help at different levels and in different fields [4, 5]. Aldebaran introduced their Pepper robot, a social robot, and explored in a blog post the ways that it could help as a receptionist, being able to greet employees and visitors, and automating some tasks such as check-in for visitors, or providing information about arranged meeting for visitors [6].

2.2 Robots in libraries

Following the expansion of robots in more and more fields for different situations and tasks, they have also started being used in libraries. Many tasks that librarians perform can be automated, and the use of robots can make life easier for librarians. Different types of robots have been employed and a number of technologies have been developed and used in order to facilitate the navigation of the libraries, and the detection of books by the robots. In most cases, robots are used to help people find books, but they are sometimes also tasked with checking that every book is on the shelf where it should be and with sorting the books that are not in the right place.

A popular solution to the problem of robots navigating autonomously in the libraries, has been to give said robots internal maps which they use to find their way. The robot presented in [7] uses a predefined map and Dijkstra's algorithm in order to find the shortest path to its goal. Similarly the robot from [8] has an internal map of the library and uses it to find the shortest path, but it also dynamically updates the map as it moves in order to avoid potential obstacles. The robot in [9] is used in a library where shelves have RFID tags and books have barcodes. It has an internal map made of a tree where the nodes are the RFID tags of the shelves, and the books are mapped to their corresponding shelves. When it needs to go somewhere in the library, the robot plans its route using the tree and it scans the RFID tags as it navigates the library in order to find its path, and then scans the barcodes to identify the books.

When it comes to the book detection, RFID (Radio Frequency Identification) tags are one of the most used methods [7, 10, 11]. The books have RFID tags which, when triggered by an RFID reader, transmit digital data. The robots are equipped with the RFID reader and receive the digital data, which they match against their book database. This allows them to detect and identify books. Similarly, some robots scan barcodes or QR codes added on the spine of the books to identify books [8, 9, 12]. This robot [13] uses a model of invisible bookmarks made with special ink to identify books.

Another study [14] uses a self-made dataset to train a segmentation model which detects each book in an image. It matches each detected instance against a database of images of the book spines in order to recognize precisely which book it is. The robot presented in [15] uses a database with for each book a picture of its spine. When the robot is looking for a specific book, it compares the pictures it takes of the shelf full of books with its database and thus finds the book it is looking for. While this method seems efficient, the time necessary to create the database by taking and processing pictures of all the books in the library makes it very time-consuming.

This study [16] would put a note taped on the spine of the books, with an identification code for each book. The robot would read the note (using OCR or other Machine Learning algorithms), and look for the identification code in its database to identify the book. Another study [17] uses a similar approach, where the robot extracts the book labels using image segmentation and OCR.

Something to note is that most robots used in libraries are not humanoid or social robots, they are much closer to industrial robots. The robots as mentioned previously tend to not have much interaction with either librarians or patrons, and thus they don't need to be humanoid robots, which would be more suited for contact with people. Most of them are simply comprised of a mobile part, to navigate the library, and sensors, to be aware of their surroundings and detect books, with some of them also having a gripper to take the books.

While it is true that the robots used in library are not, for the most part, made to have interactions with people like social robots, they are designed for the libraries and for detecting, recognizing and interacting with books. However, social robots like Pepper are missing the RFID sensors that many library robots have and which enable them to recognize books. Therefore, it will be necessary to implement another way for them to identify books.

2.3 Object Detection and Optical Character Recognition

This subsection will introduce some Machine Learning algorithms and their implementation, which will be useful in the context of a social robot recognizing its environment: Object Detection and Optical Character Recognition. Object Detection will allow the Pepper robot to identify the objects it sees with its camera, more specifically to identify books, and Optical Character Recognition will be used to extract the text on the spine of the previously identified books, corresponding to the titles of said books.

2.3.1 Object detection in real time

Object detection is a computer vision problem, where the goal is to locate instances of objects in images or videos, as seen in figure 2.2. It is a popular topic in the field of Machine Learning and Deep Learning, and there are many existing algorithms and models that tackle this problem. The additional constraint of real-time object

detection makes the problem more specific: the algorithms need to process images fast enough while still maintaining high accuracy. Real-time object detection is often used on real-time video sequences, and thus requires an extremely short inference time.

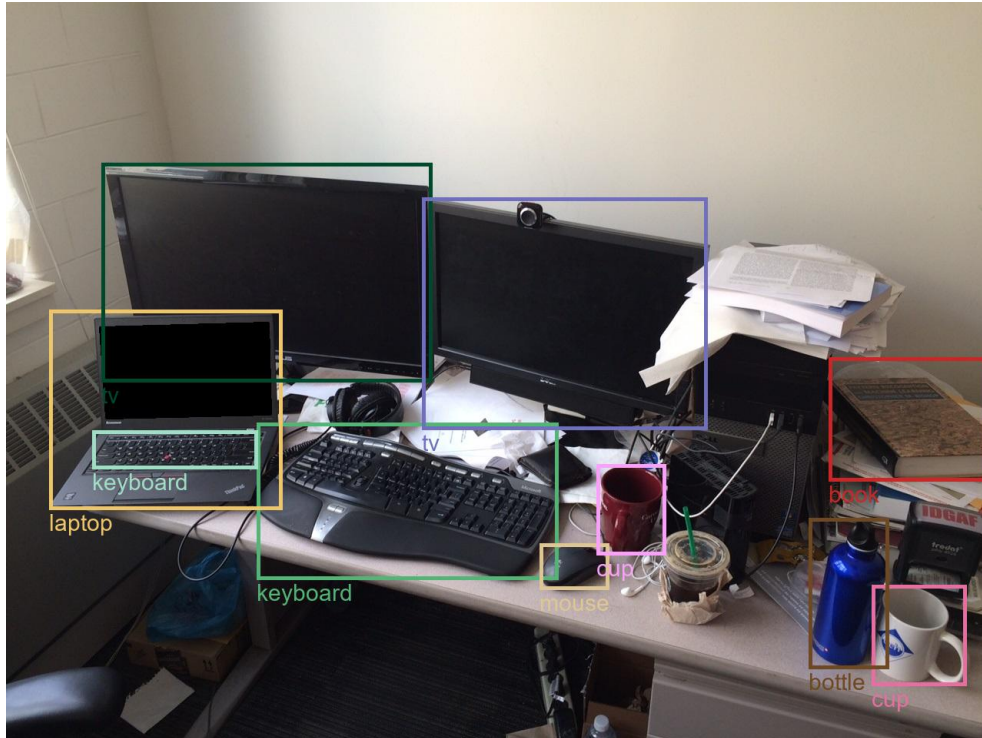


Figure 2.2: Results of object detection on a photo of a desk

There are two big categories of object detection algorithms: one-stage algorithms without proposals, and two-stage algorithms with a proposal phase. The first category, one-stage algorithms, processes the image only once to make predictions on the location and label of objects in the image. The second category, with two stages, first uses a network to predict the location of objects on the images (the proposals), and then in the second stage refines these proposals and assign labels to them (the names of the object detected). Both types of algorithms have their advantages and disadvantages, two-stage algorithms tend to be more accurate while one-stage algorithms tend to be faster and able to detect in real time, but that is not always strictly the case [18].

Faster R-CNN [12] is an example of two-stage algorithm. As such, it first processes the image through a RPN (Region Proposal Network) which finds proposals (locations where objects might be), and then the image is passed through a detection network which classifies the objects detected. Proposed in 2015, it offered a new RPN which shared features with the detector, allowing the calculations to be faster while maintaining state-of-the-art accuracy when it came out. There have since been improvements on it. For example, in 2017, **Mask R-CNN** [19] which

extends Faster R-CNN to make it easier to generalize to other tasks.

The **Yolo** detector [20] is currently the state-of-the-art when it comes to real-time object detection. It is a one-stage detector, which uses a single neural network to process an input image and output the detected objects with their label, bounding box and confidence level. There have been several versions of the Yolo detector since its first iteration, each improving on the previous version. The last version to date is Yolov7 [21].

2.3.2 OCR or Optical Character Recognition

OCR stands for Optical Character Recognition. It is an algorithm which is able to 'read' text from an image [22, 23]. There are two main use cases for OCR: the digitization of paper documents, in which a paper document is scanned as an image file or a pdf and OCR is used to get the text from the document, and the analysis of a scene image which contains text in the environment it depicts. The second case tends to be much more difficult than the first, as complex backgrounds, low resolution and variations in fonts and layouts of text in real-life scenes make the text harder to read than in clean scanned documents [24].

There are many implementations of OCR algorithms, here are some OCR libraries in python:

Pytesseract [25] is a python wrapper for Google's Tesseract-OCR Engine. Tesseract was first developed at HP in the 1980s and made open source in 2005, after which Google further developed it. Having been developed for a few decades, it has a different structure than more recent OCR software. It first finds blocks of text, which are then broken down into lines, words and characters. Recognition then proceeds, and each recognized word is fed to an adaptive classifier, which makes it easier to recognize words further down the page. Thus Tesseract gets better at recognizing words as it reads the page, but because the adaptive classifier might have learned something 'too late' (towards the bottom of the page), Tesseract goes through the page once again to try to recognize more words [26]. Tesseract is today a very popular OCR, and performs especially well on scanned documents, though it tends to have a harder time for scene images [27].

EasyOCR [28] is a python OCR module that recognizes more than 80 languages. EasyOCR works in two steps: first it uses the CRAFT (Character-Region Awareness For Text detection) algorithm to detect the text in the image, and then recognition is done with a CRNN (Convolutional Recurrent Neural Network) made of 3 main components: feature extraction, sequence labeling and decoding. It is quite easy to install and use compared to other OCR libraries and it is implemented using the PyTorch library, meaning that having a CUDA-enabled GPU (so that it can run some of the workload usually only on the CPU) can make the computations much faster.

keras-OCR [29] is a python library which provides OCR models and an end-to-end training pipeline capable of training new OCR models. It is based on the CRAFT method for text detection and a Keras implementation of CRNN (Convo-

lutional Recurrent Neural Network) for text recognition.

PaddleOCR [30] is an open-source tool built by Baidu Research for OCR. It supports many languages and non-latin scripts. It uses a series of OCR models called PP-OCR. They are ultra-lightweight algorithms using two stages: first text a detection model followed by a CRNN for text recognition. They are very fast models, and tend to perform well on documents, but not so much in scene images [31].

Calamari OCR [32, 33] is a python OCR tool that was developed to be used on historical documents. It uses techniques such as voting and pretraining to avoid the effort of manually annotating all the training data with the actual results. Calamari performs well, with a low character error rate and fast computation on scanned documents [33].

2.3.3 OCR accuracy

The question of how to measure OCR accuracy has been explored in a 2021 survey [34] which presents different metrics used. The current state-of-the-art comes from the work of Stephen V. Rice [35] who presented algorithms that measure the performance of OCR algorithms. He defines two main measures, character accuracy and word accuracy.

Character accuracy uses the Levenshtein distance [36]. If you have a string A of length n which is the correct string, and a string B which is the string generated by the OCR, then E is the minimum number of edits (insertions, deletions and substitutions of characters) necessary to go from B to A . Then the character accuracy is $\frac{n-E}{n}$.

Word accuracy is similarly defined. From a correct list A of n words and an OCR-generated list B , E is this time the minimum number of insertions and substitutions of words needed to go from B to A . Rice does not count deletions of words in this case. Then the word accuracy is defined as $\frac{n-E}{n}$.

The character accuracy can actually be negative, if the two strings have no characters in common and the OCR string has more characters than the correct string. In place of character and word accuracy, the metrics more commonly used are CER (Character Error Rate) and WER (Word Error Rate), defined as $\frac{E}{n}$ [34]. The lower the values of the CER and WER, the higher the accuracy, with an error rate of 0 signifying that the correct and OCR strings are identical. And in the previous case of two strings having no characters in common with the OCR string being longer than the correct string, instead of having a negative accuracy, the error rate is simply higher than 1, or than 100%.

Chapter 3

Methodology

This chapter will present the robot used for the study, the design of the application, and the experiments performed.

