Master's thesis

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

Endre Vee Hagestuen

Machine-learning techniques for human-robot collaboration in manufacturing

Master's thesis in Cybernetics and Robotics (Master, 5 years)
Supervisor: Jan Tommy Gravdahl
Co-supervisor: Mathias Hauan Arbo
June 2024

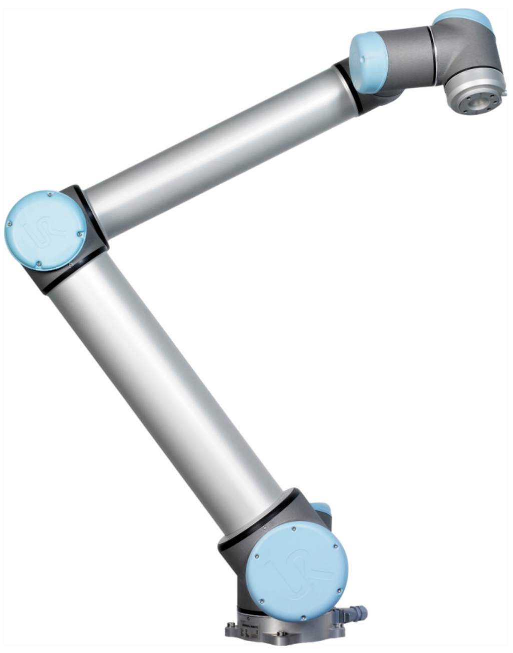


Image credits: Universal Robots

NTNU
Norwegian University of
Science and Technology

Endre Vee Hagestuen

Machine-learning techniques for human-robot collaboration in manufacturing

Master’s thesis in Cybernetics and Robotics (Master, 5 years)
Supervisor: Jan Tommy Gravdahl
Co-supervisor: Mathias Hauan Arbo
June 2024

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

Abstract

This thesis addresses the challenge of efficient human-robot collaboration in product manufacturing, focusing on machine-learning techniques and their potential applications. It explores hand tracking using MediaPipe's hand detector with a depth camera for robotic visual servo systems, re-training MediaPipe's object detector for new manufacturing object categories, and language model-driven robot task planning using the GPT model family. Firstly, the research compares ArUco marker tracking with MediaPipe and a depth camera, and the experimental results highlight the latter's promise for gloveless tracking for visual servo systems. However, the results also emphasize the need for further development to handle dynamic environments and situations where the operator holds tools while wearing work gloves. The thesis suggests future research into creating a custom dataset for training a hand-detection algorithm based on machine learning tailored for workspace situations, including scenarios where the operator is wearing work gloves. Next, re-training MediaPipe's object detector revealed challenges like image downscaling and model bias to contextual information. Despite these challenges, the thesis provides a step-by-step re-training guide and found that adding one object category to the model only requires 20 hours of work, including the time needed to create the dataset. Finally, the thesis found that integrating large language models with object detection shows potential for flexible task planning based on simple language instructions from human operators. This integration allows the robot to be aware of its surroundings through object detection, grounding the task planning in the real world. However, variability in responses from the language model affects the system's reliability. These findings demonstrate the feasibility and potential applications of machine learning in manufacturing and collaborative robotics.

Sammendrag

Denne masteroppgaven tar for seg utfordringene med effektivt samarbeid mellom mennesker og roboter i produktproduksjon, med fokus på bruk av maskinlæringsteknikker og deres potensielle anvendelser. Den utforsker håndfølging ved bruk av MediaPipes hånddetektor og et dybdekamera for å styre robotbevegelser i visuelle servoingsoppgaver, omtrening av MediaPipes objektdetektor for nye objektkategorier innen produksjon, og språkmodell-drevet robotoppgaveplanlegging ved hjelp av GPT-modellfamilien. Oppgaven sammenligner ArUco-markørfølging med MediaPipe og dybdekamera-metoden, og de eksperimentelle resultatene tydeliggjør sistnevntes lovende egenskaper for håndfølging uten hansker. Resultatene understreker imidlertidig også behovet for videre utvikling for å håndtere dynamiske miljøer og situasjoner hvor operatøren holder verktøy mens hen har på seg arbeidshansker. Oppgaven foreslår videre forskning på å lage et tilpasset datasett for å trene en håndgjenkjenningsalgoritme basert på maskinlæring skreddersydd for arbeidssituasjoner, inkludert scenarier hvor operatøren bruker arbeidshansker. Videre avdekket omtreningen av MediaPipes objektdetektor utfordringer som bildenedskalering og modellens skjevhet mot kontekstuell informasjon mellom objekter. Til tross for disse utfordringene presenterer masteroppgaven en trinnvis guide for omtrening og fant at det kun kreves 20 arbeidstimer for å legge til én objekttype i modellen, inkludert tiden som trengs for å lage datasettet. Til slutt fant oppgaven at integrering av store språkmodeller med objektdeteksjon viser potensial for fleksibel oppgaveplanlegging basert på enkle språkkommandoer fra menneskelige operatører. Denne integreringen lar roboten være oppmerksom på sine omgivelser gjennom objektdeteksjon, og forankrer oppgaveplanleggingen i den virkelige verden. Imidlertidig påvirker variasjon i responsene fra språkmodellen systemets pålitelighet. Funnene i masteroppgaven demonstrerer muligheter og de potensielle anvendelsene av maskinlæring i produksjon og samarbeidende roboter.

Preface

This master's thesis is submitted per the requirements for the course code TTK4900 at the Norwegian University of Science and Technology (NTNU). It was conducted under the supervision of Prof. Jan Tommy Gravdahl at the Department of Engineering Cybernetics, NTNU, and researcher Mathias Hauan Arbo at SINTEF Manufacturing.

The master thesis is a continuation of a six-week internship I conducted with SINTEF Manufacturing during the summer of 2023, as well as the specialization project submitted in December 2023 per the requirements for the course TTK4550 (Hagestuen, 2023*a*). Parts of the work from the specialization project are included in this master thesis, either directly by incorporating nearly identical sections from the specialization project report or indirectly by rewriting sections or building upon results from the specialization project.

The reader should understand "nearly identical sections" as sections with the same structure but where sentences are rephrased or minor improvements or additions have been made. The following list provides an overview of the *nearly identical* sections incorporated directly from the specialization project report:

- Literature study on a human-controlled visual robot servo - Section 2.2.
- Theory about computer vision and Kalman filters - Section 3.1 and Section 3.2.
- Scope of hand-tracking methods for visual servoing subject to experiments in the thesis - Section 4.1.
- Description of hardware components and some of the software components in the experimental setup - Section 5.1 and from Section 5.2.1 to Section 5.2.4, plus Section 5.2.5.1.
- Camera calibration procedure - Section 5.3.

Next, the following list provides an overview of the sections that are *derivatives*, meaning they either build upon results or are structurally rewritten work from the specialization project:

- Description of the methodology employed in the literature study - Section 2.1.
- Experiment 1 - visual servo with ArUco marker tracking. The practical implementation of the visual servo system on the robot was conducted by myself during the summer internship with SINTEF Manufacturing - Section 6.1.1.
- Experiment 2 - hand detection in typical workspace situations using MediaPipe - Section 6.1.2.

Disclaimers are also provided throughout the report for the reader's convenience, marking the work nearly identical to or derivatives of parts from the specialization project. Generally speaking, the re-used work from the specialization project is present mainly in chapter 2 (literature review), chapter 3 (theory), and chapter 5 (experimental setup).

Throughout the thesis, I was granted access to SINTEF Manufacturing's robot laboratory in Trondheim, providing me with the hardware required to carry out the experiments. I want to

thank researchers Andrej Cibicik and Mathias Hauan Arbo at SINTEF Manufacturing for seamless communication when coordinating my many visits to the laboratory. Additionally, I direct a special thanks to Andrej Cibicik for 3D-printing the camera bracket and ArUco marker surface used in the experiments.

I thank my supervisors, Prof. Jan Tommy Gravdahl and researcher Mathias Hauan Arbo, for their invaluable contributions to both the technical aspects of my work and the structure of this thesis. Through our engaging discussions, profound insights have emerged, paving the way for deeper understanding and meaningful discoveries.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Problem description and motivation

In product manufacturing, there are numerous challenges regarding efficient human-machine collaboration. One of these challenges involves the necessity of physical interaction between robots and humans or the use of a teach pendant to control robots. Another challenge is flexible robot task planning in dynamic environments. The current reliance on code for programming new tasks makes robotic systems less accessible to non-technical workers. This thesis aims to explore the feasibility of using machine learning methods for human-robot collaboration in manufacturing to improve these challenges. In particular, the research discusses hand-tracking methods for guiding the robot movement and explores their feasibility by conducting experiments. The work involves tracking the operator's hand relative to the robot and steering the robot's motion based on the operator's hand movements. Applications for this work include robot arms assisting with handling bricks or other materials, handing objects to operators, and providing supportive lift-by-guiding tasks to improve work ergonomics. Next, the research explores leveraging the extensive knowledge within large language models (LLMs) to enable flexible robotic task planning based on simple natural language prompts from the operator. Specifically, LLMs integrate with an object detection model to ground the task planning in the real world by allowing the robot to be aware of its surroundings. To adapt the system to object categories in manufacturing, the thesis explores the possibility of customizing an existing open-source object detection model for new categories without needing to design a model from scratch. Applications for the prompt-to-task system include supporting assembly workers in a collaborative environment and providing a flexible tool-selection system for computer numerical control (CNC) machines, which interprets the operator's desired task from natural language input.

1.2 Contributions

The contributions of this thesis are:

- **Device-less hand tracking**: combined MediaPipe's hand detection solution with a depth camera to allow device-less hand tracking, demonstrated through a visual servo system.
- **A step-by-step guide for re-training MediaPipe's object detector for new categories**: developed a step-by-step guide for re-training MediaPipe's object detection solution for new object categories using MediaPipe Model Maker. The work showed that just 20 work hours are required per new object category added to the model, including the time needed to create the dataset.
- **Demonstrated real-world-aware task planning**: integrated large language models with object detection to enable real-world-aware task planning for robots.

- **Presented existing literature on visual servo systems and prompt-to-tasks mapping**: presented the current state of the art within human-controlled visual servo systems and natural language prompt mapping to robotic tasks in a comprehensive literature review.

1.3 Contribution to United Nations Sustainability Goals

The contributions of this master thesis toward the United Nations (UN) sustainability goals are addressed briefly. However, please note that the content of this publication has not been approved by the United Nations and does not reflect the views of the United Nations or its officials or Member States.

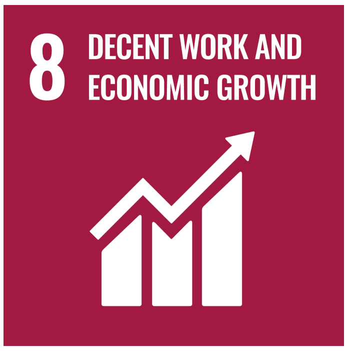


Figure 1.1: United Nation's sustainability goal 8: decent work and economic growth.
Image source: United Nations.

Figure 1.2: United Nation's sustainability goal 9: Industry, innovation and infrastructure.
Image source: United Nations.

Figure 1.3: United Nation's sustainability goal 12: Responsible consumption and production.
Image source: United Nations.

Goal 8: Decent Work and Economic Growth
This thesis promotes efficient human-robot collaboration in manufacturing, which can enhance productivity, create new job opportunities, and drive economic growth through technology development.

Goal 9: Industry, Innovation, and Infrastructure
The thesis supports industrial innovation and technological advancement by integrating machine-learning techniques with robotic systems.

Goal 12: Responsible Consumption and Production
The research encourages sustainable production processes by improving the efficiency of manufacturing tasks, which can reduce waste and optimize resource use, contributing to more responsible production practices.

1.4 Outline

First, a literature review on machine learning techniques for human-robot collaboration in manufacturing is presented in chapter 2 to demonstrate established techniques and methods. Next, chapter 3 presents relevant theory for introducing candidate tracking-, task planning- and object detection methods in chapter 4 and implementing an experimental setup in chapter 5, which also includes camera calibration procedures. Furthermore, chapter 6 documents the experiments conducted in the project, including hand-tracking for visual servoing and mapping natural language prompts to robotic tasks using environmental information gathered through object detection, and discusses experimental results and exciting findings. Lastly, chapter 7 concludes if the discussed methods are feasible for enhancing human-robot collaboration in manufacturing, also suggesting future work on the topic.

Chapter 2

Literature study

2.1 Methodology

Disclaimer: this section (Section 2.1) is a derivative of work from the specialization project (Hagestuen, 2023*a*).

The following literature study explores the current state of the art in human-robot collaboration, mainly focusing on (1) human-controlled visual robot servo and (2) mapping natural language prompts to robotic tasks. This review aims to provide a structured overview of the techniques and methodologies employed in these fields, highlighting the advances and challenges that researchers have encountered.

The academic database *Scopus* facilitates keyword-based search for literature, where operators such as AND, OR, and NOT can be applied to the keywords to narrow down and tune the search. Further, a range can be specified for the publishing year, and additional field codes such as author, conferences, funding, and more can be added to elevate the narrowing of the search.

Originally conceived as a structured process utilizing the academic database Scopus, the realities of the search process required the strategy to be adapted. The initial plan was to conduct two separate searches, one for each research topic mentioned above, i.e., (1) human-controlled visual robot servo and (2) mapping natural language prompts to robotic tasks. Each of the searches would encompass all relevant aspects of the topic. For instance, when researching human-controlled visual robot servo, the aim was to conduct a search encompassing both the hand tracking and visual servoing elements of the problem and then fine-tune this search until there was a manageable number of articles at hand. This would allow for easy repeatability for later research on the two topics. However, this approach proved challenging, yielding limited results, with many being irrelevant to the problem at hand, especially in the context of (1) human-controlled visual robot servo. Despite this, the unified searches for literature that includes all aspects within the two respective topics (1) and (2) are summarized in Appendix A, allowing them to be repeated at a later stage. Note that several articles (Cuevas-Velasquez et al., 2019; Gong et al., 2018; Muthusamy et al., 2021; Wang et al., 2019; Xiao and Chen, 2019; Liu and Zhang, 2019) discovered through the unified searches have been addressed and included in the literature review, even though the search-strategy was modified.

As such, the decision was made to separate the searches into sub-topics. Using (1) human-controlled visual robot servo as an example, the search was split into two sub-topics: hand tracking and visual servoing. This approach allowed for a more comprehensive understanding of the individual components of the topics but also presented the possible drawback that the produced articles are not necessarily directly connected to a use case similar to that of the human-controlled visual robot servo.

Additionally, some valuable articles emerged organically through citations and references in already-found articles and discussions with supervisors Prof. J. T. Gravdahl and researcher M. H. Arbo.

These findings, made outside the framework of structured search queries in Scopus, were incorporated into the literature review.

To improve the structure of the review, it is divided into subsections based on topics. Articles addressing similar topics are discussed in the same subsection. Lastly, all articles and papers utilized in the review are summarized in a table, highlighting their main contributions to human-robot collaboration.

2.2 Human-controlled visual robot servo

Disclaimer: this section (Section 2.2) is a nearly identical version of work from the specialization project (Hagestuen, 2023*a*).

The first part of this literature review is devoted to research on human-controlled visual robot servo.

2.2.1 Visual servoing - early history

The concept of utilizing machine vision to close the feedback loop of a position controller for a robot end-effector is referred to as *visual servoing*. Hutchinson et al. (1996) claim that this term was first introduced by Hill and Park (1979) to distinguish this approach from the decoupled, open-loop look-and-move systems, where the vision system is not employed while the robot is moving. According to Hutchinson et al. (1996), the less specific term *visual feedback* was used before the introduction of the term *visual servoing*. However, Shirai (1971) demonstrates a closed-loop visual servo utilizing a commercial vidicon TV camera, implementing a system for placing a square prism block into a square block. This article dates back to as early as March 1971 and relies on an algorithm for extracting line drawings of the object based on light-dark boundary points in the image. An initial, rough recognition of the box is done, then the block is carried above the box before the more precise cycling of the visual feedback is deployed. The scheme only uses a single camera, thus requiring a constraint on the picture data to extract 3-dimensional coordinates, a technique also leveraged by Wu and Cheng (2018). As such, the system imposes a constraint on the vertical position of the block and the height of the plane where the box is placed. Thus, the position and attitude of the box can be calculated. The system completes the entire task in about 80 seconds, mainly due to the long processing times of image data, even with just 3-4 cycles of the feedback loop necessary to decrease the error to an accepted limit after the rough recognition of the box. For illustration, a processing time of about 10 seconds is needed for each cycle of the visual feedback loop, demonstrating the importance to modern robotics of the impressive progress within computational power.

2.2.2 Replicating human-like behavior in the manipulator

Jiang et al. (2022) demonstrate a visual servo controller on a 7 degrees of freedom (DOF) robot, leveraging motions within the manipulator's null space to achieve human-like behavior in the robot arm. This is done by replicating the swivel angle of the operator's arm derived from the image data. The swivel angle is defined as the angle between the plane containing the whole arm and the vertical plane passing through the shoulder and the wrist, see Figure 2.1. Replicating the swivel angle reduces psychological distress for the operator and nearby humans by making the robot more human-like. As stated in the article, it is not enough to enforce a human-like behavior of the end-effector, as the arm pose can also contribute to an improved human-like behavior. An enhanced robot interaction experience has the potential to contribute to a more positive perception of collaborative robots (cobots) within the industry.