3.1 The Pepper Robot

Pepper is a social robot that was developed by SoftBank Robotics [37], which is shown in figure 3.1. Pepper is a 1.2-m-tall wheeled humanoid robot. It has sensors that allow it to perceive its surroundings, including two RGB cameras on the forehead and in the mouth with a resolution of 640×480 pixels at 30 frames per second, and a 3-D sensor behind the eyes with a resolution of 320×240 pixels at 20 frames per second. It has a total of 17 joints which allow it to move in different ways and mimic a humanoid behaviour. It also has a tablet on its chest that can be used to display images, videos or even websites.

Pepper is able to move and express itself through speaking and body language. It can interact with people using voice and emotion recognition, and many more aspects were designed to make it seem more sociable: its head following the person to look at them while talking to them, expressive body language through moving its arms (hands and fingers included) and its body in a natural way... Pepper can be programmed to bow to greet customers, to dance when asked to or to giggle when someone touches its head for example.

Pepper's navigation is performed with the use of three wheels (left, right and back) at its bottom. They allow Pepper to move around its environment. Pepper also has two degrees of freedom in its neck which lets it turn its head around, and several joints in the arm and hands (shoulders, elbows, wrists and hands). Pepper is programmed for basic navigation and to avoid obstacles when moving around.

Naoqi

Naoqi is the name of the operating system that runs in the Pepper robot and controls it. The Naoqi framework can be used to program Pepper, as well as other robots created by SoftBank robotics like Nao.

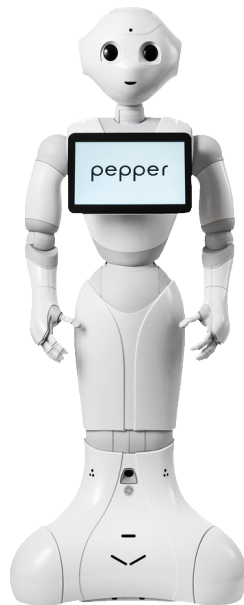


Figure 3.1: The Pepper robot

Naoqi had many modules that are used for different aspects of Pepper. ALDialog gives the ability to make Pepper talk to a human following a set of rules, and ALPhotoCapture allows the programmer to take pictures with the robot's cameras and save them. All these modules are available in Python and make it possible to create complex programs for Pepper.

3.2 Design of the scenario

The goal of this study is to determine explore the use of a social robot, in this case Pepper, in the library. The robot would be able to navigate the library to detect and identify books.

In order to implement this, Pepper would take pictures of its environment. On each of these pictures, an object detection algorithm would be run to detect books. For each book detected on the picture, a copy of the picture would be created and cropped to show only the book detected. And finally on this cropped picture, an OCR algorithm would be run in order to read the title of the book. The process is described in figure 3.2.

As an example, figure 3.3 shows a photo of books taken by Pepper. Then the object detection model is used to detect books, which gives the result on the right of figure 3.3. Then each book would be identified them by reading the title on

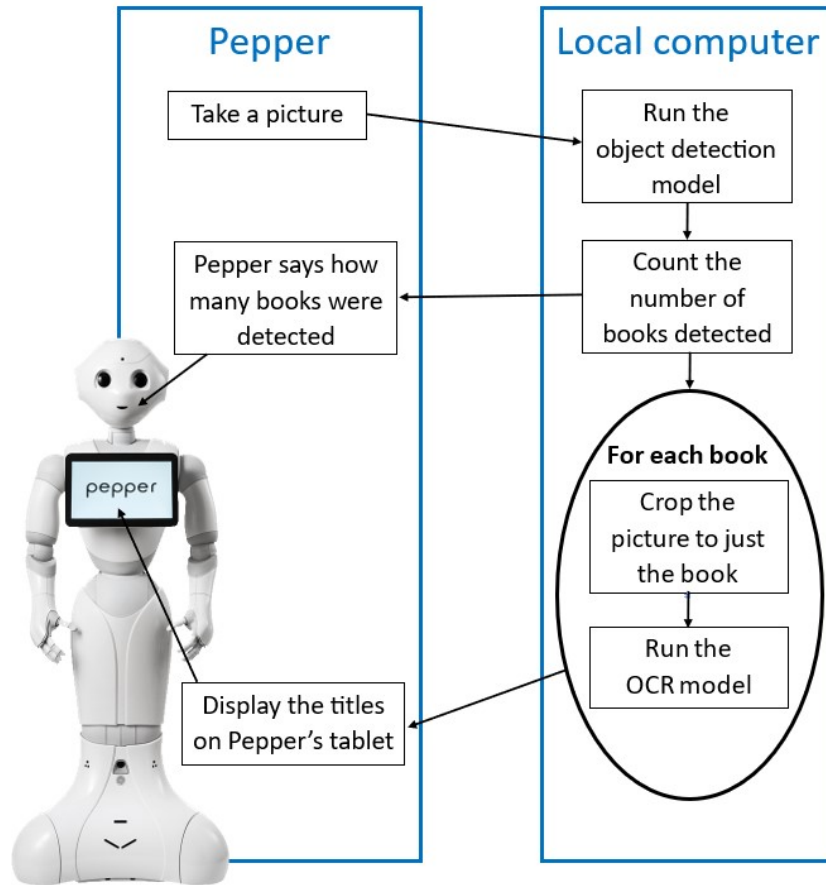


Figure 3.2: The process to identify a book in the application with the Pepper robot

their spine. An example can be made using the book at the top of the pile in figure 3.3. Figure 3.4 shows the photo cropped to just the book on top of the file, based on the coordinates of the book as detected by the object detection model. Once the photo is cropped, an OCR model can read the title from the cropped picture: "Handbook of Algorithms for Wireless Networking and Mobile Computing", which is how the robot can identify the book by its title.

One of the key elements of this process is the OCR model. Thus, the experimental phase of this study will compare some OCR models that can be used in this scenario, in order to find the most accurate one.

3.3 Experiments

This section will present the experiments, the tools chosen and preliminary decisions made for this study.



(a) A photo of books taken by Pepper

(b) The same photo after using the object detection model

Figure 3.3: Example of the process explained previously: Taking a photo with Pepper and running the object detector model



Figure 3.4: The previous photo from figure 3.3, cropped to only show the book at the top of the pile

3.3.1 Procedure

As a first approach to this problem, the scenario explored was simplified.

- The books were put in a pile in front of Pepper. They were put on a box so that they would be at the right height for Pepper to see them without having to turn its head.
- They were positioned on their side so that their titles on the spine would be the right way up to avoid having to figure out which way the text was written and having to rotate the image based on that.
- The books were chosen to be thick and have their title written in big text.
- All books are also in English to avoid confusion with letters from other alphabets.

The procedure of the experiments followed the process detailed in figure 3.2. Pictures were taken of the books in front of Pepper. They were then analysed with the object detection model to detect books, and for each book, a cropped image

was created and run through an OCR model to extract the text of the book title. All the results were put in an excel sheet and compared to the actual title of the book to determine the parameters which made the detection and text extraction most accurate.

3.3.2 Parameters

In the experiments, different parameters were compared to determine the most accurate way to read the title of the books.

Books

All the books used in the experiments were borrowed from the library of NTNU in Gjøvik. They were chosen because they fit the conditions mentioned in section 3.3.1. All books are thicker than 2cm and have text whose height is between 0.7 and 1.1cm. There are 6 books in total with different colors and sizes.

The books are shown in figure 3.5. Their number is an identifier which made the recording of results easier.



Figure 3.5: The books used for the experiments

Distance

A parameter that can influence the identification of the books is the distance between the robot and the books. If the books are too close to the robot, they might not fit in the photo taken and the camera might not be able to focus on the books, or on the other hand if they are too far away, they might not be detected by the object detection model, or the text might not be readable for the OCR model.

The distance was measured as the horizontal distance between the books and the tablet on the chest of Pepper. Preliminary tests have shown that a distance between 45cm and 70cm was best. At less than 45cm, the books did not fit in the photo, and the detection rates got worse at more than 70cm.

Thus the experiments took photos of the books when they were at a distance of 45cm, 50cm, 60cm and 70cm of the robot.

Resolution

The camera at the top of Pepper's head can take photos in different resolutions, numbered with an ID value in the documentation of the robot ([link here](#)). While the resolution 4 (Image of 2560x1920px) seems like the best one in the documentation, it turned to not be usable at all, as the photos take with this resolution were completely black (as seen in figure 3.6).



Figure 3.6: A photo taken by the Pepper robot with its top camera and resolution of 4

The second best resolution according to the documentation is the resolution 3 (Image of 1280x960px). Further experimentation with Pepper revealed a resolution of 6 (Image of 1920x1080px) not present in the current 2.8 documentation, but in a previous 2.0 version ([link](#)).

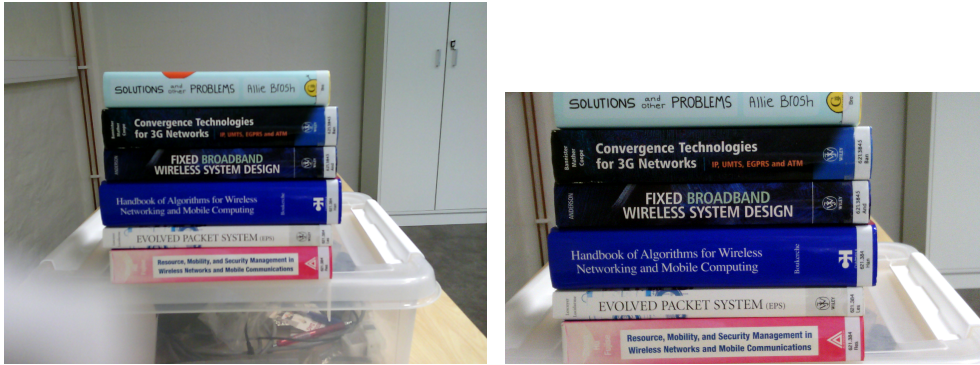
The resolution 6 is higher than resolution 3, but it also narrows the image, making it harder to fit all the books in, as can be seen in figure 3.7. Thus, in the experiments, photos were taken of resolution 3 and 6 in order to compare the accuracy with both.

Iteration

For each distance and resolution, 6 pictures were taken with all of the books in each picture.

In a few cases, the object detection model did not manage to detect all of the books, or detected two books as one. Having more than one picture made it possible to have results for the OCR part even if the detection failed sometimes. This can be equated to Pepper taking another picture in the original scenario if something went wrong.

The books were moved to a different order for each iteration and the results were then averaged to see the global accuracy.



(a) A photo of books taken with Pepper's camera at resolution 3 (1280x960px) (b) A photo of books taken with Pepper's camera at resolution 6 (1920x1080px)

Figure 3.7: Photos of the same pile of book, taken with resolution 3 (left) and resolution 6 (right)

Object detection library

In the case of our application, it is necessary to have a fast object detection algorithm. As such, since it is required to have real time detection, a 1-stage algorithm was chosen with YOLO.

The object detection was performed with cvlib [38], a simple open source computer vision library for python. It has a function to detect common objects in scenes, including books. The function uses the YOLOv4 model [39], proposed in 2020 as a faster and more accurate improvement on the YOLO detector.

OCR libraries

An important decision was the choice of which python OCR libraries to compare in the experiments. The amount of time available for this work limited the number of OCR libraries that could be implemented and used in the experiments.

Pytesseract was chosen for several reasons. Tesseract is a popular and commonly used OCR, and due to having first been developed in the 1990s, it functions differently from current OCR models.

EasyOCR and keras-ocr were chosen to compare two similar models, both using the same CRAFT model for text detection and a CRNN for text recognition.

PaddleOCR and Calamari OCR were not implemented. They were both reported to perform well on scanned documents but poorly on scene images, which is what is required here.

More libraries could and should be tested in further works on this topic.

3.3.3 Data collection

The object detection and OCR models were run on the pictures and the results were put in an excel document.

The excel document has four sheets for the results: the first one is a table with the description of the books used for the experiments, then both object detection and OCR results have one sheet for the data and one sheet for data analysis with tables and graphs.

Object detection results

When taking the photos and for the object detection, there were two variables to take into account: the resolution (two values, 3 or 6) and the distance between the robot and the books (four values, 45cm, 50cm, 60cm or 70cm). This is a total of 8 configurations. There are 6 photos taken of each configuration (for 6 iterations), which makes a total of 48 pictures.

All 6 books are on each picture, and the results must detail whether each of the 6 books was detected or not. Therefore, in the detection sheet, there are six rows for each picture, one for each book. On each row are the image file name, the corresponding variables: resolution, distance, iteration and book ID, and one column for the results.

The results are related to the output of the object detection model, each output having been saved to a json file. The object detection model outputs an array of bounding boxes, labels and confidence levels, which are then used to determine where the books are, and to crop the pictures.

In order to analyse the output of the object detection, the question that was asked was:

For each of the 48 pictures, and for each of the 6 books in each picture, was the book properly detected?

In this case, "properly detected" doesn't mean that the bounding box perfectly fits the book, this isn't the case most of the time because the bounding boxes are not rectangles and the books are not perfect rectangles on the image. "Properly detected" means that book is in the bounding box and the only writing in the bounding box is on that book.

In order to analyse this information, the results were put into one of four categories in the "results" column of the object detection sheet: Detected, Detected with another book, Detected partially with another book and Not detected. The four categories are defined with examples in table 3.1.

OCR results

After the object detection, for each book detected by the object detection model, a cropped image was created with the coordinates of the bounding box.

For the OCR results, the cropped images which were categorised as "Detected partially with another book" or "Detected with another book" were excluded.

In the OCR results sheet, each of these cropped images gets three rows, one for each OCR model. The rows still have the image file name and the variables: resolution, distance, iteration, book ID and OCR model, but instead of the detection result column, there are three columns: one for the result predicted by the OCR,


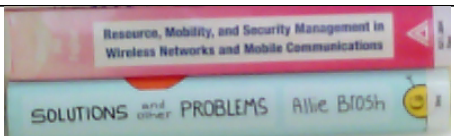
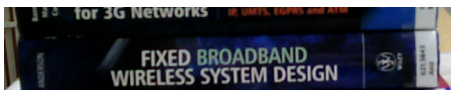
Category	Definition	Example
Detected	The book is in the bounding box and the only writing in the bounding box is on that book.	
Detected with another book	Two whole books are in the same bounding box, and none of the other bounding boxes contain these two books	
Detected partially with another book	The bounding box also contains a little bit of another book, enough that some but not all of the text is visible.	
Not detected	The book is not in any of the bounding boxes labeled as a book by the model.	

Table 3.1: The four categories of detection results

one for the actual result that is expected, and the last column for the accuracy, obtained by comparing the predicted and actual results.

In this case, the expected result corresponds to the text that is written on the spine of the book and that is horizontal when the book is on its side. Table 3.2 shows the detail of this for each book.

As detailed earlier in the Background chapter, the two main metrics for OCR are the CER (Character Error Rate) and the WER (Word Error Rate). In the case of this study, the CER is a relevant metric, but the WER not so much. WER was established with in mind the application of OCR to longer documents and for information retrieval purposes. This does not fit this study, as the strings used are quite short and with very few words compared to the entire documents that WER was designed for. Thus for this study, the "accuracy" measure that will be used will be the CER.

The fastwer library in python implements the WER and CER, and was used to evaluate the accuracy of the OCR models used. The library gives the results as percentages: if the OCR results needs 1 edit to be equal to the correct string of length 20, then the fastwer library will give a result of 5, instead of 0.05.

When analysing the results, two characteristics will be evaluated: Accuracy and Consistency. For given parameters, accuracy will be evaluated through the average of the CER values, and consistency will be evaluated through the standard deviation of the CER values.





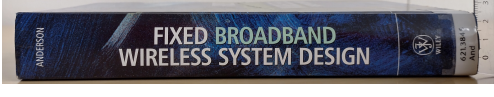

Book ID	Photo	Expected result for the book
0		Solutions and other problems Allie Brosh
1		Resource, Mobility, and Security Management in Wireless Networks and Mobile Communications
2		EVOLVED PACKET SYSTEM (EPS)
3		Handbook of Algorithms for Wireless Networking and Mobile Computing
4		FIXED BROADBAND WIRELESS SYSTEM DESIGN
5		Convergence Technologies for 3G Networks IP, UMTS, EGPRS and ATM

Table 3.2: The books and the associated expected result for each

Chapter 4

Implementation

This chapter will detail the structure of the code files used in this study, starting with the code for the experiment and finishing with the code for the scenario.

The code files are also available at *this repository*.

4.1 Experiments

The experiment was designed to determine which parameters gave the best results.

The code for the experiments was separated into four files:

- TakePictures, which takes pictures with the Pepper robot and transfers them to the computer;
- CropPictures, which uses the object detection model on all the pictures to find books, and for each book found, creates a cropped picture with just that book
- UseOCR, which for each cropped picture, runs the OCR models on it and saves the strings to txt file
- and Accuracy, which takes from the excel files with the results takes the OCR result and the ground truth to calculate the CER.

The python files have to be run one after the other.

4.1.1 TakePictures

The code of *TakePictures* is separated into a main function and a `if __name__ == '__main__':` statement.

The if statement contains an argument parser which allows us to get the robot IP address and Naoqi port number in order to establish the connection to the Pepper robot.

Code listing 4.1: Connection to Pepper

```
if __name__ == '__main__':  
    parser = argparse.ArgumentParser()
```

```

parser.add_argument("--ip", type=str, default="192.168.137.233",
                    help="Robot_IP_address.\nOn_robot_or_Local_Naoqi:_use_\n"
                    "127.0.0.1'.")
parser.add_argument("--port", type=int, default=9559,
                    help="Naoqi_port_number")

args = parser.parse_args()

# Initialize qi framework.
connection_url = "tcp://" + args.ip + ":" + str(args.port)
app = qi.Application(["Detector",
                    "--qi-url=" + connection_url])

# Connection
try:
    app.start()
    main(app)
except RuntimeError:
    print ("Can't connect to Naoqi at ip\n" + args.ip + "\n on port\n" +
          str(args.port) + ".\n")

sys.exit(1)

```

After that, the *main* function is called with the application as an argument. The connection to the ALPhotoCapture module is established, in order to be able to take the pictures from Pepper. After that, the parameters for the *photoCapture.takePicture* function must be set using the *photoCapture.setCameraID* function to choose the camera to be used (here the camera n°0, at the top of Pepper's head), the *setResolution* function to choose the resolution and the *setPictureFormat* function to set the picture format to jpg.

As mentioned, there are 48 pictures to take, with 4 different distances, 2 resolutions and 6 iterations. The different distances and iterations require manual manipulation of the books, so the code can only take 2 pictures at once, the pictures of resolutions 3 and 6 with the same values of distance and iteration. Thus the code had to be run 24 times to change the distance and iteration parameters in between.

The *takePicture* function needs the folder path where to save the photo and the name of the photo file. In order to make things clearer, each picture has in its file name the parameters of distance, iteration and resolution.

Code listing 4.2: Taking a picture with Pepper

```

PepperPath = "/home/nao/recordings/cameras/library/"
pictureFormat = "png"
distance = 45
iteration = 5
resolutions = [3,6]

photoCapture.setCameraID(0)
photoCapture.setPictureFormat(pictureFormat)

for resolution in resolutions:
    photoCapture.setResolution(resolution)
    pictureName = "image_" + str(resolution) + "_" + str(distance) + "_" + str(
        iteration)

```



```
photoCapture.takePicture(PepperPath, pictureName)
```

Once the pictures have been taken, they need to be transferred to the computer files where they will be further analysed with the object detection and OCR models. In order to do this, a paramiko connection is used.

The connection is established with the use of the same ip address and port as previously, and from this, the code can transfer the files from the robot to the computer using the `sftp.get` function.

Code listing 4.3: Transferring the files to the computer

```
ComputerPath = "C:\\PepperRecordings\\Part_1\\"
t = paramiko.Transport(args.ip, args.port)
t.connect(username="nao", password="edutech123")
sftp = paramiko.SFTPClient.from_transport(t)
files = sftp.listdir(PepperPath)
print("Files_on_Pepper_in_folder" + PepperPath + ":")
print(files)

for resolution in resolutions:
    pictureName = "image_" + str(resolution) + "_" + str(distance) + "_" + str(
        iteration)
    f = pictureName + "." + pictureFormat
    print('Transferring_' + f + '...')
    sftp.get(os.path.join(PepperPath, f),os.path.join(ComputerPath, f))
    print("File_transferred.")
    sftp.remove(PepperPath+f)          # This deletes the file from Pepper once
    it has been transferred to the computer.
```

4.1.2 CropPictures

The *CropPicture* file takes the folder in which the pictures taken have been saved and for each picture in that folder, uses the object detection model to find the books and create a cropped picture of each of the detected book.

In order to do that, a subfolder *Cropped* is created to save the cropped images. After that, the code goes through the images and calls the function *cropPicture* on each.

Code listing 4.4: The main part of the *CropPictures* file

```
#####
# Creating folder for cropped pictures
ImagesPath = "C:\\PepperRecordings\\Part_2\\"
croppedFiles = os.path.join(ImagesPath, "Cropped")
os.mkdir(croppedFiles)

#####
# List all files and use detector to crop books
for entry in os.listdir(ImagesPath):
    fullPath = os.path.join(ImagesPath, entry)
    if os.path.isfile(fullPath):
        print(entry)
        cropPicture(ImagesPath, entry)
print("Detection_finished,_pictures_cropped")
```

The *cropPicture* function first calls the *detect* function, which uses the *opencv* library to detect the objects in the image and returns the bounding box, label and level of confidence for each object found in the image.

Code listing 4.5: The *detect* function

```
def detect(fileName):
    im = cv2.imread(fileName)
    bbox, label, conf = cv.detect_common_objects(im)

    return(bbox, label, conf)
```

After that, the *cropPicture* function saves the data returned by the *detect* function in a json file. This was mainly done for debugging purposes, so that that data could be accessed after having run the code.

Then, the code goes through the data in order to find the objects detected as books and get their bounding boxes. As it gets those bounding boxes, the *Image* module of the *PIL* (Python Imaging Library) is used to create a cropped picture and save in the *Cropped* subfolder.

Code listing 4.6: The *cropPicture* function

```
def cropPicture(ImagesPath, entry):
    # Collect data from the detector
    ImagePath = os.path.join(ImagesPath, entry)
    data = detect(ImagePath)

    # Save data to a json file
    json_obj = json.dumps(data)
    jsonFile = open(ImagePath[:-4] + '.json', 'w')
    json.dump(json_obj, jsonFile)
    jsonFile.close()
    print("#####Detection_ finished")

    # Crop picture for each book found
    numberOfBooks = 0
    for i in range(len(data[1])):
        if (data[1][i] == "book"):
            numberOfBooks = numberOfBooks + 1
            # Getting the coordinates
            x0 = float(data[0][i][0])
            y0 = float(data[0][i][1])
            x1 = float(data[0][i][2])
            y1 = float(data[0][i][3])
            # print("          - Bounding box: (" + str(x0) + ", " + str(y0) + " ) ,
              (" + str(x1) + ", " + str(y1) + " )")

            # Cropping image
            im = Image.open(ImagePath)
            im = im.crop((x0, y0, x1, y1))
            cropName = os.path.join(ImagesPath + "Cropped\\" + entry[:-4] + "Crop"
                + str(i) + ".png")
            # print("          Name of the cropped file: " + cropName)
            im = im.save(cropName)
    print("##### " + str(numberOfBooks) + " _books_ detected")
    print("#####Cropping_ finished")
```

4.1.3 UseOCR

The *UseOCR* code defines 6 functions to use the OCR libraries. After that, the main part of the code is a loop that goes through all the cropped images and calls the *pytess*, *easyOCR* and *kerasOCR* functions defined higher.

Code listing 4.7: The main loop of UseOCR.py

```
#####
# For each cropped file, use all 3 OCR algorithms and store data in txt files.
# List all files and use OCR on all
for entry in os.listdir(croppedFiles):
    fullPath = os.path.join(croppedFiles, entry)
    if os.path.isfile(fullPath):
        print(fullPath)
        pytess(fullPath)
        print("#####Pytesseract_OCR_done")
        easyOCR(fullPath)
        print("#####EasyOCR_done")
        kerasOCR(fullPath)
```

pytess is the first function called. It uses the pytesseract library and applies it to the image. The pytesseract library and the Tesseract OCR Engine had to be installed and the UseOCR.py file started with the imports among which *import pytesseract*, and the line *pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files (x86)\Tesseract-OCR\tesseract'* which enables the use of the installed Tesseract OCR Engine.