The authors investigate a torque-level controller to actuate the robot arm to the desired position. Since $M(q)\dot{\alpha}+C(q,\dot{q})\alpha+G(q)$ is unknown, a sliding mode torque controller is suggested to overcome this problem. Here, α represents the desired velocity vector, $M(q)$ is the inertia matrix of the

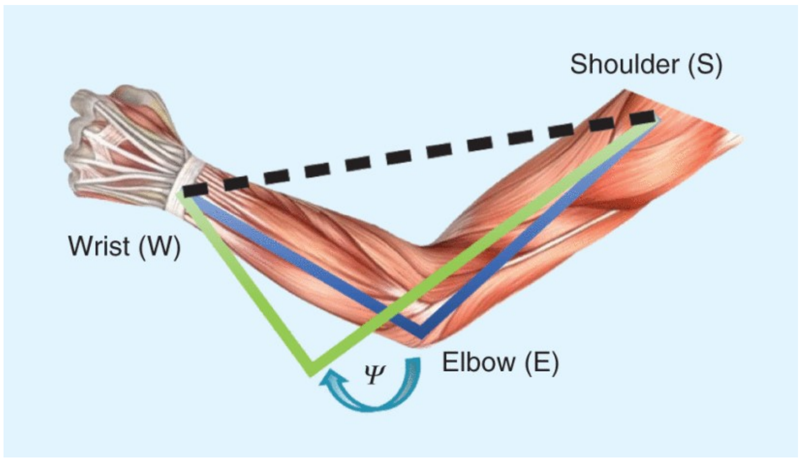


Figure 2.1: Illustration of the swivel angle, defined as the angle between the plane containing the whole arm and the vertical plane passing through the shoulder and the wrist.
Source: Jiang et al. (2022).

manipulator, $C(q, \dot{q})$ represents the centripetal and Coriolis terms, and $G(q)$ is the gravitational matrix. Together, they make up the dynamical model of the manipulator. The sliding surface is defined as $z = \dot{q} - \alpha$, q being the joint positions. By driving the system's state to this surface, the advantages of sliding mode control are exploited, as this method offers robustness to uncertainties and disturbances.

To enhance the performance of the visual servoing control, a barrier Lyapunov function (BLF) is used to establish a function constraint. This constraint's primary purpose is to guarantee that the image features remain within the field of view (FoV) by ensuring that the control variables stay within the bounds of the suggested BLF.

2.2.3 Utilizing reinforcement learning for improved controller performance

Li and Zhang (2023) suggest a reinforcement learning (RL) scheme to increase the performance of the controller, specifically trying to solve two issues related to non-effective control of the velocity of the robot arm; firstly, the object may fall out of the FOV due to too-fast motion, and secondly, too-slow motion will cause a slow convergence to the target position. In particular, this paper proposes an RL scheme where the agent chooses the servo-gain λ adaptively at each time step to overcome and improve on these issues.

A limitation of the proposed scheme is that it depends on a calibration plate for extracting image features used in the feedback loop. Note that this does not entirely discredit that the RL scheme outperforms the static-gain servo controller, as both controllers are fed the same image features. On the contrary, the method selected for learning the policy π mapping the state space $\mathbf{S}$ to the action space $\mathbf{A}$ in the RL scheme will suffer if the scheme is subject to extensive loss of image feature points. This is because extra penalties are imposed through the reward function whenever feature points are missing. Specifically, the scheme by Li and Zhang (2023) adopts the Q-learning method, where the strategy is to build a Q-table with a score associated with each state-action pair. A higher score implies that the given action is more beneficial to the goal state. Due to the chosen reward function, this score is reduced whenever there are missing feature points in a given iteration. Consequently, the convergence of the Q-table will likely be negatively impacted if a less reliable object than a calibration plate is used for detection.

2.2.4 Using image moments to represent objects

Guo-Dong et al. (2010) utilize image moments, a kind of global image feature, claiming that they provide a generic representation of any object. This is opposed to using more straightforward geometric image features like circles, lines, and dots. The implemented scheme relies on the research by Chaumette (2004) and Tahri and Chaumette (2005) on deducing the analytical form of the interaction matrix based on image moments. This could present a solution to the issue in Li and Zhang (2023) of using a calibration plate to extract image features. The paper also proposes a nonlinear feedback controller, proving an inverse relation between the steady state error and the feedback gain's exponent. This is an improvement compared to the linear case, where the inverse relation is simply between the steady state error and the feedback gain itself.

2.2.5 Technologies in augmented and virtual reality

The task of tracking the hand of the operator and estimating its 3D position, in many ways, overlaps with technologies within virtual reality (VR) and augmented reality (AR), where hand tracking is a key enabling technology. Fang et al. (2023) provide a systematic review of how augmented reality is used in manufacturing, specifically concerning head-mounted devices (HMDs). The article's main contribution is a map of currently deployed AR HMDs in the manufacturing industry and related challenges, enabling researchers to identify technical perspectives that require attention and research. As indicated by the authors, the number of AR HMD systems deployed in the manufacturing industry has increased over the previous years. However, they still suffer from limitations that prevent them from being widely used on the shop floor. One of the critical challenges is the unpredictable environment the manufacturing industry represents, with elements such as changing light conditions and texture-less scenes making it difficult for the AR system to accurately track or anchor elements onto them because there are fewer distinct features or textures to use as reference points. Eventually, this may lead to the system misunderstanding instructions, increasing assembly time and causing errors. It is worth noting that the general idea of hand tracking in manufacturing settings does not necessarily suffer from the same limitations as an AR HMD system. For example, the issue of having a texture-less scene could be solved if the operator wears gloves with a specific color. On the flip side, widespread industrial use of hand tracking for visual servoing in manufacturing might be limited by challenges with related systems, which cause a general slowdown of deployment of vision and tracking systems in the manufacturing market.

2.2.6 Gesture recognition by fingertip tracking

Cao and Wang (2019) propose a scheme where the fingertips of the operator are tracked using a pan-tilt-zoom (PTZ) camera after a "right click down" gesture is detected. A silhouette of the hand is created by skin color segmentation. Next, a curve is fitted to four control points using cubic Bezier curves. The optimal control points are generated through the Monte Carlo method, which approximates solutions to mathematical problems through statistical sampling. The paper also concludes that this technique can be extended to other areas where curvatures of objects, and possibly also hands, need to be detected.

2.2.7 Device-based vs. camera-based tracking

In general, hand recognition can be divided into two subcategories: firstly, actual hand tracking, and secondly, recognizing hand gestures. The former, the most relevant for human-controlled visual servo, can be divided into device- and camera-based tracking. Device-based tracking requires wearing a wearable, like gloves or a hand-held motion device. Wang et al. (2019) present a novel and practical approach that utilizes a wearable sensory system to recognize human hand-over intentions. The authors address that the main focus of existing research has often been on robot motion planning and control during hand-over, assuming knowledge of human hand-over intentions. By reducing the intentions to *Need, Give, Stop, Continue, Speed-up, Slow-down*, and sensing signals from the human's forearm postures and muscle activities representing the control intentions, the robot can perform the actions requested by the human. One potential limitation of this approach is its reliance on the wearable sensory system.

This type of tracking is the classical approach and might offer improved accuracy. Still, it is unnatural for the user due to possible restrictions the wearable imposes, for example, during interactions with objects. This issue is emphasized by Wu et al. (2019), who propose a camera-based setup for body-tracking where data from several cameras is fused to a skeleton while comparing different camera-weighting methods. Camera-weighting can be understood as how the ratio of data coming from each device should be adjusted to achieve better tracking. The hand's position can be obtained from the produced skeleton after fusing data from the devices using the chosen camera weighting. The physical setup is illustrated in Figure 2.2.

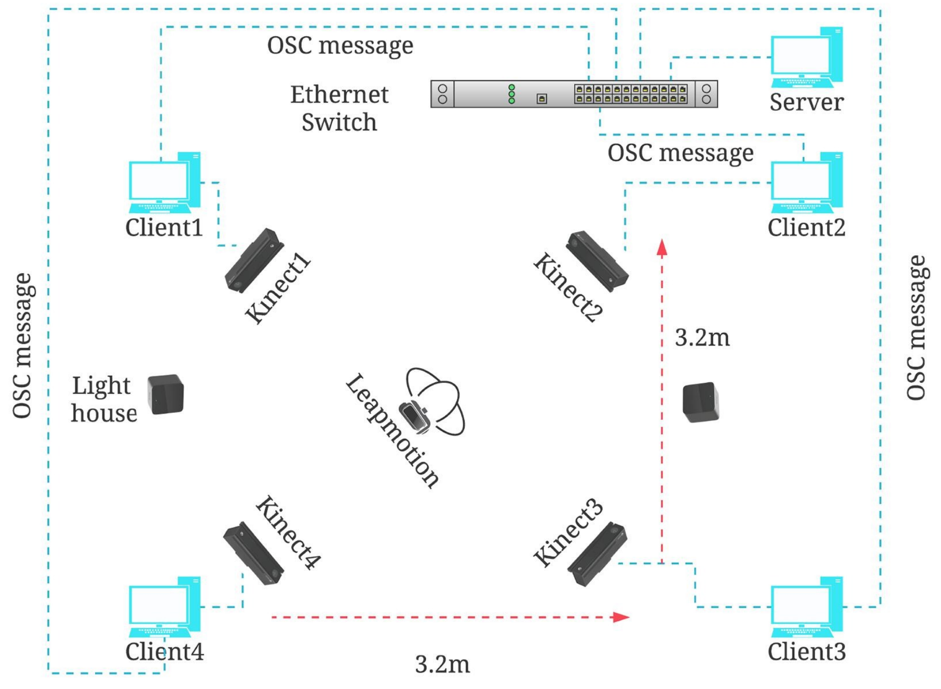


Figure 2.2: A camera-based setup for body-tracking where data from several cameras are fused. Data from the different cameras are weighted according to the situation for improved tracking. An OSC (Open Sound Control) message is a protocol for communication between computers, sound synthesizers, and other multimedia devices. The Kinect devices are the cameras used in the setup, and the Leapmotion device enhances the accuracy and stability of hand gesture recognition through hand and finger tracking.
Image source: Wu et al. (2019).

Although this work demonstrates hand tracking without the use of a wearable, one can argue that the scheme suffers from relatively high demand on space, in addition to substantial installation effort. The space requirement of the setup is in strong contrast to cobots, which are marketed as space-efficient, easy-to-install machines. This contradiction limits the possible use cases of a human-controlled visual servo employed on a cobot had such a camera setup been used. Secondly, the cameras require precise extrinsic calibration to fuse the data in a common coordinate frame. This limits the usage of such a system in a visual servo to static shop floors and similar environments, as it is hard to maintain the calibration on, e.g., a construction site, where the scene might be dynamic, and the system needs to be moved more often.

The approach presented by Cuevas-Velasquez et al. (2019) also suffers from the abovementioned issues. This paper demonstrates a hybrid system utilizing four red-green-blue-depth (RGBD) sensors as eye-to-hand (EtoH) visual input and a stereo camera as the eye-in-hand (EinH) visual input. A master supervisor selects the visual input based on the distance between the robot and the object. This exploits the advantages of the EtoH and EinH configurations simultaneously; EtoH offers reduced occlusion and supports the EinH should the target move out of its field of view, while EinH is used for fine alignment when performing tasks. The scheme is demonstrated by performing four different tasks, one of which is delivering an item to a moving hand. A pink glove was employed to segment the target based on color in EtoH, while a blue palm circle was utilized to enhance localization in EinH. Experiments involving the hand moving to 12 distinct locations repeatedly demonstrated a 75% success rate in delivering the item to the palm when operating in hybrid mode, as opposed to a 58% success rate in the EtoH-only mode.

Gong et al. (2018) address that visual servo methods usually require extrinsic/intrinsic calibration and that the hand-eye relationship is unknown. As such, the paper proposes a visual servo method for use without prior knowledge of camera parameters and hand-eye relationship, featuring a task function rooted in projective homography. By exploiting this technique, corresponding points in different images can be related, even though the views might be distorted.

2.2.8 Hand tracking by machine learning

For solving the task of recognizing the hand of the operator and estimating its 3D pose, looking at research within machine learning is relevant, as it can be an enabling technique for the camera-based tracking system. In particular, convolutional neural networks (CNNs) have proven well-suited for computer vision and image processing tasks, including hand pose estimation. Recent works have exploited this by applying 2D CNNs, achieving good performance (Tompson et al., 2014; Oberweger et al., 2015; Ge et al., 2018*b*). However, achieving precise and reliable hand pose estimation remains challenging due to significant variations in hand positions and orientations, the complex nature of hand movements, different parts of the hand blocking each other, and the similarities between fingers in the image. Some of these issues are related to the fact that some information is lost in the projection from 3D to 2D, even if more views are utilized. Also note that an increased number of views requires more computational performance, possibly affecting the system's frame rate negatively.

Therefore, Ge et al. (2018*a*) argue that 3D CNNs are more suitable due to the added spatial information. This claim is supported by Yuan et al. (2018). Ge et al. (2018*a*) present a CNN-based method producing the full 3D hand pose from a 3D volumetric representation of the depth image of the hand. This is possible due to short-range depth cameras like the Intel RealSense series. The authors address the issue of facing variations in hand sizes by performing data augmentation; in addition to training the model on a suitable training set based on real data, slight changes are made to the existing data to simulate different situations, for example, small rotations.

2.2.9 Kalman Filters for tracking purposes

Tracking systems are often combined with a Kalman filter for better pose estimation. Additionally, a Kalman filter can generate velocity estimates. Xiao and Chen (2019) leverage partial knowledge of the target's dynamics to improve on challenges caused by low sampling rates and delay in the visual feedback loop. When a partial understanding of the target's dynamics is available, multi-rate sampling can be utilized to recover fast signals from slowly sampled ones. By exploring different dynamics like constant acceleration and constant velocity models, this paper demonstrates tracking a target with fast dynamics even when the sampling is slow, and the measurements are not timely.

2.2.10 Neuromorphic cameras for reduced latency

A different approach to the low sampling rate problem is presented by Muthusamy et al. (2021), who introduce an event-based neuromorphic camera, i.e., a camera only recording changes in the scene. Each pixel in a neuromorphic camera is independent and reacts to changes in light intensity, producing an event (a data point) when a change occurs. This is in contrast to traditional cameras that capture a full data frame at a fixed rate, regardless of whether there is a change. The event-based nature of a neuromorphic camera allows for low latency to visual changes. Additionally, this paper demonstrates the superior performance of the event-based camera in poor lighting conditions compared to frame-based vision.

2.3 Mapping natural language prompts to robotic tasks

The second part of this literature review is devoted to research on how natural language prompts can be used to interact with robots by mapping the prompt to robotic tasks.

2.3.1 Exploiting LLMs for adaptive robotic task sequencing

When researching the area of human-robot collaboration, a topic of interest is how natural human language can be mapped into, or *grounded* in robotic tasks. Large language models (LLMs) contain vast knowledge that can be exploited to possibly achieve such a mapping. However, as addressed by Ahn et al. (2022), LLMs lack real-world experience, particularly concerning the current environment of the robot. For instance, if the language prompt is ”I need to boil these eggs, can you help?”, the LLM might give reasonable instructions on how to do so, but they might not apply to the robot in question. As such, the paper suggests constraining the mapping to actions that are not only possible for the robot to execute but also appropriate given the current state of the robot and its environment. The latter is achieved by introducing *value functions*; for a particular skill, the value function returns the probability of successful completion c_π given the current state of the robot. The value functions are learned using RL, and the paper finds that grounding the LLM in the real world using these functions doubles the performance of the system. For instance, two of the robot skills known to the LLM might be *move to the kitchen counter* and *pick up an apple*. If the text prompt is ”I'm hungry, can you help me find a fruit?”, the generated sequence of actions will differ based on whether an apple is visible to the robot. In particular, if an apple is not visible to the robot, then the likelihood of success for *pick up an apple* is lower, and it will be given a lower score. However, if *move to kitchen counter* is executed first, an apple might become visible, and the score will be adjusted.

Once a skill has been selected, the robot will execute it. Additionally, the following query to the LLM will be adjusted to include the executed skill. The process is repeated until a termination criterion is met. This process is described in algorithm 1.

Algorithm 1: SayCan algorithm. Source: Ahn et al. (2022).

```
Data: A high-level instruction $i$, state $s_0$, and a set of skills $\Pi$ and their language
      descriptions $\mathcal{L}_\Pi$
Result: Execution of the SayCan algorithm
$n = 0$, $\pi = \emptyset$;
while $\mathcal{L}_{\pi_{n-1}} \neq$ "done" do
    $C = \emptyset$;
    for $\pi \in \Pi$ and $\mathcal{L}_\pi \in \mathcal{L}_\Pi$ do
        $p_\pi^{LLM} = p(\mathcal{L}_\pi | i, \mathcal{L}_{\pi_{n-1}}, ..., \mathcal{L}_{\pi_0})$ ;        // Evaluate scoring of LLM
        $p_{affordance}^\pi = p(c_\pi | s_n, \mathcal{L}_\pi)$ ;        // Evaluate affordance function
        $p_{combined}^\pi = p_{affordance}^\pi p_{LLM}^\pi$;
        $C = C \cup p_{combined}^\pi$;
    end
    $\pi_n = \arg\max_{\pi \in \Pi} C$;
    Execute $\pi_n(s_n)$ in the environment, updating state $s_{n+1}$;
    $n = n + 1$;
end
```

Further research on this work was done by Huang et al. (2022) to include feedback. In particular, the system now incorporates three types of feedback to enable what is referred to as an *inner* *monologue*. Firstly, *success detection* is incorporated as a binary classifier of whether the skill π_k was successful or not. Secondly, *passive scene description* covers all sources of feedback originating from information automatically gathered about the scene without any query by the LLM planner. An example of passive scene description is object recognition. Thirdly, *active scene description* includes queries from the LLM planner, which a human can answer. The example illustrated in Figure 2.3 highlights these three types of feedback.

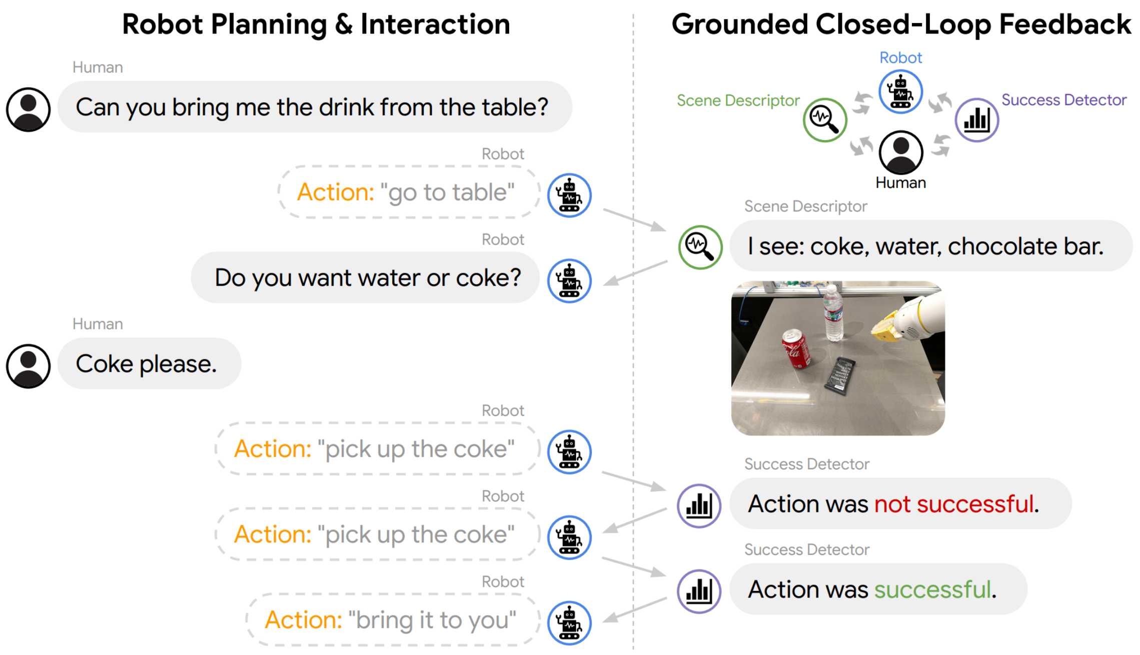

Robot Planning & Interaction
Grounded Closed-Loop Feedback
Human
Can you bring me the drink from the table?
Robot
Action: "go to table"
Robot
Do you want water or coke?
Human
Coke please.
Robot
Action: "pick up the coke"
Robot
Action: "pick up the coke"
Robot
Action: "bring it to you"
Robot
Scene Descriptor
Success Detector
Human
Scene Descriptor
I see: coke, water, chocolate bar.
Success Detector
Action was not successful.
Success Detector
Action was successful.

Figure 2.3: Example of *inner monologue*, enabling robot planning with LLMs coupled with pre-trained robot skills. Feedback is implemented by leveraging scene descriptors and success detectors. Source: Huang et al. (2022).

After the robot has reached the table, object recognition (passive scene description) generates two drink options for the user. The LLM planner then asks the human (active scene description) if water or coke is preferred before trying to pick up the coke from the table, which is the human's preferred drink. Picking up the coke fails on the first attempt (success detector), causing the LLM planner to initiate the action again. This time, the action succeeds, and the robot brings the coke to the human. By introducing feedback, Huang et al. (2022) achieves significantly improved performance, especially when injecting noise into the policy actions.

2.3.2 Design procedure for robot programming using LLMs

A key challenge within human-robot collaboration is that current robotic systems require a specialized engineer *in the loop* to translate a requirement into code that can be deployed onto the robot. As highlighted by Vemprala et al. (2023), this process suffers from three main issues: firstly, it is time-consuming, requiring the user to write new code whenever the requirements are adjusted. Secondly, it is expensive for the customer due to the deep knowledge of robotics needed for the worker to perform the job. Lastly, it is inefficient, as it may require multiple iterations of testing and improvement to ensure proper functionality. As such, the authors introduce a set of design principles, exploiting the knowledge contained by an LLM like ChatGPT to fundamentally change the way a requirement (or task) is translated into code:

1. **Define a set of high-level functions** relevant to the task at hand. The functions must have intuitive names so the LLM can process them correctly. This library, or API, is robot-independent but should map to existing low-level controls from the robot-specific control stack. Examples of functions could be *move_to_position(X, Y, Z)* or *pick_up_object(obj_name)*. Note that this is the most knowledge-demanding step in the procedure.

2. **Write a text prompt for the LLM using engineering principles**. In addition to describing the requirement, the prompt must specify which functions are available from the high-level API, coupled with brief code instructions, like how the input and output of the functions are formatted. Additional information about task constraints can also be provided.

3. **The user stays *on the loop*** by evaluating the code output from the LLM and suggests improvements using natural language.

4. Once the user is satisfied, the code can be **deployed onto the robot**.

An essential part of the design procedure is incorporating feedback, in particular by allowing the user to stay on the loop by evaluating and providing feedback on output code from the LLM. According to the authors, this is one of the key improvements of this scheme compared to existing approaches. Several tasks are demonstrated throughout the paper, such as inspecting a shelf in a lawn mower pattern and taking a selfie in a mirror using an aerial drone, or catching a baseball using a planar robot.

2.3.3 Re-training an object detection model for arbitrary objects

An essential aspect of a scheme implementing a mapping from language prompts to robotic tasks is awareness of the robot's surroundings. This enables the mapping to be restricted to tasks appropriate to the situation for an increased probability of success, essentially rooting the mapping in the real world. As mentioned, object detection can play a crucial role when gathering information about the surroundings. For improved adaptability of such systems, it is worth looking into how existing models for object detection can be re-trained on a new dataset to enable the detection of new objects.

Microsoft facilitates this through the MediaPipe Model Maker module, which enables customizing models such as the object detection solution. This module allows the user to replace only the classification layers of the model, meaning that parts of the existing model are preserved; see Figure 2.4 for an illustration. When replacing the classification layers with the rebuilt layers tailored for the application at hand, the model generally gets smaller because it is now limited to detecting the objects it was re-trained for. Since a significant portion of the existing model is preserved, training requires less time than training a new model because less data is needed. The documentation suggests creating a dataset of 100 images per new object to be detected. For instance, if the re-trained model needs to detect triangles, squares, and circles, the guideline proposes a dataset of 300 images (Google, 2023*b*). After re-training the model for new, selected object categories, it is exported as a TensorFlow Lite model and can be employed in the desired application.

A search was conducted in the academic database Scopus to find papers and studies that utilize the MediaPipe Model Maker. The goal was to highlight the experiences of other researchers with this tool, particularly to assess its feasibility for customizing the object detector for identifying objects in new scenarios, such as in workplace environments. The search outcomes were relatively sparse, yielding only a small number of articles. Unfortunately, most of these articles were devoted to related topics rather than the MediaPipe Model Maker itself. However, one of the articles (Uboweja et al., 2023), written by Google employees, utilizes MediaPipe Model Maker for customizing one of the MediaPipe machine learning models. Unfortunately, this paper customizes the hand gesture recognition solution and not the object detection solution.

Table 2.1 documents the search strings used in Scopus, the number of articles produced, and the search date. Note that this table must not be confused with the table mentioned in the literature review methodology (Section 2.1), which aims for a much broader reach and covers the baseline searches for this literature study. In contrast, the searches made for research on MediaPipe Model Maker are more specific. Yet, they are still documented in this review for easy repeatability because of the low amount of articles produced, making it an exciting area of research. MediaPipe Model Maker is a relatively new tool introduced in May 2023 (Ruiz and Tibthat, 2023), possibly explaining why it has not been subject to more research.

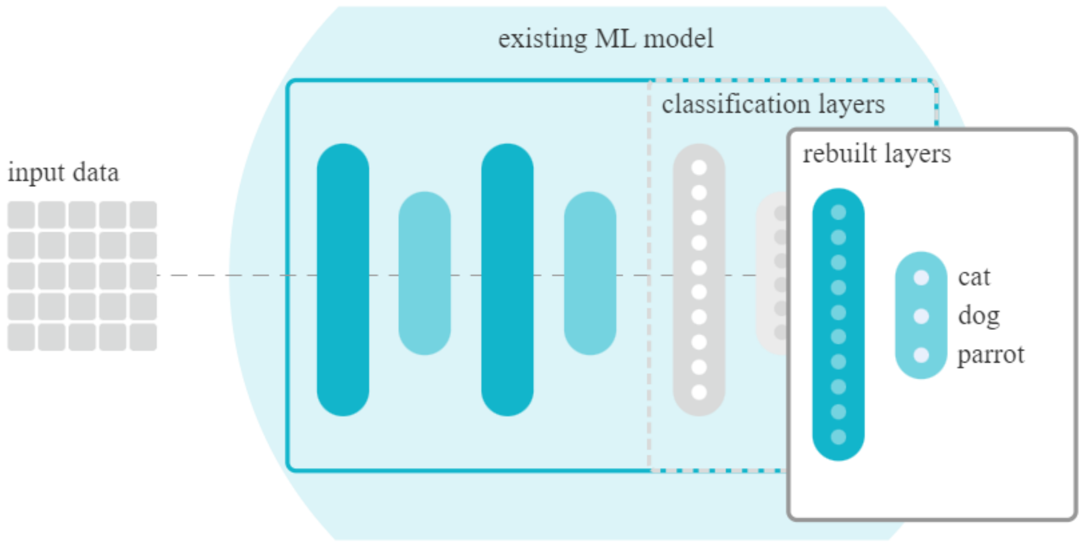


Figure 2.4: The MediaPipe Model Maker module rebuilds the classification layers of the model using the new dataset.
Source: Google (2023*c*), license: Creative Commons Attribution 4.0.

Search string	**Number of articles**	**Date of search**
mediapipe AND model AND maker	5	February 7th, 2024
ABS(mediapipe AND model AND maker)	1	February 7th, 2024

Table 2.1: Scopus searches aiming to find literature and research on the MediaPipe Model Maker solution, released in May 2023. The "ABS" keyword constrains the search to only the article's abstract.

2.3.4 Common Objects in Context - COCO dataset

Creating the dataset used for training, validation, and testing is the most work-intensive part of re-training a machine learning model. COCO 2014 (Lin et al., 2014) is an example of such a dataset, containing 328.000 images with 91 different object types. Parts of this dataset were used to train the default MediaPipe object detector, i.e., the existing model, which can be customized for new object categories.

The creators focus on creating a dataset with images containing the desired objects in natural environments and from different viewpoints, as opposed to typical images of the objects at hand. For example, a picture of a white dog facing the camera, centered in the middle of the scene with a contrasting background (e.g., grassy) is deemed a typical image and would not be included in the dataset. A detailed rundown of the procedure is provided in the article; only a short version is reproduced here. First, images were collected from Flickr, a site where amateur photographers can upload pictures. Images were collected from this site using a selection strategy aiming to avoid iconic images of the objects; image searches included pairwise object-object combinations like "dog + car", pairwise scene-object-scene combinations like "chair + restaurant", and in some cases single-category searches with explicit filtering to remove iconic images. The authors claim that this strategy was effective for compiling a collection of non-iconic images.

Next, the selected images were put into a pool before being subject to an extensive annotation pipeline consisting of three steps: (a) category labeling, (b) instance spotting, and (c) instance segmentation; see Figure 2.5 for an illustration.

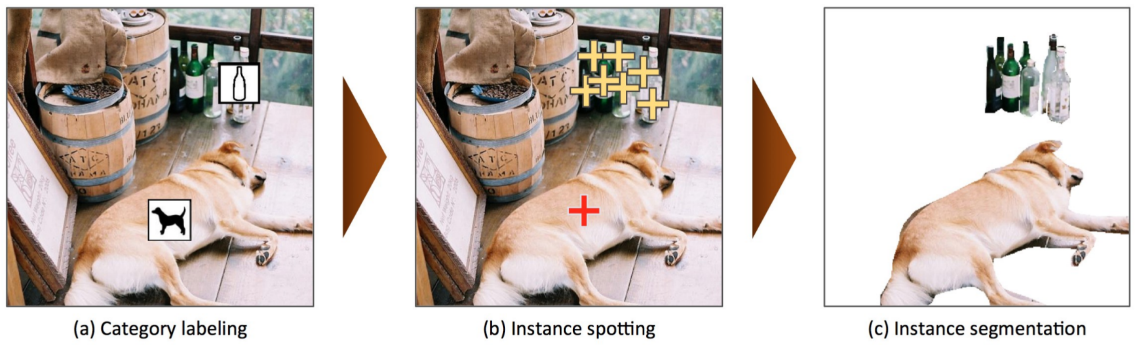


Figure 2.5: Annotation pipeline applied to all images in the pool. The pipeline consists of three primary steps: (a) identifying the object categories in the image, (b) locating and highlighting all occurrences of the identified categories, and (c) outlining the boundaries of each object.
Source: Lin et al. (2014), license number 5747111134972 granted through CCC RightsLink.

Consequently, creating this dataset required over 70.000 worker hours, sourced using Amazon Mechanical Turk, a marketplace for outsourcing tasks that computers cannot perform. Annotating new datasets for training a machine learning model is a typical example of such a task. One could argue that the number of worker hours could be reduced by sourcing images using search engines such as Google or Bing, allowing direct searches for images containing the desired objects. However, these searches would produce more typical images of the objects rather than showing them in a natural environment, which is an undesirable scenario. Additionally, this would likely reduce the number of instances per image, highlighted as another advantage of the procedure because it might help learn more contextual information.

While this article addresses the older COCO 2014 dataset, it should be noted that the newer COCO 2017 uses the same images as COCO 2014 but with a different split (Voxel51, n.d. *a*). Even though training object detection models using datasets like COCO 2014 can significantly reduce the workload required for employing such a system, a clear drawback is that the dataset might not support the desired object categories. For instance, using an object detection system in a manufacturing scenario might require support for specific tools and parts not part of the COCO dataset. As such, the discussion on re-training existing models for new object categories is highly relevant when aiming to tailor the model for specific applications.

As a final note on the COCO dataset, it is mentioned that FiftyOne is a valuable tool for downloading datasets and evaluating model predictions stored in the COCO format, whether they are pre-existing datasets like COCO 2014 or custom ones (Voxel51, n.d. *b*).

2.3.5 A review of language-facilitated human-robot cooperation

Liu and Zhang (2019) point to the fact that a lot of research has been conducted on natural-language-facilitated human-robot cooperation but that a thorough review revealing the latest methodologies is missing. The paper suggests intelligent manufacturing as one of the scenarios where this form of human-robot collaboration can be employed. One limitation of this paper is its publication date in 2019, which implies that it does not cover further research developments in the field up to the present day. An example of a later development is the widespread use of LLMs after ChatGPT's introduction in November 2022 (OpenAI, 2022), which can be used for understanding instructions by the operator.

Despite the potential drawback of being outdated, Liu and Zhang (2019) provides valuable insight on the topic of human-robot communication using natural language. As discussed in the paper, traditional human-robot communication methods, including physical indications like contact sensing and force measurements and visual techniques such as pose detection and motion tracking, suffer three apparent drawbacks. Firstly, to a varying degree, the communication method requires humans to be trained on specific actions/poses that the robot can understand. Second, the aforementioned communication methods require designing and implementing informative patterns. Examples of such patterns could be that ”rotate hand” means ”try again” or that ”nod head” means ”yes”. Thirdly, since natural language instructions can be delivered orally, the operator’s hands can be used for other purposes during the cooperation, which is not necessarily the case for traditional communication methods.

The paper outlines three aspects of natural-language-facilitated human-robot cooperation: language instruction understanding, language-based execution plan generation, and knowledge-world mapping. It presents a summary of open problems for each aspect, creating a foundation for future research.