The *pytess* function calls the *image_to_string* function of the pytesseract library to extract the text from the image. The text extracted is then saved to a text file in the same folder as the image.

Code listing 4.8: The *pytess* function

```
def pytess(fileName):
    #Use OCR algorithm
    data = pytesseract.image_to_string(fileName, lang='eng')
    # print("Text recognized: ")
    # print(data)

    #Save data to a txt file
    print('#####Name_of_txt_file:_' + fileName[:-4] + 'Pytess.txt')
    with open(fileName+'.txt', 'w') as f:
        f.write(data)
```

The **easyOCR** function works similarly to the *pytess* function. The library has been imported with *import easyocr* and a reader has been initialised at the beginning of the code with the line *reader = easyocr.Reader(['en'])*.

The *easyOCR* function then extracts the data from the image, which is saved to a text file again.

However, the *easyocr* reader doesn't return simply the string of the text read, but an array of tuples containing the bounding box, string and confidence level of each word detected. The array is structured as such:

[(bounding box of word 1, string of word 1, level of confidence of word 1), (bounding box of word 2, string of word 2, level of confidence of word 2),...].

Only the strings need to be extracted. Fortunately, the words detected are in the same order as what humans can read on the image, so the code can simply go through the array and take the strings.

Code listing 4.9: The *easyOCR* function

```
def easyOCR(fileName):
    #Use OCR algorithm
    data = reader.readtext(fileName)
    # print("Text recognized: ")
    # print(data)

    #Save data to a txt file (only recognized text)
    print('Name_of_txt_file:' + fileName[:-4] + 'EasyOCR.txt')
    with open(fileName+'EasyOCR.txt', 'w') as f:
        for i in range (len(data)):
            f.write(data[i][1])
```

kerasOCR works similarly to the *easyOCR* function. The *keras_ocr* is imported at the beginning of the code with *import keras_ocr*, and the *keras_ocr* pipeline is initialised with the line

pipeline = keras_ocr.pipeline.Pipeline(). The *kerasOCR* function can then use the pipeline to read the image and extract the text with the *pipeline.recognize* function. However, this time, the data extracted is again different from before, it is structured as an array: [(string of word 1, bounding box of word 1), (string of word 2, bounding box of word 2),...].

And this time the words are not in the same order as humans can read them on the image, which is why the functions *get_distance*, *distinguish_rows* and *sorting* are needed to sort the words in the right order.

An online post ([link here](#)) provided these three functions and an explanation on how they work. First, the *get_distance* function calculates, for each of the bounding boxes, its distance to the origin. Then *distinguish_rows* splits the words detected into sublists, each representing a different row of text. The function goes through the words detected and uses a threshold to separate the words based on their vertical distance to the origin. If the current word's vertical distance minus the previous word's vertical distance is higher than the threshold, then the word is in a different row. And finally *sorting* will call the previous two functions, and take the sublists and return a list of the words ordered in their reading order. This method worked well most of the time on the images used in the experiment, but might return some strange results if the text in the image is tilted. It is then more difficult to distinguish between rows using this method.

Code listing 4.10: The *get_distance*, *distinguish_rows* and *sorting* functions

```
def get_distance(predictions):
    """
    Function returns dictionary with (key,value):
```

```

    * text : detected text in image
    * center_x : center of bounding box (x)
    * center_y : center of bounding box (y)
    * distance_from_origin : hypotenuse
    * distance_y : distance between y and origin (0,0)
    """

# Point of origin
x0, y0 = 0, 0 # Generate dictionary
detections = []
for group in predictions:
    # Get center point of bounding box
    top_left_x, top_left_y = group[1][0]
    bottom_right_x, bottom_right_y = group[1][1]
    center_x = (top_left_x + bottom_right_x) / 2
    center_y = (top_left_y + bottom_right_y) / 2 # Use the Pythagorean
    Theorem to solve for distance from origin
    distance_from_origin = math.dist([x0,y0], [center_x, center_y]) #
    Calculate difference between y and origin to get unique rows
    distance_y = center_y - y0 # Append all results
    detections.append({
        'text':group[0],
        'center_x':center_x,
        'center_y':center_y,
        'distance_from_origin':distance_from_origin,
        'distance_y':distance_y
    })
return detections

def distinguish_rows(lst, thresh=15):
    """Function to help distinguish unique rows"""
    sublists = []
    for i in range(0, len(lst)-1):
        if lst[i+1]['distance_y'] - lst[i]['distance_y'] <= thresh:
            if lst[i] not in sublists:
                sublists.append(lst[i])
            sublists.append(lst[i+1])
        else:
            yield sublists
            sublists = [lst[i+1]]
    yield sublists

def sorting(predictions, thresh=15, order='yes'):
    """
    Function returns predictions in human readable order
    from left to right & top to bottom
    """
    predictions2 = get_distance(predictions)
    predictions2 = list(distinguish_rows(predictions2, thresh)) # Remove all
    empty rows
    predictions2 = list(filter(lambda x:x!=[], predictions2)) # Order text
    detections in human readable format
    ordered_preds = []
    ylst = ['yes', 'y']
    for pr in predictions2:
        if order in ylst:
            row = sorted(pr, key=lambda x:x['distance_from_origin'])
            for each in row:
                ordered_preds.append(each['text'])

```

The `kerasOCR` function calls the `sorting` function after having ran the keras OCR pipeline, to sort the words in the normal reading order. After that, the text detected is saved to a text file.

Code listing 4.11: The `kerasOCR` function

```
def kerasOCR(fileName):
    image = keras_ocr.tools.read(fileName)
    data = pipeline.recognize([image])
    # print("Text recognized: ")
    # print(data)
    text = sorting(data[0])
    # print(text)

    #Save data to a txt file (only recognized text)
    print('Name_of_txt_file: ' + fileName[:-4] + 'KerasOCR.txt')
    with open(fileName+'KerasOCR.txt', 'w') as f:
        for item in text:
```

4.1.4 Accuracy

The `Accuracy` file makes the assumption that the excel file has been filled with the OCR results and the corresponding ground truth. It takes the OCR results and their associated ground truth from the excel file, computes the Character Error Rate and puts it in the right cell in the excel file.

The `openpyxl` library is used to open the excel. A little manipulation is necessary, because the cells that contain the ground truth are written using a formula. Opening the excel normally means that only the formulas can be accessed, not the values. So it is necessary to open the excel file with the argument `data_only=True`. But if the excel file is saved after being opened with that argument, it erases all formulas in the sheets and only saves the values.

In the end, in order to avoid this, the excel file is opened twice, once as `readingDataframe` with the `data_only=True` argument and once as `writingDataframe` without it. The `readingDataframe` is used to access the values of the cells, while `writingDataframe` is used to enter the values of the CER and only that variable is saved to the excel file.

The code loops through all the lines with results, from line 2 to line 850. For each line, it gets the OCR result and the ground truth, updates them to an empty string if the cell was empty and puts both of them to lowercase-only strings. Then, it uses the `fastwer` library and its `score_sent` function with the argument `char_level=True` to compute the CER which is saved to the corresponding cell.

Code listing 4.12: The `Accuracy` file

```
import fastwer
import openpyxl

ExcelName = "Results.xlsx"
```

```

# Loading the results
readingDataframe = openpyxl.load_workbook(ExcelName, data_only=True)
readingSheet = readingDataframe.worksheets[3]          #The OCR results are on the
    fourth sheet

writingDataframe = openpyxl.load_workbook(ExcelName)
writingSheet = writingDataframe.worksheets[3]

for i in range(2,851):
    ocrText = readingSheet.cell(row = i, column = 8).value
    groundTruth = readingSheet.cell(row = i, column = 9).value

    if (ocrText == None):
        ocrText = ""
    if (groundTruth == None):
        groundTruth = ""

    ocrText = ocrText.lower()
    groundTruth = groundTruth.lower()

    cre = fastwer.score_sent(ocrText, groundTruth, char_level=True)
    writingSheet.cell(row=i, column=10).value = cre

writingDataframe.save(ExcelName)
writingDataframe.close()

```

4.2 Scenario

The code for the scenario is structured into 3 files: *scenarioMain.py*, *detector.py* and *OCR.py*. *scenarioMain.py* is the main file and will call the other two as it runs.

scenarioMain.py is structured similarly as *TakePictures.py*, with a main function and a `if __name__ == '__main__':` statement. The main function however is much longer than in *TakePictures.py* and separated into seven parts:

- Taking the picture from Pepper, which follows the same principle as in the experiment, except this time the resolution has been optimized with the result from the experiment
- Transferring the picture from Pepper to the computer with a paramiko connection,
- Using the object detection model on the image to detect the books, which uses the *detector.py* file
- Counting the number of books found and having Pepper say it out loud
- A loop going through each book found to create a cropped image
- Use the OCR model to read the text on the books on the cropped images
- And Pepper's reaction: displaying on its tablet the title of the books read by OCR.

The scenario mainly reuses the code from the experiments with some necessary changes.

One difficulty came from the python version to use. The libraries needed for the object detection and OCR are only available in python3, while the programs

for Pepper need to be run in python2.7.

To work around this issue, subprocesses had to be used. The subprocess makes it possible to spawn new processes. In the case of the object detection, a command string is create with *python3*, *detector.py* and the name of the image file to run the object detection model on. The subprocess is then started with the *subprocess.call* function.

Code listing 4.13: Use of a subprocess in *scenarioMain.py* for the object detection

```
# Object detection
imagePath = os.path.join(ComputerPath, f)
detection_command = ['python3', "detector.py", "--file" , imagePath]
# print("Python command: "+ detection_command)
startTime = time.time()
process = subprocess.call(detection_command)
elapsedTime = time.time() - startTime
print("Detection finished in %.2f seconds" % elapsedTime)
```

The *detector.py* file contains *if __name__ == '__main__':* statement which extracts with an argument parser the name and path of the picture to run the object detection model on, the same *detect* function as in the experiment, and a *main* function.

The *main* function calls the *detect* function and then stores the data found in a json file so that it will be available from the *scenarioMain.py* code.

Code listing 4.14: The *detector.py* file

```
import cv2
import cvlib as cv
import argparse
import json

def detect(fileName):
    im = cv2.imread(fileName)
    bbox, label, conf = cv.detect_common_objects(im)

    return(bbox, label, conf)

def main(fileName):
    #Collect data from the detector
    data = detect(fileName)

    #Save data to a json file
    json_obj = json.dumps(data)
    jsonFile = open(fileName + '.json', 'w')
    json.dump(json_obj, jsonFile)
    jsonFile.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("--file", type=str, help="Name of the picture to use the detector on")

    args = parser.parse_args()

    # print("args: ", args)
```



```
main(args.file)
```

Back in *scenarioMain.py*, the detection data is retrieved from the json file, and a quick loop allows us to extract the books found and their coordinates. Pepper can then react by talking and announcing the number of books found.

Code listing 4.15: Counting books in *scenarioMain.py*

```
# Counting books found
f = open(imagePath + '.json',)
print("JSON_File_Name:_" + imagePath[:-4] + '.json')
data = json.load(f)
data = json.loads(data)
booksFound = 0
booksCoordinates = []

for i in range(len(data[1])):
    if (data[1][i] == "book"):
        booksFound += 1
        booksCoordinates.append(data[0][i])
print("%s_books_found" % booksFound)
tts.say("I_have_found_" + str(booksFound) + "_books")
```

After this step, the code is again similar to the experiment. A loop goes through the books found and creates cropped images for each. The command is created in this loop, by adding to it the names of the cropped images files. A subprocess is called to execute the *OCR.py* file and run the OCR model on the cropped images. Said file is very similar in structure to *detector.py*. It was modified to take a list of the names of the files to use, the *detect* function was replaced by the function from the experiment corresponding to the chosen OCR, and the result is saved to a text file instead of a json file.