Firstly, **language instruction understanding** is addressed, and two main approaches are suggested: utilizing a *literal* model or an *interpreted* model. A literal model extracts information from the user input by evaluating *literal linguistic features* such as words, word references/associations, and sentence structure. This differs from the interpreted model, which extracts information from *interpreted linguistic features* such as relations, object physical properties, and object functional roles. The literal model suffers from the challenge of summarizing all the potentially encountered grammar rules and incorrect associations when searching for word references. Meanwhile, the interpreted model is better at considering practical conditions but encounters issues when combining several modalities, such as speech and visual input. The interpreted model also suffers from the overfitting problem, where it may excessively tailor itself to the specifics of the training data, making it less adaptable to new, unseen scenarios (Liu and Zhang, 2019). This issue highlights one of the possible benefits of exploiting the potential of LLMs; due to the vast amount of training data, the model is trained to understand a wide range of contexts while also being aware of practical aspects such as physical object properties.

Secondly, consider the aspect of **planning robot executions** using information extracted from the language instruction. Several models are suggested to perform this task, including *probabilistic*, *logic*, and *cognitive* models. In essence, a typical *probabilistic* model associates instructions y from the human with an execution parameter x by evaluating a joint probability $p(x|y)$. For instance, a typical joint-probability can be an activity-object association such as ”read” $\rightarrow$ ”book”. Next, *logic* models teach robots rational reasoning by implementing logic formulas describing complex procedures such as tool usages, action sequences, and locations, and the robot execution planning is based on *not* violating these logical rules. Lastly, *cognitive* models are made by defining a set of soft rules, essentially defining logic formulas and corresponding weights. By removing hard constraints, the execution plan becomes flexible, allowing instructions to be re-ordered, omitted, or added during the procedure. The soft logic in the cognitive model is more similar to the human thought process because of its flexibility. However, this approach encounters challenges in the learning phase of the cognitive model (Liu and Zhang, 2019).

Thirdly, the paper addresses **knowledge-world mapping** as an aspect of language-facilitated human-robot cooperation. The robot needs to utilize the information gathered in the two previous steps in practical cooperation situations. The first method employed is *theoretical knowledge grounding*, which revolves around the mapping between learned knowledge about items into corresponding objects in the real world. This method shares similarities with *passive scene description* suggested by Huang et al. (2022). It is typically implemented using an RGB camera, searching for known properties such as colors or shapes in the scene. An essential challenge concerning this technique is that human instructions are often unclear, abstract, incomplete, or even inconsistent with the real world. The second method to establish a knowledge-world mapping is *gap-filling methods*, which detect and tries to fill knowledge gaps between the execution plan and the real-world scenario. This technique is employed in either direction: information not covered in the execution plan about something seen in the real world and information provided by the human with no correspondence in the real world. A key challenge with this method is that difficulties arise when a robot seeks additional information after detecting a gap when asking humans or checking databases. Problems include scalability issues and insufficient content for specific needs (Liu and Zhang, 2019).

As mentioned, the article used as a baseline for this discussion is from 2019, possibly serving as a weakness since it does not cover developments made after its publication. An observation is that the challenges mentioned thus far, such as understanding language nuances, diverse contexts, complex relationships, and dealing with new information, are areas where LLMs such as ChatGPT excel. As such, exploiting the vast knowledge contained in LLMs can offer solutions to challenges in language instruction understanding, planning robot executions, and knowledge-world mapping by providing a rich understanding of language, contexts, relationships, and general knowledge.

This literature review is based on the articles listed in Appendix B, which also highlights their main contributions.

Chapter 3

Theory

This chapter presents the theory necessary for implementing the experimental setup and conducting experiments in this master thesis, with a particular focus on computer vision, Kalman filters for tracking, and neural networks for object detection.

3.1 Computer vision

Disclaimer: this section (Section 3.1) is a nearly identical version of work from the specialization project (Hagestuen, 2023*a*).

The structure of the following theory on computer vision is inspired by Plazaola and Agustín (2016) and Haavardsholm (2023).

3.1.1 Camera Theory

The camera represents the sensor that closes the high-level control loop in the human-controlled visual robot servo scheme. The modern camera is complex, but every camera, whether analog or digital, can be reduced to two fundamental components, each responsible for fulfilling the essential tasks expected from an imaging device: firstly, capturing incoming light rays from the external environment and directing them in a particular direction, and secondly, rendering (and recording) those rays into an image on a surface, so that it can be manipulated at a later stage. This surface would in a digital camera be a matrix of light sensors, and a photosensitive film in an analog camera (Plazaola and Agustín, 2016).

In simple terms, the camera captures a 3D point $\mathbf{x}^c \in \mathbb{R}^3$ in the camera frame and projects it onto pixels $\mathbf{u} \in \Omega$ in the image plane. In other words, the camera model is a projection $\pi : \mathbb{R}^3 \rightarrow \Omega$ such that

$$\mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix} = \pi(\mathbf{x}^c) \tag{3.1}$$

Fully extracting information about the 3D point $\mathbf{x}^c$ from the pixel $\mathbf{u}$, i.e., performing the inverse operation, requires additional information about the depth z. Achieving sufficient depth information is possible using various techniques; this is addressed in Section 3.1.4. If depth information is available, the inverse operation is represented as follows:

$$\mathbf{x}^c = \begin{bmatrix} x^c \\ y^c \\ z^c \end{bmatrix} = \pi^{-1}(\mathbf{u}, z) \tag{3.2}$$

A common camera model is the *perspective camera model.* This representation can be used to derive essential equations relating the variables of the target object and the resulting camera image. This model is illustrated in Figure 3.1. The x-axis of the camera frame $\mathbf{F}_c$ points to the right, the y-axis points down, and the z-axis points along the camera toward the object.

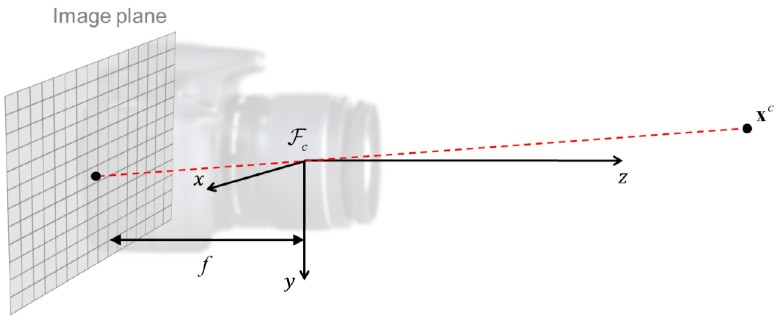


Figure 3.1: Illustration of the perspective camera model. Points $\mathbf{x}^c$ are projected onto the image plane through the origin of the camera frame $\mathcal{F}_c$. The image plane is located at a distance f behind the projective center, f being the camera's focal length.
Image source: Haavardsholm (2023).

A drawback of this model is the fact that the image is flipped. Thus, we introduce an alternative model where the image plane is placed in front of the projection center; see Figure 3.2 for an illustration of the *frontal projection model.* In particular, we place a *normalized image plane* such that the focal length equals 1, i.e., at $z = 1$.

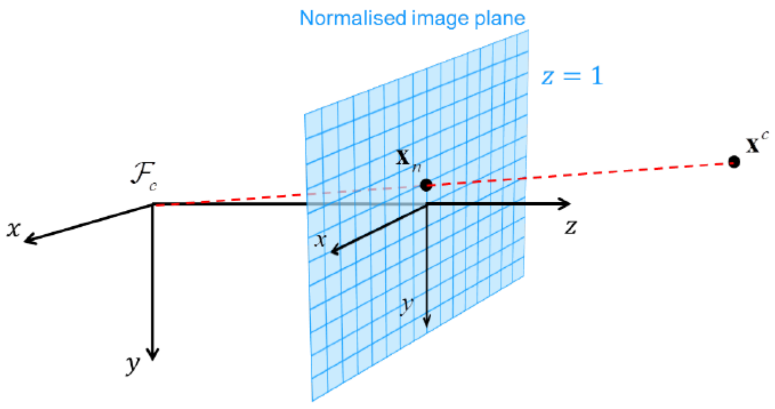


Figure 3.2: Frontal projection model with a normalized image plane, i.e., with a focal length of $f = 1$. A point $\mathbf{x}^c$ in the camera frame $\mathcal{F}_c$ is projected to a point $\mathbf{x}_n$ in the normalized image plane. The point where the z-axis of the camera frame intersects the normalized image plane is called the principal point.
Image source: Haavardsholm (2023).

Points on the normalized image plane are given the subscript n, e.g., $\mathbf{x}_n$. With the normalized model, we can describe the imaging process independently of the focal length f, for example, when performing stereo-vision using cameras with different *intrinsic parameters.* These camera-specific parameters can be collected into what is known as the 3x3 *intrinsic matrix* $\mathbf{K}$, which is discussed in more detail in Section 3.1.3.2. The intrinsic matrix takes into account scaling, shear, and translation and takes us from the idealized case to the camera-specific model:

$$\tilde{\mathbf{u}} = \mathbf{K}\tilde{\mathbf{x}}_n \tag{3.3}$$

To understand this transformation, we introduce the concept of homogeneous representations, a powerful tool used in computer vision and robotics. It allows for simple coordinate transformations using a single matrix multiplication while also being useful in projective geometry, where 3D points must be mapped to a 2D plane. Given a Cartesian 2D vector $\mathbf{u}$, we can introduce the homogeneous 2D vector $\tilde{\mathbf{u}}$ so that

$$\mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix} \in \mathbb{R}^2 \rightarrow \tilde{\mathbf{u}} = \breve{\mathbf{u}} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \in \mathbb{P}^2 \tag{3.4}$$

The vector $\tilde{\mathbf{u}} = \breve{\mathbf{u}}$ in Equation 3.4 is not only homogeneous but also normalized because its last coordinate is equal to 1. In general, this is not the case, and the homogeneous vector is mapped back into a Cartesian vector with:

$$\tilde{\mathbf{u}} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \in \mathbb{P}^2 \rightarrow \mathbf{u} = \begin{bmatrix} \tilde{u}/\tilde{w} \\ \tilde{v}/\tilde{w} \end{bmatrix} \in \mathbb{R}^2, \tag{3.5}$$

However, we can always normalize a homogeneous vector $\tilde{\mathbf{x}}$ with

$$\breve{\mathbf{x}} = \begin{bmatrix} \breve{x} \\ \breve{y} \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{x}/\tilde{z} \\ \tilde{y}/\tilde{z} \\ 1 \end{bmatrix} = \frac{1}{\tilde{z}}\tilde{\mathbf{x}}, \tag{3.6}$$

where $\breve{\mathbf{x}}$ is the normalized vector. Note that a point $\mathbf{x}_n$ on the *normalized image plane* should not be confused with a vector $\breve{\mathbf{x}}$ in the *normalized* form. Combining the two notations, a point $\breve{\mathbf{x}}_\mathbf{n}$ would be a *normalized* point on the *normalized image plane.*

As such, $\tilde{\mathbf{u}}$ in Equation 3.3 is the homogeneous pixel coordinate given in the image frame $\mathbf{F}_i$, which spans out the normalized image plane and has its origin at the top-left corner of the image; see Figure 3.3. Meanwhile, $\tilde{\mathbf{x}}_n$ is the homogeneous point in the normalized image plane. A point $\mathbf{x}^c$ in the camera frame can be projected onto the normalized image plane by

$$\tilde{\mathbf{x}}_n = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{\mathbf{x}}^c = \mathbf{x}^c, \tag{3.7}$$

and can also be seen as the representation of a 3D point on the plane when normalized as in Equation 3.6 so that its z-coordinate is equal to 1:

$$\breve{\mathbf{x}}_n = \frac{1}{\tilde{z}}\tilde{\mathbf{x}}_n = \frac{1}{z^c}\mathbf{x}_c \tag{3.8}$$

The intrinsic matrix $\mathbf{K}$ takes us from the ideal camera model to the camera-specific model. When knowing $\mathbf{K}$, we can map a pixel $\mathbf{u}$ to the corresponding normalized image coordinates, which are now *calibrated*:

$$\tilde{\mathbf{x}}_n = \mathbf{K}^{-1}\tilde{\mathbf{u}} \tag{3.9}$$

This operation is crucial because, as mentioned, it allows us to process the image data independently of the camera parameters.

Finally, the projection from a point $\mathbf{x}^c$ in the camera frame onto a pixel $\mathbf{u}$ in the image plane on homogeneous form is

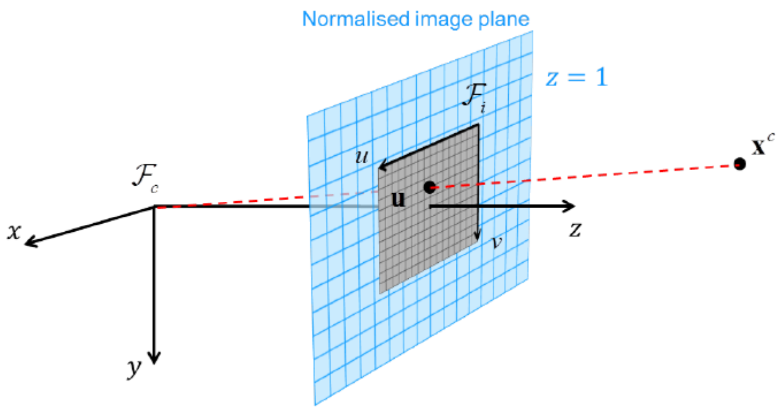


Figure 3.3: The normalized image plane is spanned out by the image frame $\mathcal{F}_i$, which is given in pixels. Its origin is located at the top left of the image. A point $\mathbf{x}_n$ in the normalized image plane is mapped to a pixel $\mathbf{u}$ in the image frame by an affine transformation.
Image source: Haavardsholm (2023).

$$\tilde{\mathbf{u}} = \mathbf{K} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{\mathbf{x}}^c = \mathbf{K}\mathbf{x}^c \tag{3.10}$$

exploiting the equality in Equation 3.7. To represent the equivalent projection on Euclidean form, we can leverage the fact that $\mathbf{K}$ preserves the normalized form of the homogeneous coordinates. Thus, we can normalize $\tilde{\mathbf{x}}_n = \mathbf{x}^c$ before the transformation and extract only the two first coordinates of $\breve{\mathbf{u}}$

$$\mathbf{u} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \breve{\mathbf{u}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{K} \frac{1}{z^c} \mathbf{x}^c, \tag{3.11}$$

finally arriving at the pixel $\mathbf{u}$.

The discussion on camera theory is finished by noting that real cameras often do not fit with the perspective camera model. Therefore, a distortion model is introduced to account for geometrical distortion arising from the optical system. This is described in more detail in Section 3.1.3.2.

3.1.2 Transformations

In the setting of a visual servo, the representation of points in space often needs to be moved between frames, e.g., from the camera frame to the robot base frame. This is made possible as a simple matrix operation through the transformation matrix. The set of valid transformation matrices is given by the *special Euclidean group* in 3D (Haavardsholm, 2023). A transformation matrix $\mathbf{T}_{ab} \in \mathbb{R}^{4x4}$ in this group is given as

$$\mathbf{T}_{ab} = \begin{bmatrix} \mathbf{R}_{ab} & \mathbf{t}^a_{ab} \\ \mathbf{0} & 1 \end{bmatrix} \tag{3.12}$$

Here, $\mathbf{R}_{ab}$ is the rotation matrix, describing the orientation of frame $\mathcal{F}_b$ relative to frame $\mathcal{F}_a$. Similarly, $\mathbf{t}^a_{ab}$ is the translation from the origin of frame $\mathcal{F}_a$ to the origin of frame $\mathcal{F}_b$ represented in $\mathcal{F}_a$. Together, they represent the *frame transformation* $\mathbf{T}_{ab}$, describing how frame $\mathcal{F}_a$ should be translated and rotated to coincide with frame $\mathcal{F}_b$. A vector $\mathbf{v}^b$ represented in $\mathcal{F}_b$ can be represented in $\mathcal{F}_a$ by

$$\tilde{\mathbf{v}}^a = \mathbf{T}_{ab}\tilde{\mathbf{v}}^b, \tag{3.13}$$

where $\tilde{\mathbf{v}}$ is the homogeneous form of $\mathbf{v}$. Note that *moving* a vector between frames does not actually move the vector seen from a global frame but instead alters its coordinates to allow its representation in the new frame.

Due to the structure of the transformation matrix, its inverse is relatively easy to compute:

$$\mathbf{T}_{ab}^{-1} = \begin{bmatrix} \mathbf{R}_{ab}^{\top} & -\mathbf{R}_{ab}^{\top}\mathbf{t}_{ab}^{a} \\ \mathbf{0} & 1 \end{bmatrix} \tag{3.14}$$

The inverse $\mathbf{T}_{ab}^{-1}$ allows the inverse operation

$$\tilde{\mathbf{v}}^{b} = \mathbf{T}_{ab}^{-1}\tilde{\mathbf{v}}^{\mathbf{a}} \tag{3.15}$$

to be performed, i.e. moving a vector from frame $\mathcal{F}_a$ to $\mathcal{F}_b$.

3.1.3 Camera calibration

3.1.3.1 Extrinsic calibration

Extrinsic calibration is about estimating the rigid transformation between coordinate frames. In an eye-to-hand scheme, extrinsic calibration would involve estimating the transformation between the robot base frame and the frame of the static camera. For an eye-in-hand scheme, which is relevant for the experiments conducted in this master's thesis, extrinsic calibration involves finding the transformation from the frame in the robot *tool center point* (TCP) to the camera frame, commonly referred to as the *hand-eye transformation.* Attention is directed towards the latter due to its relevance to this thesis.