After the OCR has been run, the text recognized is retrieved from the text file and displayed on Pepper's tablet through the *tabletService.showWebview* and *tabletService.hideWebview* functions.

Code listing 4.16: Displaying text on Pepper's tablet

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("--ip", type=str, default="192.168.137.175",
                        help="Robot_IP_address._On_robot_or_Local_Naoqi:_use_\
```


Chapter 5

Results

This chapter will present and analyse the results of the experiment, starting with the object detection and finishing with the OCR. All results are available at *this repository* in the folder *0 - Experiment \1 - Results*. This includes the image and text files, as well as the excel document used to analyse them.

5.1 Object detection

The detection results consist whether each of the 6 books was detected in each of the 48 pictures, as shown in figure 5.1. Therefore, each picture needs 6 rows in the excel sheet, which gives us a total of 288 lines of detection results.

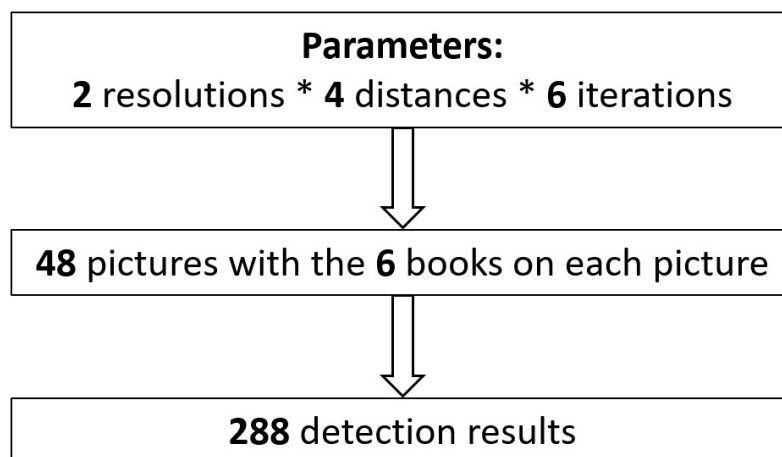


Figure 5.1: The number of detection results

As explained in the Methodology chapter, the detection results are separated into four categories:

- Detected: The book is in the bounding box and the only writing in the bounding box is on that book.

- Detected with another book: Two whole books are in the same bounding box, and none of the other bounding boxes contain these two books
- Detected partially with another book: The bounding box also contains a little bit of another book, enough that some but not all of the text is visible.
- Not detected: The book is not in any of the bounding boxes labeled as a book by the model.

Overall, most books were detected, with only a few instances where they were not. Table 5.1 details the exact numbers and percentages of each of the four categories over all the results.

Category	Number of occurrences	Percentage
Detected	272	94.4%
Detected partially with another book	8	2.8%
Detected with another book	6	2.1%
Not detected	2	0.7%

Table 5.1: The detection results

There are two parameters that can be adjusted for better results: the distance between the robot and the books and the resolution of the picture.

Table 5.2 shows the detection results separated by resolution.

Category	Resolution 3	Resolution 6
Detected	130 (90%)	142 (99%)
Detected partially with another book	6 (4%)	2 (1%)
Detected with another book	6 (4%)	0 (0%)
Not detected	2 (1%)	0 (0%)

Table 5.2: The detection results, separated according to resolution

It then appears that the majority of the detection failures happen with resolution 3, with 6 of the "Detected partially with another book", all 6 of the "Detected with another book" and both of the "Not detected".

To be even more precise, table 5.3 shows the detection results separated by both resolution and distance.

On this last table, one can see that most of the detection failures (9 out of the total 16) happen with resolution 3 and distance 70cm. The other configurations give much better detection performances. Resolution 6 with distances 45cm and 70cm have no detection failures, and the other two configuration with distances 50cm and 60cm, only have one "Detected partially with another book". For resolution 3, the least amount of detection failures come for a distance of 50cm, with only one "Not detected". Resolution 3 with distances 45cm and 60cm both have two "Detected partially with another book".

Category	Resolution 3				Resolution 6			
	Distance				Distance			
	45cm	50cm	60cm	70cm	45cm	50cm	60cm	70cm
Detected	34	35	34	27	36	35	35	36
Detected partially with another book	2	0	2	2	0	1	1	0
Detected with another book	0	0	0	6	0	0	0	0
Not detected	0	1	0	1	0	0	0	0

Table 5.3: The detection results, separated according to resolution and distance

5.2 OCR

As explained in the Methodology chapter, the OCR results are evaluated after excluding the failed detection results of "Detected partially with another book" or "Detected with another book". As shown in figure 5.2, this leaves us with 272 cropped pictures, and 3 OCR results for each cropped picture, so 816 OCR results in total.

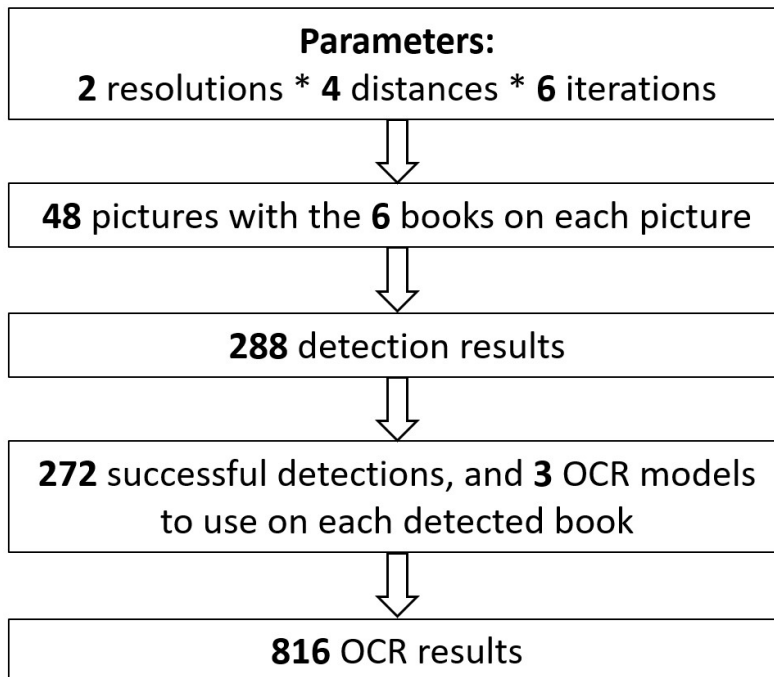


Figure 5.2: The number of OCR results

5.2.1 Parameters of the experiment

There are three parameters that can be adjusted for better results: the distance between the robot and the books, the resolution of the picture, and the OCR model used.

The first parameter to look at is the OCR model. Table 5.4 and figure 5.3 show the average CER and the Standard Deviation of the CER, separated by OCR model.

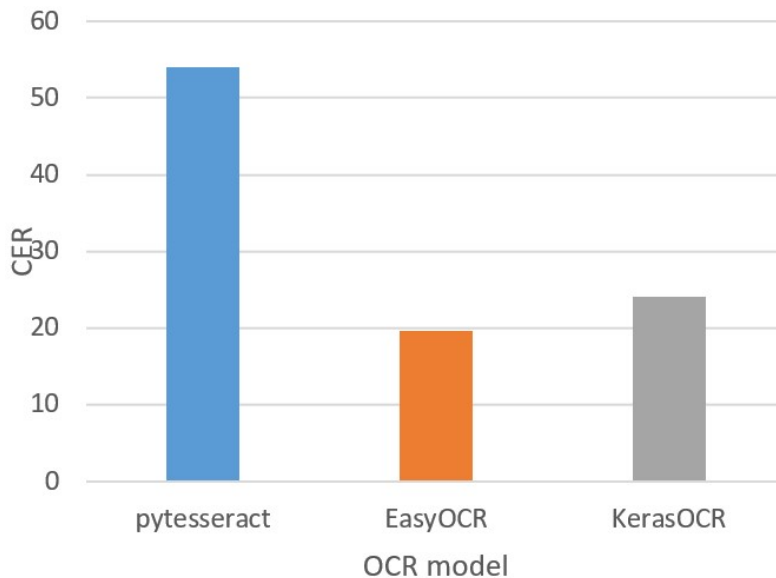


Figure 5.3: The average CER, separated according to OCR model

OCR model	Average CER	Standard deviation
pytesseract	54.1	37.3
EasyOCR	19.6	16.0
KerasOCR	24.1	21.6

Table 5.4: The OCR results, separated according to OCR model

The table shows EasyOCR as the most accurate model, as it has the lowest average CER, and the most consistent as it has the lowest standard deviation of CER. KerasOCR is slightly higher in both of those values by about 5, while pytesseract performs much worse, its values being over twice those of EasyOCR.

Pytesseract's high average and standard deviation are further confirmed by looking at the upper values of CER associated. It has 57 occurrences (out of 272 results) of a CER equal to or higher than 100, 40 of which are the result of the OCR returning a blank string when trying to read an image and thus getting a CER of 100. A CER higher than 100 is an indication that the OCR results is really bad, as it means that it would be easier to type all the characters of the actual title than to edit the OCR result.

It should be noted that the consistency is not very good in these results. The values of the standard deviation is very high and close to the averages. It demonstrates that the CER is not very consistent for any of the OCR models (or as will be seen, for any of the parameters), and has a wide range of values. While this consistency could be improved in further works, it is still possible to compare the current values to find the least inconsistent of the parameter.

The parameters of distance and resolution can also be compared in the same way. Figure 5.4 shows the average CER for both resolutions including all OCR models, and table 5.5 details the values of the average CER and the standard deviation of the CER.

Resolution	Average CER	Standard deviation
3	33.4	31.6
6	31.9	29.7

Table 5.5: The OCR results, separated according to resolution

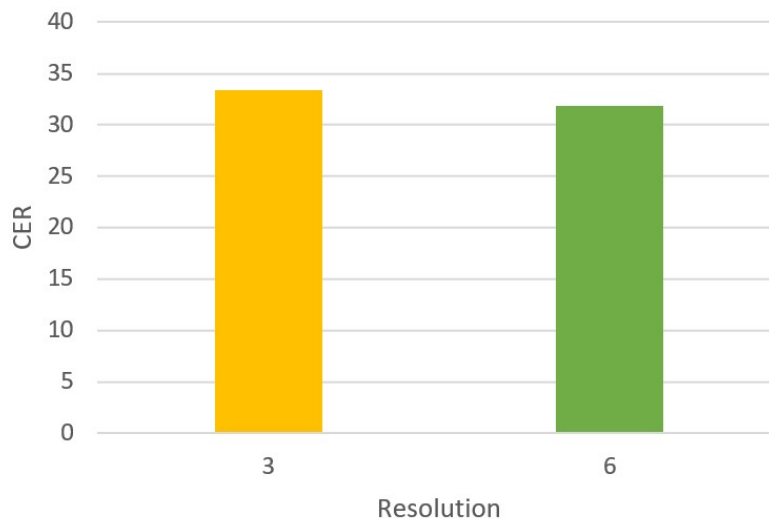


Figure 5.4: The average CER, separated according to resolution

This time, there is not a big difference in the accuracy and consistency of both resolutions. The resolution 6 seems to be slightly better, but only by a difference of less than 2 for both average and standard deviation. Once again and even more than previously, the standard deviation are quite high and their values are very close to the values of the average.

The accuracy and consistency by distance are presented in figure 5.5 and table 5.6.

The accuracy values vary a bit more this time. The optimal distance for accuracy is 50cm, with an average CER of 29.7. The standard deviation on the other

Distance	Average CER	Standard deviation
45cm	33.5	29.5
50cm	29.7	31.7
60cm	32.2	29.8
70cm	35.3	31.4

Table 5.6: The OCR results, separated according to distance between the robot and the books

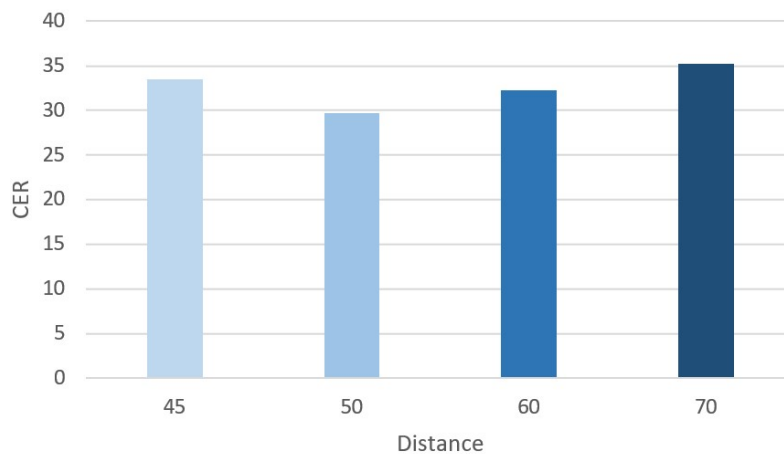


Figure 5.5: The average CER, separated according to distance

hand varies less than 2. The lowest value is 45cm with a standard deviation of 29.5, and the highest is 50cm with a standard deviation of 31.7.

To get more perspective, the parameters can be compared in the same table. Table 5.7 and figure 5.6 show the average CER and the standard deviation of the CER for each combination of values of resolution and distance.

Distance	Resolution	Average CER	Standard deviation
45cm	3	34.5	31.4
	6	32.5	27.8
50cm	3	26.0	33.9
	6	33.4	29.1
60cm	3	33.4	28.1
	6	31.2	31.5
70cm	3	41.6	31.4
	6	30.5	30.7

Table 5.7: The OCR results, separated according to distance and resolution

The table shows a clear optimal accuracy for a resolution of 3 and a distance of 50cm, with an average CER of 26.0. It is the only distance for which a resolution of

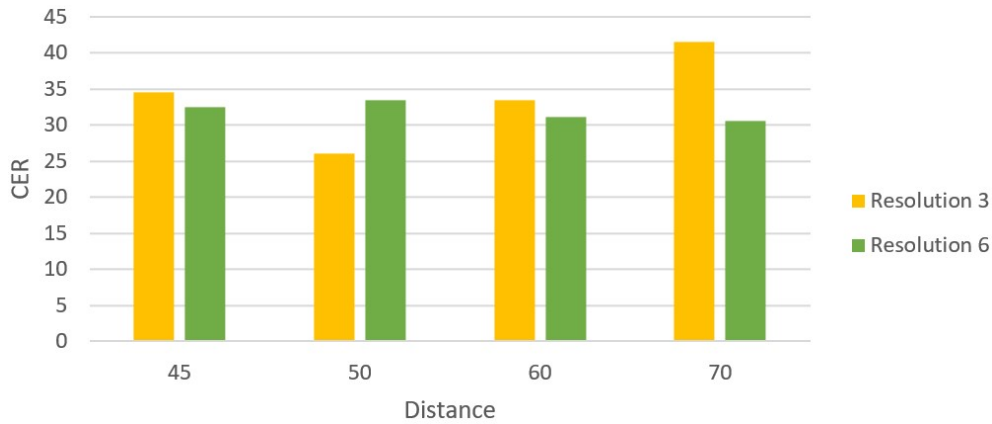


Figure 5.6: The average CER, separated according to distance and resolution

3 is better than a resolution of 6. In terms of consistency however, it is the worst, with a standard deviation of 33.9. The lowest standard deviation values are 27.8 for a resolution of 6 and distance of 45cm, but these values also correspond to an average CER of 34.5, 8.5 higher than the lowest value.

Table 5.7 and figure 5.8 show the average and standard deviation of the CER by distance and OCR model.

Distance	OCR model	Average CER	Standard deviation
45cm	pytesseract	55.9	35.6
	EasyOCR	22.3	18.0
	kerasOCR	22.2	16.9
50cm	pytesseract	55.5	40.6
	EasyOCR	15.7	12.2
	kerasOCR	17.9	15.4
60cm	pytesseract	50.4	36.5
	EasyOCR	19.8	15.2
	kerasOCR	26.5	24.6
70cm	pytesseract	54.4	36.94
	EasyOCR	20.9	17.8
	kerasOCR	30.5	26.5

Table 5.8: The OCR results, separated according to OCR model and distance