A popular method for performing extrinsic calibration is chessboard calibration. The procedure involves capturing a series of images of a chessboard pattern from different angles using the camera mounted on the robot's end-effector and recording the corresponding robot pose for each image. Relative poses are used, which implies that knowing the absolute position and orientation of the chessboard in the robot base frame is not required as long as the chessboard remains stationary during the procedure.

A straightforward, top-level algorithm for capturing images and corresponding robot poses is presented in algorithm 2.

Algorithm 2: Capture chessboard images and robot poses for camera calibration

```
Data: Robot IP address, Save directory
Result: Color images and corresponding robot poses saved in the specified directory
while not all images captured do
    Move the robot to a new pose;
    Capture a color frame using the camera;
    Read the robot pose;
    Save the color image and the robot pose to a file;
```

After recording a series of images and poses, an algorithm for estimating the extrinsic parameters is presented in algorithm 3.

First, the two lists *objpoints* and *imgpoints* are initialized. For each image where chessboard corners are detected, these two lists are populated by 3D coordinates of chessboard corners in the chessboard frame and corresponding 2D points in the image plane, respectively. Note that each entry in the *objpoints* list is identical because the chessboard corners always have the same coordinates in the chessboard frame. These coordinates only depend on the chessboard dimensions. Next, robot poses are loaded from a file, and a check is performed to see if chessboard corners are detected in

Algorithm 3: Extrinsic (hand-eye) camera calibration

```
Data: Camera matrix, distortion coefficient, images of chessboard pattern with
      corresponding robot poses
Result: Hand-eye transformation matrix
Define chessboard size and square size;
Initialize empty lists: objpoints, imgpoints;
Set termination criteria for the iterative chessboard corner algorithm;
foreach image in the dataset do
    Convert image to grayscale;
    Find chessboard corners;
    if corners are found then
        Refine corner locations;
        Add object points and image points to lists;
    end
end
Load robot poses from file;
if number of robot poses matches the number of object points then
    Initialize empty transformation matrix;
    Initialize empty lists: robot rotation vectors, robot translation vectors, chessboard
      rotation vectors, chessboard translation vectors;
    Estimate rotation and translation vectors from camera to chessboard using
      PnP-algorithm with objpoints and imgpoints and add to lists;
    Extract robot rotation and translation vectors from robot poses and add to lists;
    Perform hand-eye calibration using the robot rotation vectors, robot translation
      vectors, chessboard rotation vectors, chessboard translation vectors, camera matrix,
      and distortion coefficients;
    Construct the hand-eye transformation matrix from the calibration results;
    return Hand-eye transformation matrix;
else
    Print error: mismatch between number of object points and robot poses, chessboard
      corners not recognized in all images;
end
```

all images. If the check fails, an error message is printed, and the algorithm does not return an estimate of the hand-eye transformation matrix. As such, the images yielding no corner detection must be removed from the dataset with the associated robot pose. Alternatively, the algorithm can be altered to ignore the robot poses associated with images where the detector cannot find corners. If the check is successful, however, an empty transformation matrix is initialized together with lists for robot rotation and translation vectors and chessboard rotation and translation vectors in the camera frame. The latter is computed by pairing the n 3D points of an entry in *objpoints* with the n 2D points of an entry in *imgpoints*. This is essentially a *Perspective-n-Point* (PnP) problem, where the goal is to find the rotation and translation that will transform the 3D world points to their corresponding 2D image points, which is analog to estimating the pose of the chessboard in the camera frame. Several methods are available for solving this problem; Pan and Wang (2021) present a review. Before performing the actual calibration step, the robot rotation and translation vectors are extracted from the robot poses. Ultimately, the hand-eye transformation matrix can be calculated. This problem is often formulated as $\mathbf{AX} = \mathbf{XB}$ where $\mathbf{X}$ is the hand-eye transformation matrix, as is illustrated in Figure 3.5.

The image shows the robot in two poses, each with an associated observation of the chessboard pattern. Since the chessboard pattern is stationary, the following equations can be set up:

$$\mathbf{T}_{RH} = \mathbf{T}_{RE_1}\mathbf{X}\mathbf{T}_{C_1H} \tag{3.16a}$$
$$\mathbf{T}_{RH} = \mathbf{T}_{RE_2}\mathbf{X}\mathbf{T}_{C_2H} \tag{3.16b}$$

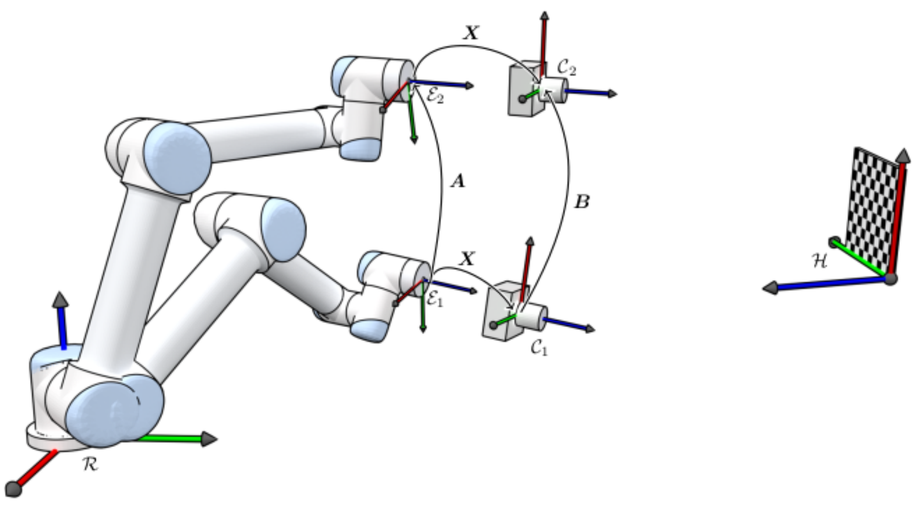


Figure 3.4: Illustration of two robotic poses during the chessboard calibration procedure. The robot poses are used to compute the **A** matrix, representing the transformation between the hand frame in the two poses. Similarly, observing the chessboard pattern from the two poses is used to compute the transformation **B** between the two camera frames, exploiting that the chessboard is stationary. These matrices are used when formulating the extrinsic calibration problem $\mathbf{AX} = \mathbf{XB}$, where **X** is the unknown hand-eye transformation. Image source: Myhre (n.d.).

equating to

$$\mathbf{T}_{RE_1}\mathbf{X}\mathbf{T}_{C_1H} = \mathbf{T}_{RE_2}\mathbf{X}\mathbf{T}_{C_2H} \tag{3.17}$$

Now, left-multiplying with $\mathbf{T}_{E_2R}$ and right-multiplying with $\mathbf{T}_{HC_1}$ gives

$$\mathbf{T}_{E_2R}\mathbf{T}_{RE_1}\mathbf{X} = \mathbf{X}\mathbf{T}_{C_2H}\mathbf{T}_{HC_1} \tag{3.18}$$

which is on the form $\mathbf{AX} = \mathbf{XB}$. Essentially, the **A** and **B** matrices are set up based on the recorded robot poses and the observations of the chessboard pattern, respectively. Note that several equations on this form can be set up when performing the calibration procedure. Naturally, this number depends on the number of poses and chessboard observations recorded, but also on whether one equation is set up for each pose in combination with every other pose or if independent equations are desired.

There exist several methods (Andreff et al., 1999; Daniilidis, 1999; Horaud and Fadi, 1995; Park and Martin, 1994), for solving this problem which are not covered here.

As a final note on extrinsic calibration, it is mentioned that algorithm 3 relies on an estimate of the camera matrix and distortion coefficients, explained in more detail in the upcoming section. As an alternative to having prior knowledge of these parameters, they can be estimated together with the rotation and translation vectors from the camera to the chessboard.

3.1.3.2 Intrinsic calibration

As mentioned, the camera matrix **K** takes us from the idealized case to the camera-specific case. It is given as

$$\mathbf{K} = \begin{bmatrix} f_u & s_\theta & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.19}$$

where the different elements of the matrix represent the following:

- c_u is the u-coordinate of the principal point given in the image frame $\mathcal{F}_i$.
- c_v is the v-coordinate of the principal point given in the image frame $\mathcal{F}_i$.
- s_θ is a skew-factor, where θ is the angle between the u and v axes in $\mathcal{F}_i$. s_θ is proportional to $cot(\theta)$.
- f_u is the focal length expressed in number of horizontal pixels.
- f_v is the focal length expressed in number of vertical pixels.

Performing intrinsic calibration includes finding the camera matrix $\mathbf{K}$ and a set of *distortion coefficients*, which describe the distortion model selected to relate the undistorted $\mathbf{x}_n$ and distorted points $\mathbf{x}_n^{'}$ in the normalized image plane. An example of such a model is the *radial distortion model* (Haavardsholm, 2023)

$$r^{'2} = x_n^{'2} + y_n^{'2} \tag{3.20a}$$

$$x_n = x_n^{'}(1 + k_1 r^{'2} + k_2 r^{'4}) \tag{3.20b}$$

$$y_n = y_n^{'}(1 + k_1 r^{'2} + k_2 r^{'4}), \tag{3.20c}$$

where k_1 and k_2 are the distortion coefficients. This model only considers radial distortion when straight lines appear curved. This effect is more pronounced the farther away points are from the center of the image.

Another form of distortion is tangential distortion, which occurs when the image plane and lens are not parallel. The distortion model used later in this master thesis considers both of these effects. Five coefficients parameterize the distortion; three coefficients, k_1, k_2, and k_3, describe the radial distortion, while p_1 and p_2 describe the tangential distortion. These coefficients are collected in a vector and given as:

$$Distortion\ coefficients = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{3.21}$$

Similar to when estimating extrinsic parameters, a chessboard can be used when performing intrinsic calibration. The intrinsic parameters can be computed using the known geometry of the chessboard and the produced 2D image points. Given a series of pictures of a chessboard pattern from different angles, a high-level algorithm for computing the intrinsic parameters is given in algorithm 4.

Algorithm 4: Intrinsic camera calibration

```
Data: Images of chessboard pattern captured from different poses
Result: Camera matrix, Distortion coefficients
Define chessboard size and square size;
Initialize empty lists: objpoints, imgpoints;
Set termination criteria for the iterative chessboard corner algorithm;
foreach image in the dataset do
    Convert image to grayscale;
    Find chessboard corners;
    if corners are found then
        Refine corner locations;
        Add object points and image points to lists;
    end
end
Perform camera calibration method using objpoints and imgpoints;
Save camera matrix and distortion coefficients to file;
```

The algorithm estimates the camera matrix and distortion coefficients after looping over the images in the dataset to find chessboard corners. This part of the algorithm is a new algorithm in itself and could, for example, be Zhang's method as described in detail in Burger (2016). Zhang's method, explained briefly, first computes a homography relating the 2D image points to the 3D object points on the planar object. This is possible because the pattern is planar, simplifying the transformation to a 2D-2D correspondence. From the calculated transformations, the camera parameters can be estimated. Lastly, an optimization process, e.g., a least-squares method, is applied to minimize the *re-projection error*, which is the difference between the observed corner positions in the images and the positions predicted by the estimated camera parameters.

When comparing the extrinsic and intrinsic calibration algorithms, it becomes clear that performing these calibrations in a joint procedure can reduce the total overhead. This is discussed in more detail later in this thesis when the calibration is performed in practice.

3.1.4 Depth estimation

As addressed previously, recovering the 3D point $\mathbf{x}^c$ from the corresponding pixel $\mathbf{u}$ as represented in Equation 3.2 requires additional information about the depth z. From Equation 3.11 we can re-arrange and formulate

$$\mathbf{x}^c = z^c\mathbf{K}^{-1}\breve{\mathbf{u}} = z^c\mathbf{K}^{-1}\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3.22}$$

indicating that the inverse operation requires the depth z^c.

This section will introduce two established methods to acquire this required depth information. Firstly, the stereo vision method is discussed due to its widespread use, and secondly, enforcing geometric constraints on objects in the scene is considered due to its relevance to this thesis. Several other methods exist, such as techniques involving machine learning (Srinivasan et al., 2018) or focus-based depth estimation (Pertuz et al., 2018).

3.1.4.1 Stereo vision and triangulation

In stereo vision, two cameras with a known relative pose are used to estimate the 3D position of an object. Identifying corresponding points between the images captured by the two cameras is known as *matching.* It can be done using techniques such as *feature matching* (Rublee et al., 2012), where corresponding key features such as corners and edges are detected in the two images. When the disparity between these corresponding features is measured, depth information can be computed.

Epipolar geometry is the geometry associated with the stereo vision problem. Given two perspective cameras with corresponding camera frames $\mathcal{F}_a$ and $\mathcal{F}_b$, and a known relative pose $\mathbf{T}_{ab}$, we observe the same point $\mathbf{x}$ from two perspectives. Observing the same point in space from two perspectives constrains how the given point is projected in the two images. We call this the *epipolar constraint.*

The plane containing the origin of both camera frames and the observed point in space is called the *epipolar plane.* The line between the camera centers is called the *baseline*, and the line where the image plane intersects the epipolar plane is called the *epipolar line.* One way of formulating the epipolar constraint is that a point in space must lie on corresponding epipolar lines for the two cameras.

A mathematical formulation of the epipolar constraint can be formed by relating the corresponding points $\mathbf{x}_n^a$ and $\mathbf{x}_n^b$ in the normalized image planes of the respective cameras. By introducing the *essential matrix* $\mathbf{E} \in \mathbb{R}^{3x3}$ given by

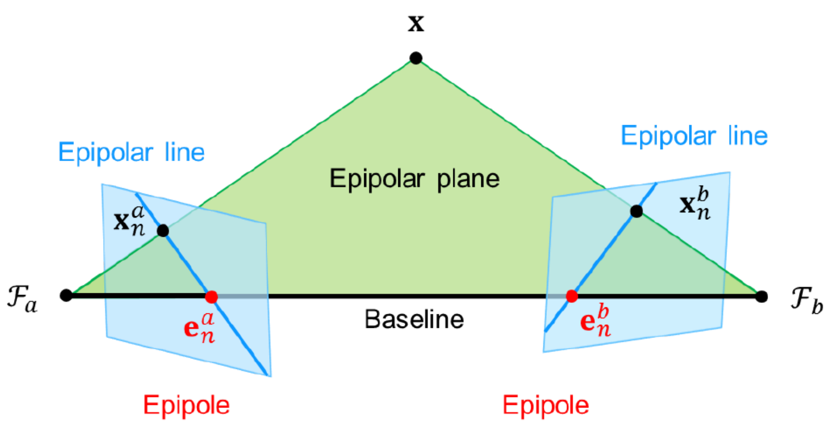


Figure 3.5: Illustration of key elements in epipolar geometry. A point $\mathbf{x}$ is observed by two cameras in their respective camera frames $\mathcal{F}_a$ and $\mathcal{F}_b$. The epipolar plane contains the centers of the cameras and the observed point, and the epipolar lines are the intersections between this plane and the image planes. The constraint relating how points $\mathbf{x}$ are projected to points $\mathbf{x}_n^a$ and $\mathbf{x}_n^b$ in the normalized image planes is called the epipolar constraint.
Image source: Haavardsholm (2023).

$$\mathbf{E}_{ab} = [\mathbf{t}_{ab}^a]_\times \mathbf{R}_{ab}, \tag{3.23}$$

with $[\mathbf{t}_{ab}^a]_\times$ being the skew-symmetric matrix associated with $\mathbf{t}_{ab}^a$, we can formulate the epipolar constraint as

$$\tilde{\mathbf{x}}_n^{a\top} \mathbf{E}_{ab} \tilde{\mathbf{x}}_n^b = 0 \tag{3.24}$$

As this constraint is formulated in the normalized image plane, it follows that a similar constraint formulated between corresponding *pixels* $\mathbf{u}^a$ and $\mathbf{u}^b$ in the two image planes must be related to the intrinsic camera matrices $\mathbf{K}_a$ and $\mathbf{K}_b$ of the two cameras. In fact, an equivalent constraint can be formulated as

$$\tilde{\mathbf{u}}^{a\top} \mathbf{F}_{ab} \tilde{\mathbf{u}}^b = 0 \tag{3.25}$$

where $\mathbf{F}_{ab} \in \mathbb{R}^{3x3}$ is the fundamental matrix, related to the essential matrix through

$$\mathbf{F}_{ab} = \mathbf{K}_a^{-\top} \mathbf{E}_{ab} \mathbf{K}_b^{-1} \tag{3.26}$$

where $\mathbf{K}_a^{-\top}$ is the transpose of the inverse of $\mathbf{K}_a$.

The theory on stereo vision is finished by mentioning that the nature of the epipolar constraint restricts the search for corresponding points to the epipolar line. As such, the distance between a candidate point correspondence and the epipolar line can be evaluated to test the validity of the candidate point.

3.1.4.2 Geometrical constraints

Another technique for solving the problem of having insufficient depth information is to enforce constraints on the geometry observed in the image to reduce or possibly remove the ambiguity of the 2D projection, enabling the backprojection into a point in 3D space. Using Equation 3.19 and Equation 3.22, the coordinates of the point $\mathbf{x}^c = [x^c \ y^c \ z^c]^\top$ in space as a function of the pixel coordinates are given as

$$\mathbf{x}^c = \begin{bmatrix} x^c \\ y^c \\ z^c \end{bmatrix} = z^c \begin{bmatrix} \dfrac{u - c_u}{f_u} \\ \dfrac{v - c_v}{f_v} \\ 1 \end{bmatrix}, \tag{3.27}$$

providing us with 2 equations and 3 unknown variables, as the third equation, $z^c = z^c$, is trivial and provides no value. This suggests that introducing constraints will provide additional equations, giving the problem a unique solution. In the following, this concept is discussed by investigating backprojection where reference points on an object are constrained relative to each other. The selected shapes, all of them rigid bodies, are as follows:

- A *rod*, using the two end-points as reference points.
- An *equilateral triangle*, using the corners as reference points.
- A *square*, using the corners as reference points.

The constraint is the Euclidean between two points $\mathbf{x}_1 = [x_1\ y_1\ z_1]^\top$ and $\mathbf{x}_2 = [x_2\ y_2\ z_2]^\top$ defined as

$$d^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2, \tag{3.28}$$

enforced between all reference points in a pairwise manner. Due to each reference point having three coordinates, the number of degrees of freedom (DOF) for each of the three shapes is

$$DOF = 3 * \textit{num. reference points} - \textit{num. constraints} \tag{3.29}$$

Given two reference points and one constraint, the rod represents a problem containing 5 degrees of freedom (DOF). Using Equation 3.27 for both end-points, a set of 4 equations can be made, yielding one slack variable and no unique solution. A simple illustration is the following: if the projected points move closer to each other, there is no way of telling if this is because the rod is (a) moving away from the camera without rotating or (b) rotating in the horizontal plane.

Next, consider an equilateral triangle. With three reference points and three Euclidean distance constraints present, this is a 6 DOF problem. Using Equation 3.27 for all three points, a set of 6 equations can be made, suggesting that a unique solution might exist. However, seeing as the Euclidean distance is a quadratic function in the coordinates, the problem yields two possible solutions for the pose of the triangle. With no other information available, the pose of the triangle cannot be determined as a unique solution. For illustration, imagine the triangle oriented in space such that its normal vector points directly at the camera. The resulting image projection is a perfect equilateral triangle in the 2D plane. Now, one of the corners in the projection moves towards the line connecting the two other corners due to a rotation around this line in space. There would be no way of telling if this is because the moving corner is rotating *away* or *towards* the camera.

Lastly, consider a square representing a $3 * 4 - 6 = 6$ DOF problem. Again, using Equation 3.27 for each reference point, we get a set of 8 equations, finally yielding a unique solution to the backprojection problem, enabling the pose of the square to be determined. This result is exploited later in this thesis when performing ArUco-marker tracking.

Before finishing the discussion on geometric constraints, note that various constraints exist other than the Euclidean distance, which might yield different conclusions in the discussion above. One example could be to enforce all points to lie in the same plane in space, possibly removing ambiguity. Additionally, other features like edges can be used instead of corners.

3.1.5 Lie theory - mean of poses

Lie Theory is briefly mentioned as a tool for calculating the mean of poses. This could become relevant in a scheme where the imaging system operates at a higher rate than the robot controller, meaning that several images are available for processing for each iteration in the control loop. While such a technique could become useful to increase accuracy in the estimated pose, it could also be problematic if the object has moved in the camera frame in between images used for calculating a mean pose. The following theory is mainly based on lecture material from *TTK21 - Introduction to Visual Simultaneous Localization and Mapping* delivered at NTNU (*TTK21 Introduction to Visual Simultaneous Localization and Mapping - VSLAM*, 2023) and the associated hand-book (Haavardsholm, 2023). For a more comprehensive review of Lie theory, please refer to Solá and Deray (2018), whose work has also been used for supporting material in this section.

Poses can be represented as points $\mathcal{X}$ on a smooth manifold $\mathcal{M}$ in higher-dimensional space. The dimensionality of $\mathcal{M}$ is denoted m, which for the $SE(3)$ group is equal to 6. Performing mathematical operations like addition on a pose $\mathbf{T} \in SE(3)$ lying on this manifold is problematic because the resulting pose $\mathbf{T}^{new}$ might not be a valid pose in the $SE(3)$ group. Therefore, we introduce a tangent space $\mathcal{TM}_{\mathcal{X}}$ at the point $\mathcal{X}$ on the manifold, see Figure 3.6. If $\mathcal{X}$ is equal to the identity $\mathcal{E}$, we call this tangent space the *Lie algebra* $\boldsymbol{m}$ of $\mathcal{M}$. The identity point $\mathcal{E}$ is the point associated with the origin frame of the problem at hand. Elements $\hat{\boldsymbol{\tau}} \in \boldsymbol{m}$ can be identified with vectors $\boldsymbol{\tau} \in \mathbb{R}^m$ in the tangent space through the operator Vee: $\boldsymbol{m} \rightarrow \mathbb{R}^m$, allowing linear algebra to be performed. The inverse operation is Hat: $\mathbb{R}^m \rightarrow \boldsymbol{m}$. These mappings are denoted $(\cdot)^\vee$ and $(\cdot)^\wedge$, respectively.

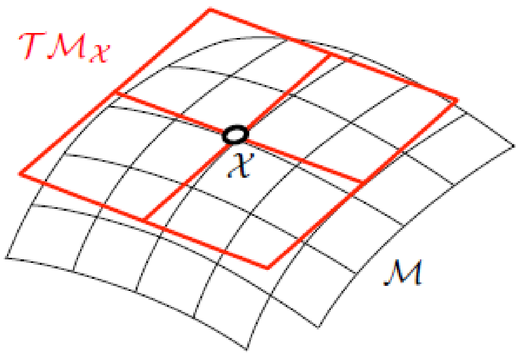


Figure 3.6: A tangent space $\mathcal{TM}_{\mathcal{X}}$ at the point $\mathcal{X}$, lying on the smooth manifold $\mathcal{M}$. Image source: Solá and Deray (2018), license CC BY-NC-SA 4.0.

An essential aspect of Lie theory when discussing the mean of poses is that each pose on the smooth manifold has an associated vector in the tangent space. A vector on the tangent space is wrapped onto the manifold using *the exponential map*, and the unwrapping operation is called *the logarithmic map*, see Figure 3.7.

A summary of the mappings Vee, Hat, $exp()$, and $log()$ can be seen in Figure 3.8. Note that the capitalized exponential map and logarithmic map $Exp()$ and $Log()$ are convenient notations that let us map a vector $\boldsymbol{\tau} \in \mathbb{R}^m$ directly to an element $\mathcal{X} \in \mathcal{M}$ in the Lie group and vice versa, instead of going via the Lie algebra $\boldsymbol{m}$.

Now, a method for calculating the mean of a set of poses can be made. First, an initial guess is made; this is where the initial tangent plane is centered. Next, we use the logarithmic map on the set of poses to present them in the tangent plane, where linear algebra can be utilized to find a mean. Then, the exponential map is applied to wrap the mean back onto the manifold. Note that this method only provides a rough approximation of the mean on the Lie group because the resulting pose represented on the manifold is not the center of the tangent plane. For more precise results, iterative algorithms may be required; a new tangent plane is centered in the calculated mean pose, and the procedure is repeated until convergence is achieved. Such an algorithm, as proposed in the course *TTK21 Introduction to Visual Simultaneous Localization and Mapping -*

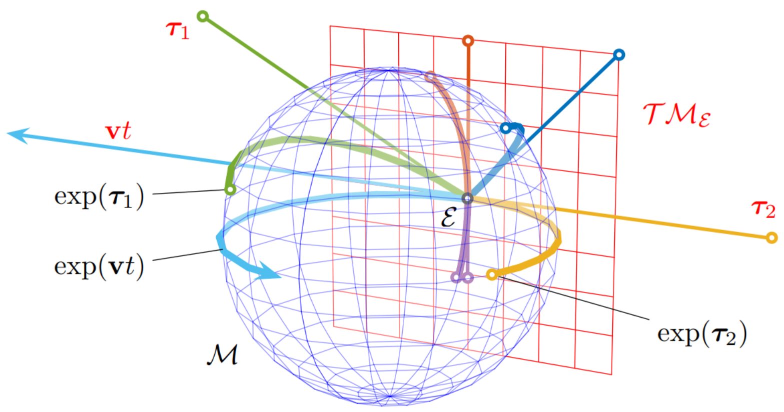
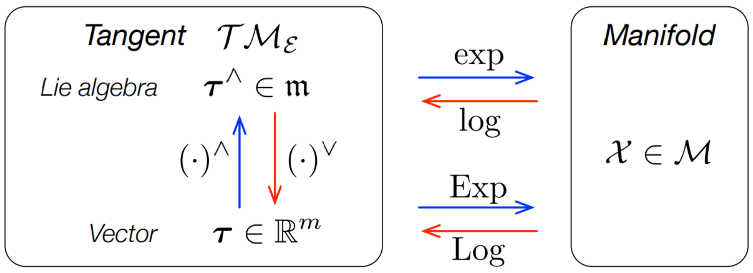


Figure 3.7: Points $\boldsymbol{\tau}_i$ and a line $\mathbf{v}t$ on the tangent plane $\mathcal{TM}_{\mathcal{E}}$ passing through the identity $\mathcal{E}$ are wrapped onto the manifold $\mathcal{M}$ using the exponential map *exp*().
Image source: Solá and Deray (2018), license CC BY-NC-SA 4.0.



Tangent $\mathcal{TM}_{\mathcal{E}}$
Lie algebra $\boldsymbol{\tau}^\wedge \in \mathfrak{m}$
$(\cdot)^\wedge$
$(\cdot)^\vee$
Vector $\boldsymbol{\tau} \in \mathbb{R}^m$
exp
log
Exp
Log
Manifold
$\mathcal{X} \in \mathcal{M}$

Figure 3.8: An overview of the mappings between the manifold $\mathcal{M}$ and the tangent space at the identify $\mathcal{TM}_{\mathcal{E}}$. Hat $(\cdot)^{\hat{}}$ and vee $(\cdot)^v$ represent the mappings between the Lie algebra $\boldsymbol{m}$ and the tangent vector space $\mathbb{R}^m$. Meanwhile, exp() and log() map to/from the Lie algebra and the manifold, and Exp() and Log() are shortcuts for mapping the tangent vector space to/from $\mathcal{M}$ directly.
Image source: Solá and Deray (2018), license CC BY-NC-SA 4.0.

VSLAM (2023) delivered at NTNU, is seen in algorithm 5.

Algorithm 5: A proposed algorithm for calculating the mean of poses using Lie theory. The operators $\oplus$ and $\ominus$ represent plus and minus operators used to express perturbations on poses.

```
Initialise for example with $\mathbf{T}^0 \leftarrow \mathbf{T}_1$ ;
for $t = 0, 1, \ldots, t_{max}$ do
    Compute the mean tangent vector in the tangent space at $\bar{\mathbf{T}}^t$;
    $\bar{\boldsymbol{\xi}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{T}_i \ominus (\bar{\mathbf{T}}^t)$;
    Update the hypothesis;
    $\mathbf{T}^{t+1} \leftarrow \mathbf{T}^t \oplus \bar{\boldsymbol{\xi}}$;
    if $\|\mathbf{T}^t \oplus \mathbf{T}^{t+1}\|_2 < \epsilon$ then
        $\mathbf{T} \leftarrow \mathbf{T}^{t+1}$;
        return;
    end
end
```

3.2 Kalman filter - position and velocity estimation

Disclaimer: this section (Section 3.2) is a nearly identical version of work from the specialization project (Hagestuen, 2023*a*).

Although the 3D coordinates of the tracked object can be acquired using the techniques addressed so far, the position estimate can be refined by incorporating the object's dynamics into the problem formulation. Additionally, estimating the object's velocity could also be of interest, e.g., for use in the servo controller. The Kalman filter solves both these problems elegantly.

At its core, the Kalman filter is a recursive filter that estimates the internal state of a linear dynamic system from a series of noisy measurements. It operates by predicting the system's state in the next time step (prediction step) and then adjusting this prediction based on the new measurement data (correction step). The combination of the prediction step and the correction step allows the filter to account for system dynamics and new measurements simultaneously.

Handling the process and measurement noise matrices is essential to the Kalman filter operation, denoted **Q** and **R** respectively. Process noise represents uncertainties or random disturbances present in the model of the system's dynamics. It accounts for the aspects of the actual system not captured by the dynamical model used in the filter. Meanwhile, measurement noise represents uncertainties in the measurements.

Selecting appropriate values for **Q** and **R** practically translates to tuning the filter. Both process noise and measurement noise are modeled as Gaussian white noise, and $\mathbf{Q}, \mathbf{R} \in \mathbb{R}^n$, n being the dimension of the state vector $\mathbf{x}$ to estimated, are often selected as diagonal matrices. This choice of **Q** and **R** assumes no covariance between noise in different states/measurements. Selecting higher values for the elements of **Q** indicates more significant uncertainty in the system dynamics, causing the filter to trust the model predictions less and rely more on the incoming measurements. Conversely, higher values in **R** suggest more significant uncertainty in the measurements, leading the filter to lean more on the model predictions than on the measurement updates.

The iterative algorithm of the Kalman filter comprises two main steps:

- **Prediction Step**: The state vector and covariance are projected forward to the next time step using the system's model.

- **Correction Step**: The predicted state vector and covariance are corrected with the incoming measurement using the Kalman gain, which balances the predicted state and the measurement based on the uncertainties represented by **Q** and **R**.

When employing a Kalman filter, a dynamical model representing the system's behavior to be observed is crucial to predict the states. In hand tracking, it is challenging to select a model because human behavior is highly unpredictable. As such, the random walk model is proposed, where the current position is considered equal to the previous position plus a random term. This is essentially made possible by assuming a Gaussian process in velocity.

3.3 Neural networks

This section will cover the fundamental theory on neural networks (NNs), and how they can be used for object detection. Core principles are covered, sufficient to follow the experiments conducted later in this master thesis. The focus is directed towards *multilayer perceptron*, one of the most essential classes of NNs (Jawad, 2023).

3.3.1 Fundamentals about neural networks

NNs make up the base of deep learning, a field within machine learning that is, in turn, a branch of artificial intelligence. In short, NNs take in data and are trained to recognize patterns and predict an output based on a new set of data not seen during the training procedure.

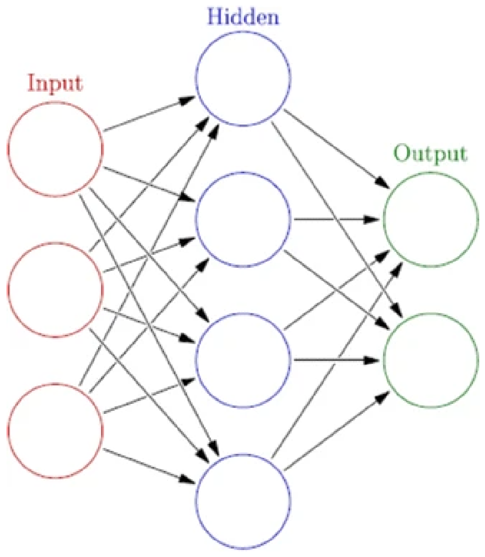


Figure 3.9: Structure of a neural network, consisting of neurons (circles), channels (arrows), and layers (columns of neurons).
Image source: Wu and Feng (2017), license number 5784870330787 granted through CCC RightsLink.

The structure of a NN is inspired by the human brain and is made of layers of *neurons*, as visualized in Figure 3.9. The first layer, the *input layer*, receives the input data. Depending on the application, this data could be pixels in an image, economic policies, or data about a real estate property about to be put up for sale. The *output layer* is the final layer and predicts the output after the data has propagated through the network, for instance, which objects are in an image, the economic inflation, or the final sales price of a property. In between the input and output layers exist the *hidden layers*, where most of the processing in the network takes place. These layers are hidden because they are not visible to the outside world; only the input and output layers are exposed to the user. The network displayed in Figure 3.9 only contains one hidden layer, but this number varies based on the network's design.

A neuron can be seen as the processing unit in the network, and neurons in neighboring layers are connected through *channels*. Each channel is assigned a *weight*, and the input to the neurons in the next layer consists of a weighted sum of the output from the neurons in the previous layer. Additionally, the neuron is associated with a *bias*. As such, the input to a given neuron in the next layer is in the form

$$y_i = x_1 w_1 + x_2 w_2 + ... + x_n w_n - B_i \tag{3.30}$$

where $x_1, x_2, ..., x_n$ are values from neurons in the previous layer, $w_1, w_2, ..., w_n$ are weights associated with the channels connecting from the prior layer, and B_i is the bias. Next, this value is passed through an *activation function*, which determines whether the given neuron should be active. An essential aspect of the activation function is the introduction of non-linearity to the network, which is instrumental to achieving non-linear and complex problem-solving (Jawad, 2023). Many activation functions can be used in the design of the network. One of the most popular ones is the sigmoid function

$$\rho(x) = \frac{1}{1+e^{-x}}, \tag{3.31}$$

which, regardless of the input, will generate a value between 0 and 1. Another example of an activation function is the Heaviside function

$$\rho(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0. \end{cases} \tag{3.32}$$

The bias B can be selected to impact at which region the neuron becomes active, which becomes clear when writing the combined expression for the neuron:

$$\rho(x_1, ..., x_n) = \rho(\sum_{i=1}^{n} x_i w_i - B), \tag{3.33}$$

where ρ is an arbitrary activation function.