Once again, the difference between accuracy of the three OCR models is stark. Pytesseract gets a much higher CER average than the other two, and while EasyOCR and kerasOCR have almost the same average at 45cm, the difference between them gets bigger with distance with EasyOCR being more accurate. The optimal parameters for accuracy in this table is for EasyOCR with a distance of 50cm. It also has the lowest standard deviation.

Table 5.8 and figure 5.9 show the average and standard deviation of the CER

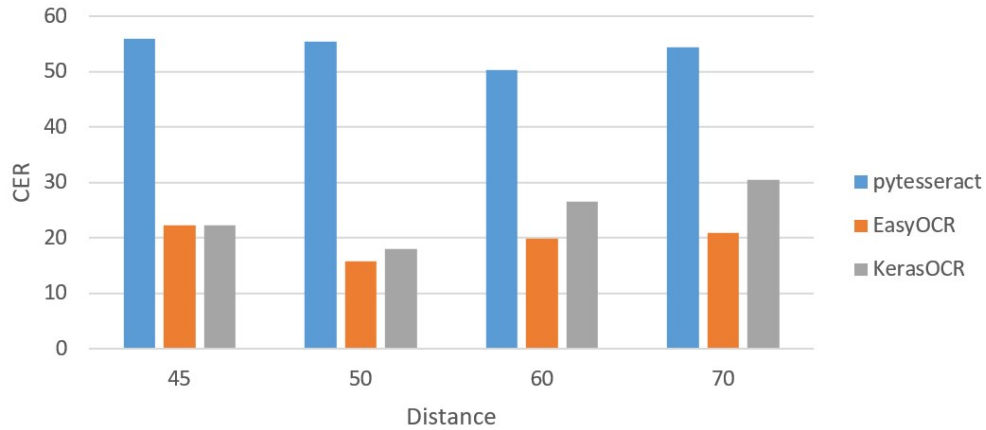


Figure 5.7: The average CER, separated according to OCR model and distance

by resolution and OCR model.

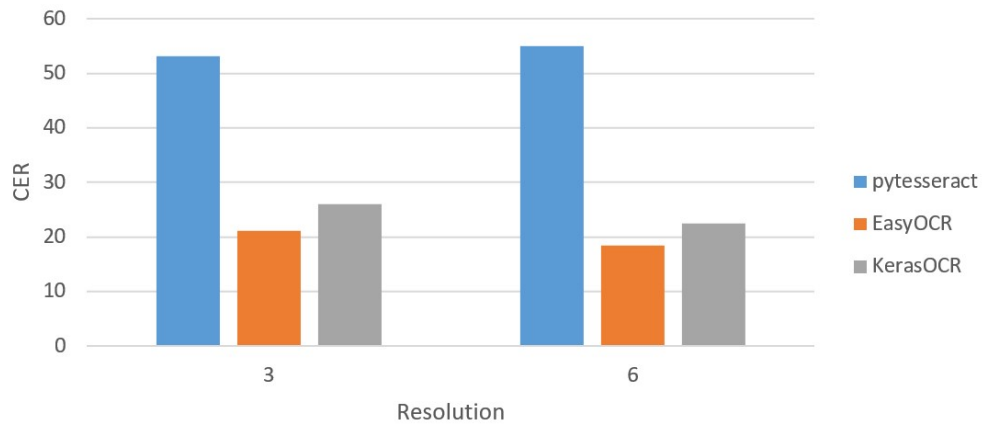


Figure 5.8: The average CER, separated according to OCR model and resolution

Distance	OCR model	Average CER	Standard deviation
3	pytesseract	53.1	40.2
	EasyOCR	21.1	14.0
	kerasOCR	26.0	24.4
6	pytesseract	54.9	34.6
	EasyOCR	18.3	17.7
	kerasOCR	22.5	18.6

Table 5.9: The OCR results, separated according to OCR model and resolution

The order of the OCR models remains obvious, with EasyOCR performing the best, followed by kerasOCR and pytesseract which is quite worse than the other two. This is true for both accuracy and consistency. The best accuracy is

with EasyOCR and the resolution of 6, with an average CER of 18.3, followed by EasyOCR and the resolution of 3, with an average of 21.1. On the other hand, the optimal consistency corresponds to EasyOCR and the resolution of 3, for a standard deviation of 14.0, followed by EasyOCR and a resolution of 6 with a standard deviation of 17.7.

So far, the results have shown that the best OCR model is EasyOCR, and that for the distance, 45cm gets the highest consistency, but 50m gets the highest accuracy. The resolution doesn't have very clear results, while the resolution of 6 seemed the best in the original table 5.5, both resolutions traded the highest accuracy and consistency in the tables 5.7 and 5.8. In order to decide a final set of optimal parameters, table 5.10 and figure 5.9 show the average and standard deviation of the CER by distance, resolution and OCR model.

Resolution	Distance	OCR model	Average CER	Standard deviation
3	45cm	pytesseract	59.8	38.7
		EasyOCR	20.1	10.6
		kerasOCR	23.4	20.1
	50cm	pytesseract	50.1	47.6
		EasyOCR	13.4	9.2
		kerasOCR	14.6	16.6
	60cm	pytesseract	48.1	34.4
		EasyOCR	22.9	13.9
		kerasOCR	29.2	26.1
	70cm	pytesseract	55.0	39.5
		EasyOCR	29.9	17.4
		kerasOCR	39.8	28.9
6	45cm	pytesseract	52.2	32.5
		EasyOCR	24.3	23.0
		kerasOCR	21.0	13.5
	50cm	pytesseract	60.9	32.0
		EasyOCR	18.0	14.3
		kerasOCR	21.3	13.6
	60cm	pytesseract	52.6	38.8
		EasyOCR	16.9	16.0
		kerasOCR	24.0	23.1
	70cm	pytesseract	54.0	35.5
		EasyOCR	14.1	15.0
		kerasOCR	23.5	22.4

Table 5.10: The OCR results, separated according to OCR model, distance and resolution

From these numbers, the best set of parameters is to use EasyOCR, with a distance of 50cm and a resolution of 3. This achieves both the highest accuracy, with an average of 13.4, and the highest consistency with a standard deviation of

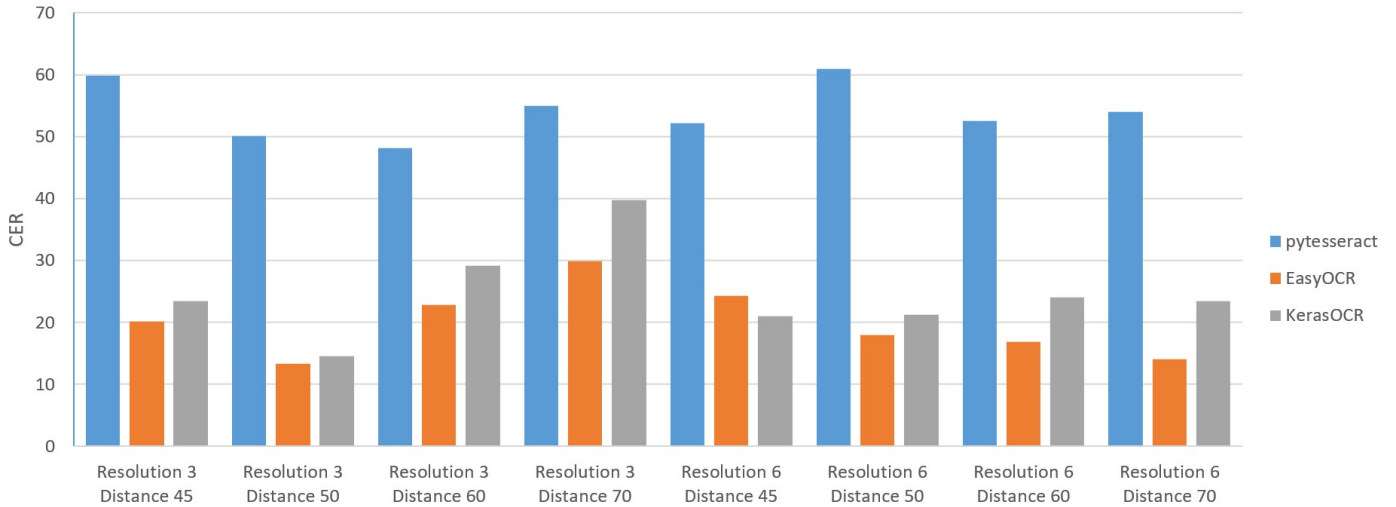


Figure 5.9: The average CER, separated according to OCR model, distance and resolution

9.2. These parameters should be used by Pepper in the scenario, to get the best OCR results

5.2.2 OCR results across books

While the books are not parameters that can be changed to improve the performance, it is still interesting to look at how well the OCR worked across the books.

Notably, the first three books (identified as 0, 1 and 2) have darker text (blue or black) on a light background (white for 1 and 2, and light blue for 0). Book n°3 is also the only one who doesn't have any kind of pattern in the background of the text, being purely blue. The text on each book also has different sizes:

- For book n°0, the text is 0.9cm high, except 'and others' which is 0.4cm high.
- For books n°1 and 2, the text is 0.7cm high.
- For book n°3 and 4, the text is 0.9cm high.
- For book n°5, the text is 1.1cm high, except for the last part "IP, UMTS, EGPRS and ATM" which is 0.5cm high.

Table 5.11 and figure 5.10 show the average and standard deviation of the CER by book.

The accuracy and consistency vary widely across the books. The book with the best accuracy is book n°3, with an average CER of 18.5. It is only the second most consistent, with a standard deviation of 22.9, behind book n°5 with a standard deviation of 21.5. However, book n°5 has a much higher average CER at 35.4, so it is safe to say that the best recognized title is that of book n°3.

It is possible to look at how both the colors of the text and its background, and the size of the text affect the OCR accuracy and consistency. In this case, they

Book number	Average CER	Standard deviation
0	37.8	30.2
1	32.5	33.6
2	43.0	37.8
3	18.5	22.9
4	28.6	28.9
5	35.4	21.5

Table 5.11: The OCR results, separated according to book

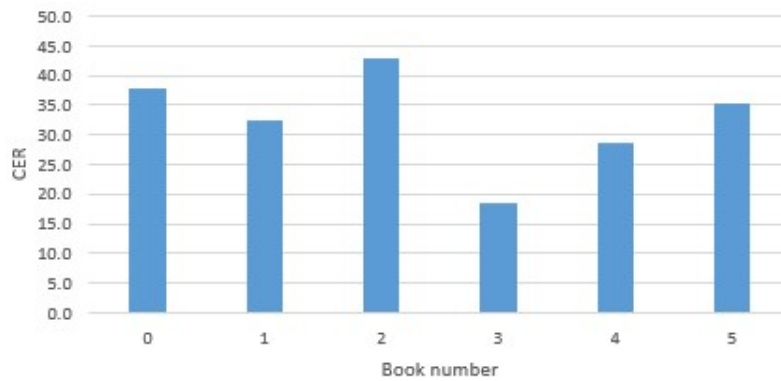


Figure 5.10: The average CER, separated according to book

don't seem to have a very strong impact.

Looking at colors, the light books (0,1,2) have a higher average and standard deviation than the dark books (3,4,5), but this grouping doesn't really make sense. Book n°5, despite being in the second group, has a higher average than book n°1 from the first book, and book n°4's standard deviation is closer to that of the first group than of books n°3 and 5.

As for the size of the text, the books could be grouped into 3 categories: Books with small text (1,2), Books with big text (3,4) and Books with big text and very small text (0,5). However, this separation again doesn't fit the values of the table. Books n°0 and 5 have similar averages, but very different standard deviations. Books n°1 and 2 have a difference of over 10 in average and over 4 in standard deviation, and books n°3 and 4 have the same difference between themselves.

While the books show substantially different results in both accuracy and consistency, it is difficult to link this to the colors of the books or the size of the texts with these results.

Chapter 6

Discussion

As robots enter more of our daily lives, it becomes natural to imagine new uses for them. This study works on expanding the presence and uses of social robots, by integrating them into the library. It has explored a way for the social robot Pepper to identify books in its environment by reading their title on the spine. To this end, object detection and OCR were employed, and different parameters were studied and compared to optimize the accuracy and consistency of both the detection and identification of books.

The purpose of this research was to answer these research questions:

- *How can we integrate machine learning with Pepper to enable it to read book titles using its camera?*
- *Which machine learning algorithms and models related to object detection and OCR can be used and are most efficient for this goal considering different parameters?*
- *To what extent can this integration of machine learning with Pepper be successfully used in the library?*

6.1 Integration of machine learning with Pepper

The machine learning was integrated with Pepper through several python libraries. The main problems of this integration were actually related to Pepper itself. As mentioned previously, Pepper works with python 2.7, while the libraries are in python3. This incompatibility forced the use of subprocesses, which considerably slows down the execution of the scenario.

The other problem is that Pepper's internal computer only has a CPU and cannot run the libraries itself, at least not fast enough to be considered a solution. Therefore, the pictures had to be transferred to the computer, analysed by the object detection and OCR models, and only then did the results make their way back to Pepper to be displayed. This problem is related to Pepper and cannot be solved, but there are ways to work around it. Two different studies [40, 41] propose two object recognition pipelines for Pepper. The first one uses a Jetson TK1, an em-

bedded computing board from Nvidia, attached to Pepper by a custom-designed backpack, to run a version of the Yolo algorithm. The second one uses a different object recognition pipeline and runs it on a Jetson TX1, a more recent and faster successor to the Jetson TK1. Both of these options allow Pepper to be more autonomous, and as a future work on this topic, the integration of a similar solution for this scenario could be useful as well.

For more details, the implementation was fully described in chapter 4.

6.2 Object detection and OCR

In the experiment, different parameters were analysed and compared to find the optimal configuration that would most consistently and accurately be able to read the text on the spine of the books. The results showed that EasyOCR was the model that was best suited to this task, and with a resolution of 3 for the pictures, which should be taken when Pepper is 50cm away from the books. While the detection results with these parameters were not perfect, they still achieved a 97% rate of recognition of the books, and the parameters allow for the highest accuracy and consistency in the OCR results. These optimal parameters from the experiment give an answer to the second research question: object detection with Yolov5 and OCR with EasyOCR, paired with the right distance to the books and the right resolution for the picture, are the most efficient models to read the book titles.

Other studies have compared different OCR models, but usually not for the purpose of reading book titles. A very common application is reading license plates on cars. A study [42] uses Yolov5 to detect the plate, and compares EasyOCR and Tesseract OCR. Another study [43] uses another framework to detect the plates, but also compares EasyOCR and Tesseract OCR. A third study [44] compares two procedures: using OCR models to read license plates on cars directly from images with cars, or from the same images previously cropped to just the car using the YOLOv3 model. This last study also compares four different OCR models: EasyOCR, Keras-OCR, CNN, WPOD-Net. All three studies find that EasyOCR is the most accurate model of all models compared. This validates the results of this study, and seem to corroborate the idea that EasyOCR is the most accurate model on scene images, such as pictures of cars or books.

6.3 Scenario

The optimal parameters found in the results of the experiment were used in the scenario. The *repository* that contains the code also has some examples of the scenario being run with Pepper in the folder *1 - Scenario*, with, for each, the image taken by Pepper, its cropped versions, the json file with the results from the object detection, the text files with the results of the OCR and a video¹ of Pepper performing the scenario.

¹The videos need to be downloaded to be viewed.

In order to answer the third research question, the scenario shows how much Pepper can read on the books in the actual situation. Looking at example 1 (which you can find here), Pepper executed the scenario and found the six books. Here is the text displayed by Pepper on its tablet:

Book 0:FIXED BROADBANDWIRELESS SYSTEM DESIGN

Book 1:Handbook ol Algorithms for WirelessFNetworking and Mobile Computing

Book 2:Convergence TechnologiesJufor 36 NetworksIp Umts EGpRS and ATM

Book 3:La1nResource, Mobility; and Security ManagementWireless Networks and Mobile Communications

Book 4:SOLUTIONSondrPROBLEMSAllie BIOSh

Book 5:EVOLVED PACKET SYSTEM(EPS)

While the OCR results are not the exact titles of the books, they are close enough to them that humans can identify each book, and for Pepper, the CER of each results compared to the expected result for the corresponding books (shown in table 6.1) are also quite low, although they vary for each book.

OCR string	Expected result	CER
FIXED BROADBANDWIRELESS SYSTEM DESIGN	FIXED BROADBAND WIRELESS SYSTEM DESIGN	2.6
Handbook ol Algorithms for WirelessFNetworking and Mobile Computing	Handbook of Algorithms for Wireless Networking and Mobile Computing	3.0
Convergence TechnologiesJufor 36 NetworksIp Umts EGpRS and ATM	Convergence Technologies for 3G Networks IP, UMTS, EGPRS and ATM	12.1
La1nResource, Mobility; and Security ManagementWireless Networks and Mobile Communications	Resource, Mobility, and Security Management in Wireless Networks and Mobile Communications	10.0
SOLUTIONSondrPROBLEMSAllie BIOSh	Solutions and other problems Allie Brosh	25.0
EVOLVED PACKET SYSTEM(EPS)	EVOLVED PACKET SYSTEM (EPS)	3.7

Table 6.1: The CER results from example 1

To answer the third research question, this study has managed to integrate machine learning with Pepper so that Pepper can read, although not perfectly, the titles on the spine of books in front of it. The integration was good enough to successfully execute the scenario.

The idea of using object detection and OCR to contribute to a robot's understanding of its environment is not new. A study [45] used object detection and OCR to recognize elevator buttons for service robots, while another [46] employs them to recognize medicine bottles and guide a robotic arm to grab them. However, to the best of our knowledge, it is the first time that it has been used with this specific social robot, or for the purpose of identifying books.

6.4 Limitations

This research has some limitations which should be taken into account, mainly concerning the project's scope. The choices made limit the generalization of the results of this study.

The books used for the experiment were chosen to simplify the scenario, they are all very thick with big text, but this work has not explored the performance of the OCR on smaller book with smaller text. The books were also limited to english text, so it is not known if other languages and alphabets, namely Norwegian, would get the same results.

The choices made when it came to Machine Learning also limits the scope of this work. One object detection model and three OCR models were used, but other models might yield good, even better results. It would be possible to train new models on custom dataset to try to reach better accuracy and consistency in both object detection and OCR. It could also be helpful to employ pre-processing techniques on the pictures taken by the robot to increase results in both tasks. The luminosity in the image, for example, can play a role in the detection and recognition of both objects and text, but can be modified and corrected in pre-processing, for example through a binarization of the image (the process of turning a coloured image into a black and white image).

6.5 Implications

Despite the mentioned limitations, this research is an important first step in the use of Pepper in the library. It has demonstrated a way to make Pepper read books' titles using machine learning. This work can be used as a base to develop new applications and scenarios of Pepper in the library.

It has enabled Pepper to perceive more precisely its environment and learn from it. Outside of the idea of using Pepper in the library, this is an important result that can be employed in a variety of different uses, as it widens the scope of Pepper's perception. It can help Pepper develop new abilities and complexify the scenarios in which Pepper can be employed. This brings up new opportunities for working with Pepper in different situations.

Chapter 7

Conclusion and Future Work

This main goal of this work was to explore the possibility of using object detection and OCR to allow Pepper to read book titles. The experiment compares several parameters and OCR models, and found the optimal combination of parameters which allows Pepper to read the text on the books in front of it.

Although the text read by Pepper doesn't correspond exactly to what is written on the books, the results are close enough that both humans and machine can recognize them. This work does face some limitations, but it presents a prototype that achieves its goal.

This research provides a novel contribution to the field of social robotics, by expanding Pepper's perception of its environment through the use of computer vision algorithms. It is a base upon which can be built many more applications and scenarios for Pepper.

There are several aspects of this work that could be further developed in future works on this topic. The scenario could be made more complex. The books used could be of a wider diversity: thinner books that might be harder to read, or books in other languages for example. They could also be positioned as books are in the library, which might require some pre-processing for the OCR to have the text the right way up. In this study, the books are also already at the right height for Pepper's camera, so solutions could be explored for bookshelves that go from the floor to higher than Pepper. Other machine learning models could be used to try to improve the results of this work. For example, it would be possible to train custom models to perform the object detection and OCR.

Further work could explore the navigation of the library by Pepper. In this work, Pepper was positioned so that it would be in front of the books, but future works could include Pepper putting itself in front of the books, at the right distance and angle.

The use of subprocesses slowed down the flow of the scenario, as the object detection and OCR itself were much faster than the opening of the new subprocesses that use them. These subprocesses were necessary in this study to work around the problem of Pepper needing an older version of Python than the libraries used. This topic could be explored to improve the speed at which Pepper is

able to read what is in front of it.

This research has managed to complete its goal to use machine learning to enable Pepper to read book titles, and has opened up many opportunities for further work on this topic.

Bibliography

- [1] G. L. Anna Henschel and E. S. Cross, ‘What makes a robot social? a review of social robots from science fiction to a home or hospital near you,’ *Current Robotics Reports*, vol. 2, no. 1, pp. 9–19, Mar. 2021. DOI: 10.1007/s43154-020-00035-0.
- [2] M. Sarrica, S. Brondi and L. Fortunati, ‘How many facets does a “social robot” have? a review of scientific and popular definitions online,’ *Information Technology & People*, vol. 33, Apr. 2019. DOI: 10.1108/ITP-04-2018-0203.
- [3] Z. Su, W. Sheng, G. Yang, A. Bishop and B. Carlson, ‘Adaptation of a robotic dialog system for medication reminder in elderly care,’ *Smart Health*, vol. 26, 2022.