3.3.2 Training a neural network

Training a neural network involves selecting the channels' weights and the neurons' biases. As one can understand, the number of parameters increases quickly with the number of neurons and layers. A common method is called *supervised learning*, where a set of known input and output data is used to train the network. Training consists of a cycle of *forward propagation* and *back-propagation*, which is iteratively repeated with multiple pairs of inputs and outputs from the dataset.

Forward propagation is how input data propagates through the network to generate a prediction. A prediction is essentially the values of the neurons in the output layer, which can be compared to the ground truth since the correct output data is already known. The difference between the prediction and the ground truth is calculated using a loss function $\mathcal{L}$.

Back-propagation is how the loss information is used to adjust the weights and biases in the network. As stated by Jawad (2023), training a fully connected network is often tricky; a proposed strategy to decrease training difficulty is to set selected weights to zero before initiating the training procedure. In essence, training is an optimization problem which, as suggested by Jawad (2023), can be put on the general form

$$\min_{(W^{(l)}, B^{(l)})_l} \sum_{i=1}^{m} \mathcal{L}(\phi(W^l, B^l)_l(x^{(i)}) - y^{(i)}) + \lambda \mathcal{P}((W^l + B^l)_l), \tag{3.34}$$

where:

- $W^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}$ is the weight matrices of the neurons in the lth layer, with N_l noting the dimension of the lth layer.
- m is the number of samples in the dataset used for training.
- $\mathcal{L}$ is the loss function used to measure the closeness of the prediction and the ground truth values $y^{(i)}$.
- $\phi : \mathbb{R}^d \to \mathbb{R}^{N_L}$ is the neural network itself, representing the forward propagation from input to prediction, with d being the dimension of the input layer, and N_L the dimension of the output layer.
- $\mathcal{P}$ is a penalty term that introduces additional weight and bias constraints.

A common approach to solving this problem is gradient descent. However, this is not computationally feasible since m can be very large. Therefore, a proposed solution is to only evaluate a few gradients at each iteration, assuming they represent a reasonable average. This is called stochastic gradient descent (Jawad, 2023).

An essential aspect of training is selecting hyperparameters used in the training process. Two important hyperparameters are *epochs* and *learning rate*. An epoch is one complete round of running through the dataset, adjusting the weights and biases according to the algorithm. At the start of the next run, the weights and biases from the previous run are used as starting points. A natural thought is that a higher number of epochs is better. However, there are two reasons why deliberately increasing the number of epochs can be undesirable: firstly, the training procedure will take a more extended amount of time, and secondly, the model might start to overfit the parameters towards the training data. Overfitting causes a model to recognize patterns and details specific to the training data, resulting in poor predictions when applied to new data. Next, selecting an appropriate learning rate is a vital part of the training process; while a high learning rate might lead to failure in loss value convergence, causing a failure in the learning process, a too-small learning rate might lead to a slow learning procedure (Sellat et al., 2022).

A common practice is to use an adaptive learning rate, meaning that the learning rate is a function of the current epoch number. Such a learning rate schedule can lead to faster learning without risking the procedure not converging to a minimum loss value (Sellat et al., 2022).

3.3.3 Neural networks for object detection

As a final note on neural networks, the following section is devoted to how a neural network can be used for object detection.

The input layer in a neural network designed for object detection typically receives image pixels as input to the neurons. As such, it becomes clear that the dimension of this layer will grow rapidly with higher image resolutions. The output layer, often called the *classification layer*, will determine which object (if any) has been detected; the neuron with the highest value determines the output. Based on the design of the network, the hidden layers might be responsible for performing preliminary calculations such as segmenting areas of interest, detecting edges and corners, and combining such features into more complex shapes.

Training data for an NN designed for object detection consists of images labeled with objects from the object classes the network is trained to recognize. These labels can be pixel masks of the objects (segmentation) or rectangular boxes encapsulating the objects (bounding boxes). As experiments conducted later in this master's thesis utilize bounding boxes, the focus is directed toward the latter.

When evaluating the performance of the model after training, a standard metric is *average precision* (AP). To understand this metric, one must comprehend the term *Intersection over Union* (IoU), which states how much of the predicted bounding box intersects the ground truth bounding box compared to the union of these two boxes. I.e., IoU is the area of overlap divided by the area of union. IoU ranges from 0 to 1 and can be used as a threshold for correct detections when comparing the predicted and ground truth bounding boxes. However, selecting a firm threshold of, e.g., 0.50 will introduce a bias, and the resulting AP does not contain sufficient information about the model performance. As such, the IoU threshold can be changed over a range at certain increments. For instance, the IoU threshold can gradually be changed from 0.50 to 0.95 at increments of 0.05, yielding 10 APs. The average of these 10 APs is denoted AP@[0.50:0.95] and can provide more useful information about the model than utilizing a firm IoU threshold at, e.g., 0.75. Note that mean average precision (mAP) is the AP values averaged over the object categories, but these two terms are often used interchangeably in the literature. Naturally, a higher AP value is desirable, with $AP = 1.0$ being a perfect score and $AP = 0.0$ describing a model with no successful detections when evaluated on the validation dataset.

Chapter 4

Techniques for enhanced human-robot collaboration

This chapter aims to define the scope of techniques (hereafter also referred to as methods) subject to experiments in this thesis, exploring their potential for enhanced human-robot collaboration. In particular, the focus is directed towards two topics. Firstly, **visual servoing is explored as a potential means of interacting with cobots** to define positions and guide the robot based on the operator's hand movements. A key challenge when exploring visual servoing as a potential means of interacting with cobots is how the robot can track the hand position of the operator. Researching this topic can be motivated in the setting of enhanced human-robot collaboration due to its potential to facilitate intuitive interaction between operators and cobots, potentially increasing productivity and safety in collaborative tasks. Secondly, this thesis will explore the possibility of communicating with robots using natural language by **mapping language prompts to robotic tasks**. The aim is to exploit the knowledge contained in LLMs while also making the robot aware of its surroundings using object detection. Developments within this topic may contribute to enhanced human-robot collaboration by introducing a new means of communication when performing situation-aware task planning.

4.1 Hand tracking for visual servoing

Disclaimer: this section (Section 4.1) is a nearly identical version of work from the specialization project (Hagestuen, 2023*a*).

After discussions with supervisors M. H. Arbo and J. T. Gravdahl and discoveries made in the literature study, it was brought up that using machine learning could be a viable technique for hand detection and possibly an enabling technology for hand tracking in 3D. In particular, if a machine learning model for hand detection is combined with a depth camera, one could estimate the 3D position of a hand in space. By mounting a depth camera on the robot's end-effector and performing real-time position estimation, one could make the robot aware of the position of the operator's hand. It also became interesting to look into the hand-detection capabilities of machine learning in typical workspace situations where the human could be wearing gloves and holding tools. Ultimately, the aim of exploring these techniques is to investigate if they can offer enhanced human-robot collaboration, particularly by reducing the dependency on code to perform manufacturing tasks.

Next, the ArUco marker was selected as a topic of interest due to its possible relevance to hand tracking. It can also be seen as an extension to the research on hand detection, for example, in the direction of object detection. In short, the ArUco marker is a square marker consisting of a matrix pattern of black and white squares; see Figure 4.1 for an example. Each unique marker has an identifier and belongs to a group called a *dictionary* depending on how many inner squares it consists of. For instance, the 5x5 dictionary group contains 5x5 inner squares. The ArUco marker offers fast and reliable detection. It can be used for camera pose estimation without using stereo vision by exploiting the known geometry and dimensions of the marker, as discussed in the theory chapter (Section 3.1.4.2).

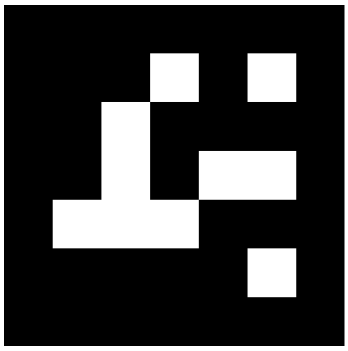


Figure 4.1: Example of an ArUco marker. This particular marker has ID=100 and belongs to the 5x5 dictionary group.

4.2 Mapping language prompts to robotic tasks

Based on discoveries made in the literature study, it became of interest to conduct experiments aiming to leverage the knowledge contained in LLMs to map natural language prompts to robotic tasks. Typically, the language prompt represents a task that the operator requires the robot to perform, and the chosen sequence of robotic actions should meet this requirement. An essential aspect of this research is the role of an LLM in such a system, with a particular focus on the GPT model family. This involves evaluating the responses from different GPT models and understanding how the formulation of prompts affects the responses, i.e., looking into the importance of prompt engineering.

Next, as motivated by findings from the literature study, this research aims to investigate how an existing object detection model can be re-trained for new object categories using MediaPipe Model Maker and a custom dataset. The focus is on enabling the detection of tools and objects commonly used in manufacturing settings, typically not part of existing open-source models and datasets. The MediaPipe Model Maker solution is especially interesting due to the limited amount of research conducted on it, as highlighted in the literature study. A portion of the research will include creating the dataset necessary to re-train the model for the desired object categories.

The objective is to utilize a machine learning model for object detection, enabling the robot to become aware of its surroundings. Thus, the natural language prompt from the operator can be mapped into a realistic set of tasks by exploiting the rich knowledge contained in an LLM. This research aims to simplify the robotic interface to language inputs, making robotic systems more accessible to non-technical operators.

In the next chapter, an experimental test setup is implemented and documented. Then, three experiments involving (1) **visual servoing** and four experiments related to (2) **mapping language prompts to robotic tasks** will be conducted. Experimental results are presented and discussed with regard to technical details in addition to whether they indicate the feasibility or infeasibility of the methods for visual servo schemes, prompt-to-task systems, or related applications in manufacturing settings.

Chapter 5

Experimental setup

5.1 Hardware

Disclaimer: this section (Section 5.1) is a nearly identical version of work from the specialization project (Hagestuen, 2023*a*).

The setup consists of a Universal Robots (UR) UR10 CB3 robot mounted on a table with a corresponding control box, an Intel RealSense D455 camera mounted on the end-effector of the robot, and a computer running the Python-script implementing the desired task. The computer is connected to the robot control box through an Ethernet cable and to the camera via USB-C. The Intel camera mounts to the end-effector with a bracket, which was 3D-printed by researcher Andrej Cibicik at SINTEF Manufacturing. The 3D-printed bracket attaches to the robot with M6 bolts and the camera with M4 bolts. For use in ArUco marker tracking, printed ArUco markers were glued on a flat surface, also 3D-printed by researcher Andrej Cibicik. Figure 5.1, Figure 5.2, and Figure 5.3 show images of the robot, Intel camera with bracket, and ArUco marker surface, respectively.

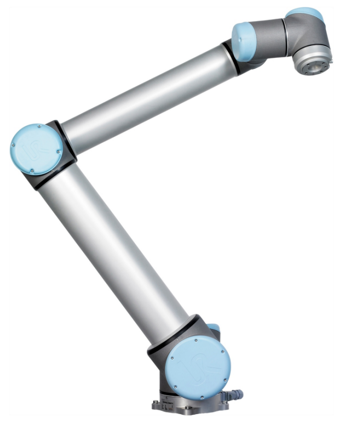


Figure 5.1: Image of Universal Robots 10 CB3, the cobot used in the experiments. Image source: UniversalRobots (2023).

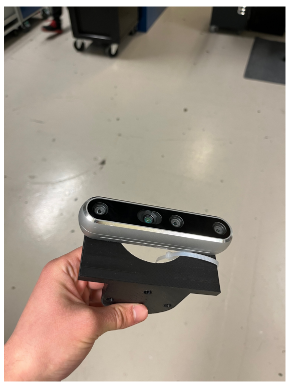


Figure 5.2: Intel RealSense D455 camera mounted on the bracket used to attach it to the robot end-effector.

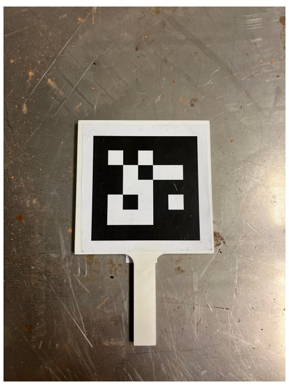


Figure 5.3: ArUco marker glued to a 3D-printed surface to keep it flat. The surface has a handle for easy operation.

Intel RealSense D455 is an RGB-D camera capable of outputting the scene's color (RGB) and depth (D) data in real time. As seen in Figure 5.2, the camera consists of four lenses. The biggest of the four lenses is the RGB lens, while the ones on either side are the left and right imaging lenses in the stereo vision system. Optionally used to improve depth accuracy for scenes with low texture, the smaller lens in the middle is an infrared (IR) projector. The stereo system correlates points from the left and right image and calculates the depth at each pixel using epipolar geometry. The D455 device operates with a 640x480 resolution throughout the experiments.

The UR10 CB3 cobot is a 6-rotational joints DOF manipulator with a carrying capacity of 10kg. The system consists of the robot, the control box, and the teach pendant. For details on the robot, refer to UniversalRobots (2023). A static IP address was used when connecting to the robot control box. The default option for new connections in Windows 10 is automatically assigned IP addresses, so it was necessary to manually configure the properties in the Windows 10 control panel for this connection.

A chessboard pattern was utilized for calibration; Section 5.3 describes this procedure in more detail.

5.2 Software

This section introduces the software tools and dependencies for the code base used in the experimental setup of this project. The reader can find all scripts used for calibration procedures, performing experiments, and processing the results in the open-source GitHub repositories (Hagestuen, 2024*a*) and (Hagestuen, 2024*b*) made when working on this master thesis. The first repository includes code for hand tracking in visual servo systems, while the second repository provides scripts for object detection and mapping language prompts to robotic tasks. As such, the reader can fully explore the different libraries' usage on GitHub. Additionally, this section summarizes the main functionality used from each package in bullet points. For details on installation of the software dependencies, refer to their respective documentation web pages.

5.2.1 Poetry

Disclaimer: the upcoming four sections (Section 5.2.1 to Section 5.2.4) are nearly identical versions of work from the specialization project (Hagestuen, 2023*a*).

Poetry is a Python package manager and dependency-handling tool. It allows the user to declare the libraries used in a project so that Poetry can manage them, for example, by keeping them updated or removing them from the project. As such, Poetry is used in this master thesis to manage the packages introduced in the upcoming subsections. Note that Poetry is not an absolute necessity. Instead, one can see it as a convenience for package handling.

5.2.2 OpenCV

OpenCV is an open-source computer vision and machine learning library. It has built-in support for various algorithms ranging from face and object recognition to stereo vision technologies such as 3D point cloud generation from stereo cameras and feature matching between images. The functionality from OpenCV used in this project is summarized:

- **Chessboard corner detection** in images.
- **ArUco marker detection** and corresponding marker pose estimation, with support for configuring the ArUco detector with parameters such as certainty threshold and desired corner refinement algorithm.
- **Image processing**, particularly color conversion from RGB to BGR and vice versa, and projecting observed chessboard corners to the image plane as a function of the estimated camera intrinsic parameters. The latter was relevant when calculating re-projection error after calibration.

- **Intrinsic camera calibration** given an observed chessboard pattern with known dimensions and a set of images of this chessboard pattern.
- **Hand-eye calibration** given a set of robot poses, observed chessboard patterns, and camera intrinsic parameters.
- **Writing and reading images to and from file**.

5.2.3 Real-Time Data Exchange

For interfacing with the UR10 CB3 robot used in the experiments, this work employed the UR Real-Time Data Exchange (RTDE) API for Python. In particular, the following functionality from the library was utilized:

- **Path following** - specifying a path through pre-defined points in space for the robot to follow with parameters such as speed, acceleration, and smoothing parameters.
- **Linear servo** - specifying a point in space for the robot to move towards. No configuration options were available concerning speed and acceleration, so this function was wrapped in a more sophisticated method, generating intermediate positions in the desired direction as a function of the selected servo gain.
- **Receiving robot pose** - used for data recording, mainly for post-processing.

5.2.4 Intel RealSense Software Development Kit

For interfacing with the Intel RealSense D455 camera, the scripts use the Python wrapper of the Intel RealSense Software Development Kit (SDK). The key functionalities utilized in this library were:

- Initializing a **frame pipeline** used to get the next frame captured by the camera device.
- **Frame alignment** between color frame and depth frame.
- **Depth estimation** at selected pixel coordinates.

5.2.5 MediaPipe

Two modules were used from MediaPipe, an open-source library suite: the **hand detection solution**, a machine learning model for detecting hand landmarks, and **MediaPipe Model Maker**, a module for customizing existing machine learning models. The upcoming subsections will briefly introduce these two modules.

5.2.5.1 Hand detection solution

Disclaimer: this subsection (subsection 5.2.5.1) is a nearly identical version of work from the specialization project (Hagestuen, 2023*a*).