- [4] L. Elloumi, M. Bossema, S. M. De Droog, M. H. J. Smakman, S. V. Ginkel, M. E. U. Ligthart, K. Hoogland, K. V. Hindriks and S. B. Allouch, ‘Exploring requirements and opportunities for social robots in primary mathematics education,’ in *RO-MAN 2022 - 31st IEEE International Conference on Robot and Human Interactive Communication: Social, Asocial, and Antisocial Robots*, 2022, pp. 316–322.
- [5] M. Donnermann, P. Schaper and B. Lugin, ‘Investigating adaptive robot tutoring in a long-term interaction in higher education,’ in *RO-MAN 2022 - 31st IEEE International Conference on Robot and Human Interactive Communication: Social, Asocial, and Antisocial Robots*, 2022, pp. 171–178.
- [6] J. Musa, *How can humanoid robots modernize your reception?* Nov. 2020. [Online]. Available: <https://www.aldebaran.com/en/blog/news-trends/how-can-humanoid-robots-modernize-your-reception>.
- [7] R. Mehta and A. Sahu, ‘Autonomous robot for inventory management in libraries,’ in *2020 IEEE International Students’ Conference on Electrical, Electronics and Computer Science, SCECS 2020*, 2020.
- [8] X. Yu, Z. Fan, H. Wan, Y. He, J. Du, N. Li, Z. Yuan and G. Xiao, ‘Positioning, navigation, and book accessing/returning in an autonomous library robot using integrated binocular vision and qr code identification systems,’ *Sensors*, vol. 19, p. 783, Feb. 2019. DOI: 10.3390/s19040783.

- [9] S. Krishnan, V. Singh, P. Shah, A. Yadav, G. Panampilly, S. Saha and H. Shukla, 'Development of an rfid-based semi-autonomous robotic library management system,' Oct. 2020, pp. 26–31. DOI: 10.1109/ICACR51161.2020.9265493.
- [10] J. Liu, F. Zhu, Y. Wang, X. Wang, Q. Pan and L. Chen, 'Rf-scanner: Shelf scanning with robot-assisted rfid systems,' in *Proceedings - IEEE INFOCOM*, 2017.
- [11] L. Shangguan and K. Jamieson, 'The design and implementation of a mobile rfid tag sorting robot,' in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '16, New York, NY, USA: Association for Computing Machinery, 2016, pp. 31–42, ISBN: 9781450342698. DOI: 10.1145/2906388.2906417. [Online]. Available: <https://doi.org/10.1145/2906388.2906417>.
- [12] M. Jampour, A. KarimiSardar and H. Estakhroyeh, 'An autonomous vision-based shelf-reader robot using faster r-cnn,' *Industrial Robot*, vol. ahead-of-print, Feb. 2021. DOI: 10.1108/IR-10-2020-0225.
- [13] X. Chaoying, 'Research on classification and identification of library based on artificial intelligence,' *Journal of Intelligent & Fuzzy Systems*, vol. 40, pp. 1–13, Dec. 2020. DOI: 10.3233/JIFS-189524.
- [14] S. Zhou, T. Sun, X. Xia, N. Zhang, B. Huang, G. Xian and X. Chai, 'Library on-shelf book segmentation and recognition based on deep visual features,' *Information Processing & Management*, vol. 59, no. 6, p. 103 101, 2022, ISSN: 0306-4573. DOI: <https://doi.org/10.1016/j.ipm.2022.103101>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0306457322002023>.
- [15] N. Nam, N. Nam and N. Truong Thinh, 'Using sift and sliding window to detect and invent literature for library robot,' *International Journal of Mechanical Engineering and Robotics Research*, pp. 386–391, Jan. 2021. DOI: 10.18178/ijmerr.10.7.386-391.
- [16] H. Pham, A. Giordano, L. Miller, J. Giannitti, M. Mena and A. DiNardi, 'A ubiquitous approach for automated library book location management,' Sep. 2018, pp. 78–82, ISBN: 978-1-4503-6540-6. DOI: 10.1145/3277104.3277115.
- [17] M. Prats, E. Martínez, P. J. Sanz and A. P. Del Pobil, 'The uji librarian robot,' *Intelligent Service Robotics*, vol. 1, no. 4, pp. 321–335, 2008.
- [18] P. Azevedo. 'Medium.' (2022), [Online]. Available: <https://medium.com/@pedroazevedo6/object-detection-state-of-the-art-2022-ad750e0f6003>.
- [19] K. He, G. Gkioxari, P. Dollár and R. Girshick, 'Mask r-cnn,' in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.

- [20] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, 'You only look once: Unified, real-time object detection,' in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.
- [21] C.-Y. Wang, A. Bochkovskiy and H.-Y. M. Liao, *Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, 2022. arXiv: 2207.02696 [cs.CV].
- [22] D. Berchmans and S. S. Kumar, 'Optical character recognition: An overview and an insight,' in *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, 2014, pp. 1361–1365. DOI: 10.1109/ICCICCT.2014.6993174.
- [23] R. Avyodri, S. Lukas and H. Tjahyadi, 'Optical character recognition (ocr) for text recognition and its post-processing method: A literature review,' in *2022 1st International Conference on Technology Innovation and Its Applications (ICTIIA)*, 2022, pp. 1–6. DOI: 10.1109/ICTIIA54654.2022.9935961.
- [24] Q. Ye and D. Doermann, 'Text detection and recognition in imagery: A survey,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 7, pp. 1480–1500, 2015. DOI: 10.1109/TPAMI.2014.2366765.
- [25] *Pypi*, 2014. [Online]. Available: <https://pypi.org/project/pytesseract/>.
- [26] R. Smith, 'An overview of the tesseract ocr engine,' in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 2, 2007, pp. 629–633. DOI: 10.1109/ICDAR.2007.4376991.
- [27] A. Parker, *Optical character recognition: Then and now*, May 2022. [Online]. Available: <https://wandb.ai/andrea0/optical-character-reports/Optical-Character-Recognition-Then-and-Now--VmlldzoyMDY0Mzc0>.
- [28] *Jaided ai*. [Online]. Available: <https://www.jaided.ai/easyocr/documentation/>.
- [29] *Keras-ocr*. [Online]. Available: <https://keras-ocr.readthedocs.io/en/latest/>.
- [30] *Paddleocr*. [Online]. Available: <https://github.com/PaddlePaddle/PaddleOCR>.
- [31] *Optical character recognition using paddleocr*, Jun. 2022. [Online]. Available: <https://learnopencv.com/optical-character-recognition-using-paddleocr/#PaddleOCR-models-comparison>.
- [32] *Calamari ocr*. [Online]. Available: <https://calamari-ocr.readthedocs.io/en/latest/>.
- [33] C. Wick, C. Reul and F. Puppe, 'Calamari - A High-Performance Tensorflow-based Deep Learning Package for Optical Character Recognition,' *Digital Humanities Quarterly*, vol. 14, no. 1, 2020.

- [34] C. Neudecker, K. Baierer, M. Gerber, C. Clausner, A. Antonacopoulos and S. Pletschacher, 'A survey of ocr evaluation tools and metrics,' in *The 6th International Workshop on Historical Document Imaging and Processing*, ser. HIP '21, Lausanne, Switzerland: Association for Computing Machinery, 2021, pp. 13–18, ISBN: 9781450386906. DOI: 10.1145/3476887.3476888. [Online]. Available: <https://doi.org/10.1145/3476887.3476888>.
- [35] T. A. Nartker and S. V. Rice, 'Measuring the accuracy of page-reading systems,' 1996.
- [36] V. I. Levenshtein, 'Binary codes capable of correcting deletions, insertions and reversals,' *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, Feb. 1966, Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.
- [37] A. K. Pandey and R. Gelin, 'A mass-produced sociable humanoid robot: Pepper: The first machine of its kind,' *IEEE Robotics and Automation Magazine*, vol. 25, no. 3, pp. 40–48, 2018.
- [38] A. Ponnusamy, *Cvlib - high level computer vision library for python*, <https://github.com/arunponnusamy/cvlib>, 2018.
- [39] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, *Yolov4: Optimal speed and accuracy of object detection*, 2020. arXiv: 2004.10934 [cs.CV].
- [40] E. Reyes, C. Gómez, E. Norambuena and J. Ruiz-del-Solar, 'Near real-time object recognition for pepper based on deep neural networks running on a backpack,' in *RoboCup 2018: Robot World Cup XXII*, D. Holz, K. Genter, M. Saad and O. von Stryk, Eds., Cham: Springer International Publishing, 2019, pp. 287–298, ISBN: 978-3-030-27544-0.
- [41] J. A. Castro-Vargas, A. Garcia-Garcia, S. Oprea, S. Orts-Escolano and J. Garcia-Rodriguez, 'Detecting and manipulating objects with a social robot: An ambient assisted living approach,' in *ROBOT 2017: Third Iberian Robotics Conference*, A. Ollero, A. Sanfeliu, L. Montano, N. Lau and C. Cardeira, Eds., Cham: Springer International Publishing, 2018, pp. 613–624, ISBN: 978-3-319-70833-1.
- [42] D. Vedhaviyassh, R. Sudhan, G. Saranya, M. Safa and D. Arun, 'Comparative analysis of easyocr and tesseractocr for automatic license plate recognition using deep learning algorithm,' in *2022 6th International Conference on Electronics, Communication and Aerospace Technology*, 2022, pp. 966–971. DOI: 10.1109/ICECA55336.2022.10009215.
- [43] N. Awalgaonkar, P. Bartakke and R. Chaugule, 'Automatic license plate recognition system using ssd,' in *2021 International Symposium of Asian Control Association on Intelligent Robotics and Industrial Automation (IRIA)*, 2021, pp. 394–399. DOI: 10.1109/IRIA53009.2021.9588707.

- [44] A. Bhardwaj, K. S. Srivastava, A. Kar and S. Gupta, 'A novel approach to recognize optical characters of number plate using object detection,' in *Futuristic Trends in Networks and Computing Technologies*, P. K. Singh, S. T. Wierchoń, J. K. Chhabra and S. Tanwar, Eds., Springer Nature Singapore, 2022, pp. 851–863, ISBN: 978-981-19-5037-7.
- [45] X. Tang, C. Wang, J. Su and C. Taylor, 'An elevator button recognition method combining yolov5 and ocr,' *Computers, Materials and Continua*, vol. 75, no. 1, pp. 117–131, 2023. DOI: 10.32604/cmc.2023.033327.
- [46] Z. Liu, K. Ding, Q. Xu, Y. Song, X. Yuan and Y. Li, 'Scene images and text information-based object location of robot grasping,' *IET Cyber-systems and Robotics*, vol. 4, no. 2, pp. 116–130, 2022. DOI: 10.1049/csy2.12049.



 **NTNU**

Norwegian University of
Science and Technology