Several computer vision and machine learning libraries (OpenCV, TensorFlow, Keras, and MediaPipe) were looked into. It appeared that the only library with a pre-trained machine learning model for hand detection was MediaPipe. The other libraries instead provide a framework for implementing and training custom models or integrate with pre-trained models from third parties. It was decided to proceed further with MediaPipe. For a simple demonstration of MediaPipe's hand-detection capabilities, see Figure 5.4 and Figure 5.5. A total of 21 landmarks, four for each

Figure 5.4: Image of Erling Braut Haaland without annotated hand-landmarks.
Image source: PA News Agency via Peeblesshire (2022).

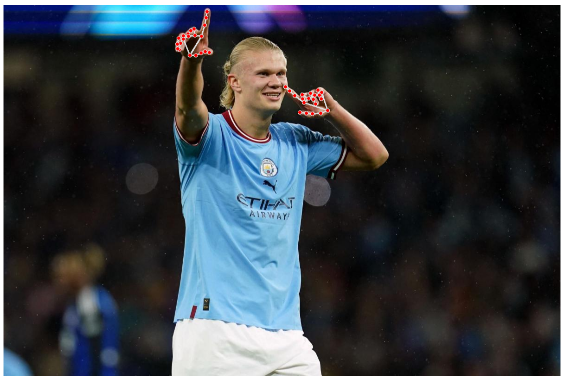


Figure 5.5: Image of Erling Braut Haaland with annotated hand-landmarks as produced by MediaPipe.
Image source: PA News Agency via Peeblesshire (2022).

finger and one on the wrist, are generated with corresponding image coordinates for each hand detected in the image.

While MediaPipe plays a crucial role in detecting hands in images in this master thesis, it is essential to note that MediaPipe is not a machine-learning library. Instead, it is an open-source framework developed by Google that is designed to simplify building pipelines for machine learning models. It provides a structure for integrating various media processing components, such as the machine learning model for hand detection.

Figure 5.6 shows a flowchart of the hand detection solution in MediaPipe. Other solutions include body-tracking, object detection, and face detection.

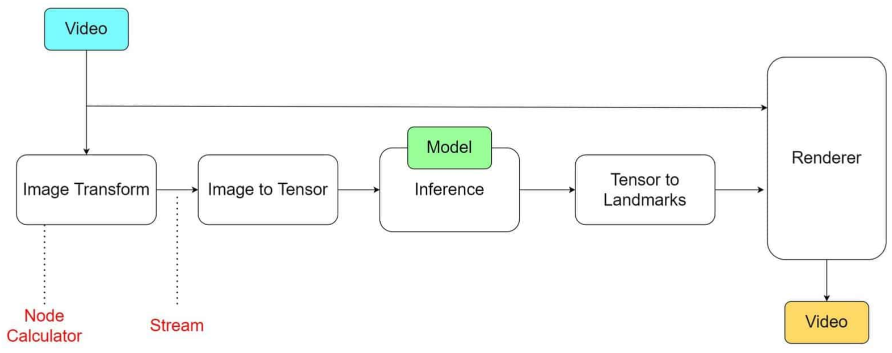


Figure 5.6: Flow chart of the hand-detection solution in MediaPipe, consisting of image pre-processing, data structuring, landmark detection, and video rendering.
Image source: Kukil (2022).

A *tensor* is a generalized form of other data structures that represent multi-dimensional data. For example, a matrix is a 2-dimensional tensor. Tensors store and manipulate data, which can be input to or output from models or model parameters.

The flow chart shows how each frame is processed in the hand detection solution. First, it is pre-processed and represented as a tensor, suitable for input into the machine learning model. The model outputs a tensor, which is then converted into landmarks. These landmarks contain information about each point's x and y coordinates. Note that the rendering and video output nodes are only of interest if the desired output is a video stream annotated with the hand landmarks. In the experiments conducted in this master thesis, the coordinates of the landmarks are either written directly to a data file for post-processing or used in real-time, e.g., depth estimation. The general strategy for performing depth estimation on the hand is to get pixel coordinates of hand landmarks from MediaPipe using the color image from the Intel Realsense RGB lens and make a call to the Intel RealSense SDK library to get a depth estimate at the given pixels, exploiting the stereo vision abilities of the camera.

Initial testing quickly uncovered that the depth request to the RealSense library on a specified pixel coordinate produced by MediaPipe yielded varying results. In particular, the received depth estimate quickly altered between what could seem like a reasonable value and the default return value of $z = 0.0m$. After observing that holding the hand horizontally in the camera view produced more stable results than holding it vertically, the issue was identified as frame misalignment between the color and depth images. Without adjusting for this misalignment, the depth request on the depth frame could be on a different pixel than desired, e.g., a pixel in the background of the hand where the depth frame yielded limited information. As such, alignment of depth and color frames had to be incorporated into the algorithm whenever depth estimation was performed.

Note that MediaPipe has built-in support for depth to landmarks *relative to the wrist* in world coordinates. To get a depth estimate of an arbitrary landmark, the depth to the wrist still has to be estimated using other techniques. As such, this specific functionality has limited value in hand tracking in 3D space.

5.2.5.2 Model Maker for re-training object detection model

MediaPipe Model Maker, introduced in the literature review (Section 2.3.3), is a module designed to adapt pre-existing machine learning models with custom data. It was employed to re-train the MediaPipe Object Detection solution for new object categories. MediaPipe Model Maker utilizes a technique called *transfer learning*, allowing a significant portion of the existing model to be re-used. Therefore, this tool provides a quicker alternative than creating a new model from scratch (Google, 2023*b*).

MediaPipe Model Maker was used on Ubuntu due to discontinuation of support for one of its dependencies, *tensorflow/text*, on Windows starting from version 2.11 (Google, 2022*b*). This dependency is not directly used in this project, meaning that MediaPipe Model Maker could still be installed on Windows without dependencies, followed by a manual installation of each requirement independently. However, Ubuntu was used for simplicity when working with MediaPipe Model Maker.

As such, the functionality used from the MediaPipe library is summarized as follows:

- **Hand detection** in images, with the option to configure detector parameters such as maximum number of hands in the image and minimum detection confidence. Upon detecting a hand, information about handedness is available from MediaPipe and used in one experiment.
- Re-training of the **Object Detection solution** for new object categories using the **Model Maker module**.

5.2.6 FiftyOne

FiftyOne is an open-source tool for building datasets for machine learning models. In this thesis, FiftyOne's Python package was utilized to download and explore datasets from the FiftyOne Dataset Zoo to find hammer and screwdriver images for a custom dataset. Additionally, the research used FiftyOne to randomly separate the dataset into train/validate/test sets according to the desired splits after labeling it.

As such, this thesis used FiftyOne to:

- **Download and explore existing datasets** to find images in the desired object categories.
- **Splitting the dataset** after labeling it.

5.2.7 OpenAI

OpenAI's Python API was utilized to interact with the GPT language models, facilitating queries to ChatGPT, where the user can specify the desired engine for each query. Note that the API is a paid service with pricing structured on a per-usage basis. Each engine has specific prices, presented as a fixed amount in dollars (or cents) per 1000 tokens, approximately equivalent to 750 words (OpenAI, 2024*c*). OpenAI's price list provides an updated overview of the prices for each model.

When querying the LLM through the API, users can provide a previous conversation as input, providing the LLM with contextual information about a potential preceding dialogue. This functionality resembles users' familiar conversational experience with ChatGPT via the website. A vital aspect of the previous conversation lies in the three roles *user*, *assistant*, and *system* in the conversation, where the additional system role can be utilized to establish ground rules or provide the assistant with instructions for the upcoming conversation.

Thus, the functionality used from the OpenAI Python library is:

- Making **queries to the GPT language models** through the Open AI Python API.

5.2.8 Label Studio

Label Studio, a software tool for labeling and preparing datasets, was utilized to annotate images with bounding boxes to create a dataset for re-training an object detection model. This thesis used the *object detection with bounding boxes* template for image annotation. It utilized the COCO dataset format for export, one of many formats Label Studio supports.

A summary of the usage of Label Studio is:

- **Importing images and annotating them** with bounding boxes.
- **Exporting the dataset** in COCO format.

5.3 Camera calibration

Disclaimer: this section (Section 5.3) is a nearly identical version of work from the specialization project (Hagestuen, 2023*a*).

An 18x29 chessboard with 10mm marker size was used for extrinsic and intrinsic calibration, meaning the same image series could be used for both calibrations. See Figure 5.7 and Figure 5.8 for images of the chessboard pattern. Note that these two images are taken using an iPhone camera and should not be confused with images captured with the Intel RealSense D455 camera used in the experimental setup. Typically, a chessboard with fewer and bigger squares is used for extrinsic calibration, as a dense pattern of inner corners is not equally important in extrinsic calibration as in intrinsic calibration, where they are crucial for calculating the distortion coefficients. Using a chessboard with fewer and bigger squares for extrinsic calibration would allow the corner detector to identify the corners of the chessboard more easily, allowing images to be captured from more challenging angles and positions. This is a potential weakness when using the same chessboard for extrinsic and intrinsic calibration.

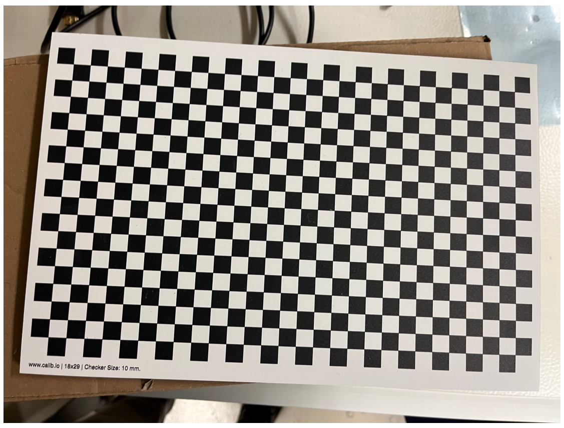


Figure 5.7: Chessboard pattern used in the calibration procedure. The image was captured using a mobile phone camera.

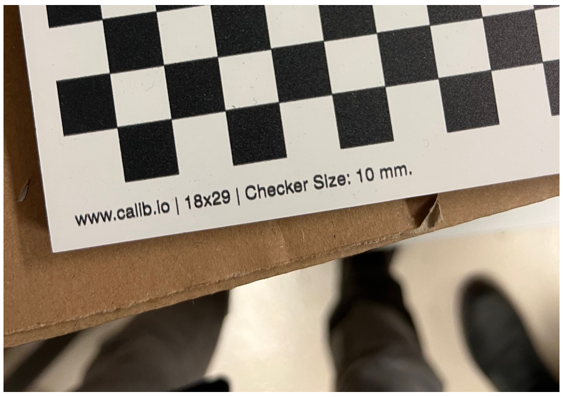


Figure 5.8: Dimensions of the chessboard pattern. The image was captured using a mobile phone camera.

The calibration data was acquired using algorithm 2. The Python script implementing this algorithm is available in the GitHub repository (Hagestuen, 2024*a*). Images of the setup used in the procedure can be seen in Figure 5.9 and Figure 5.10.

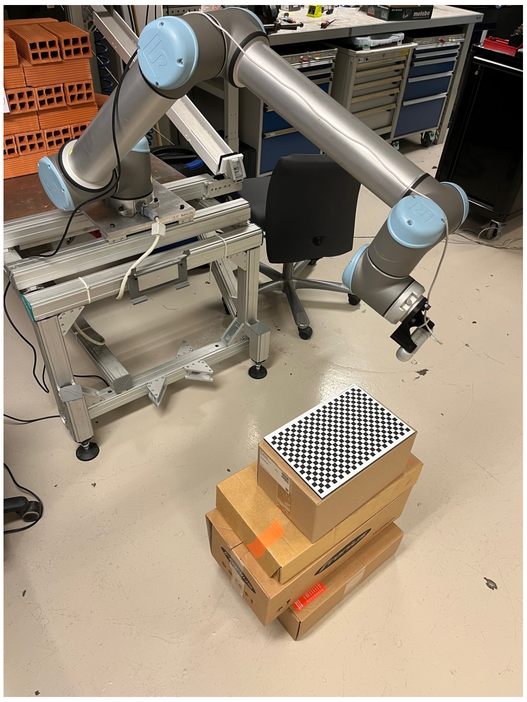


Figure 5.9: Intel RealSense D455 mounted on UR10 CB3 end-effector capturing images of the chessboard pattern as part of the calibration procedure.

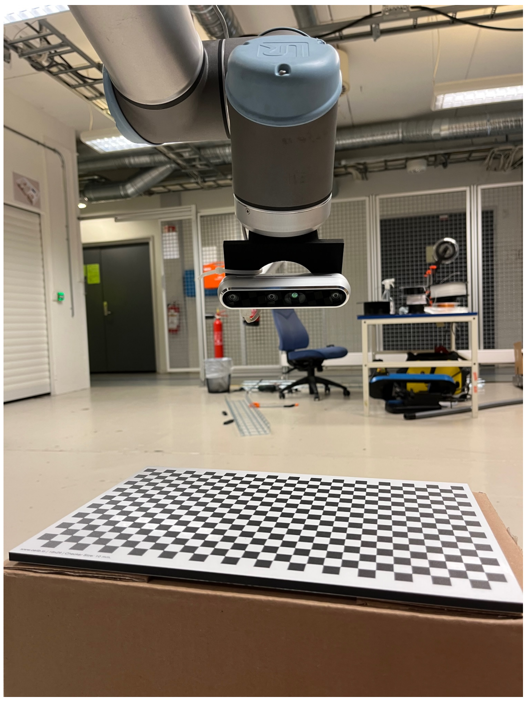


Figure 5.10: Close-up image of the Intel RealSense D455 camera and the chessboard pattern while recording images for the camera calibration procedure.

5.3.1 Intrinsic calibration

Discussions with researcher Eirik Njåstad at SINTEF Manufacturing brought up that the Intel RealSense camera parameters could change throughout an experiment due to changing temperatures in the camera, representing a potential error source. A proposed countermeasure is to calibrate at a stable operating temperature where the camera will be used, for example, after 20 minutes of operation. However, this does not solve the problem entirely due to the varying camera workload during experiments, which is thought to be correlated with its internal temperature. This error source was not devoted much attention other than the awareness of its presence, possibly explaining any inconsistent results.

The intrinsic camera parameters were estimated using the top-level algorithm in algorithm 4. This algorithm was implemented in a common Python script together with the algorithm for calculating the extrinsic parameters due to similarities in the two procedures, including chessboard corner detection. Additionally, the extrinsic calibration procedure requires an estimate of the intrinsic camera parameters, which are acquired together with the translation and rotation vectors associated with the chessboard in the image frame through a common method in OpenCV. Combining extrinsic and intrinsic calibration to a joint script reduced the overhead due to these commonalities. Refer to GitHub (Hagestuen, 2024*a*) for a full exploration of the joint script. Note that only the inner corners of the chessboard pattern are used in the algorithm, meaning that for an 18x29 chessboard, there are 17x28 corners of interest as seen in Figure 5.11.

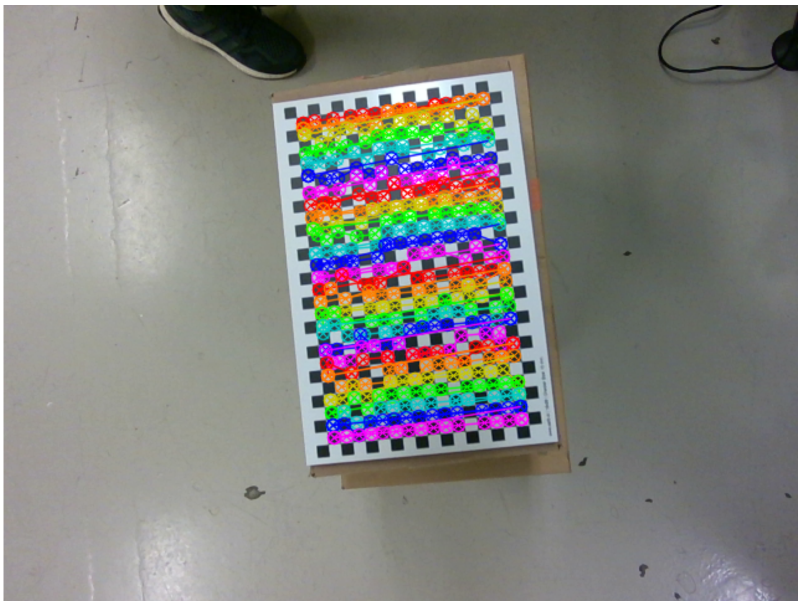


Figure 5.11: Chessboard pattern with inner corners marked as produced by chessboard corner detector in OpenCV. Note that inaccuracies can be spotted, e.g., in the top row, which is populated by both orange and red markers. This is expected when using a dense chessboard with a relatively low resolution of 640x480 pixels.

The resulting estimate of the camera matrix was

$$\mathbf{K} = \begin{bmatrix} f_u & s_\theta & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 312.78 & 0.0000 & 313.68 \\ 0.0000 & 313.22 & 263.46 \\ 0.0000 & 0.0000 & 1.0000 \end{bmatrix} \tag{5.1}$$

The distortion coefficients, presented on the format seen in Equation 3.21 were estimated to be:

$$distortion\ coefficients = \begin{bmatrix} -0.228 & 0.441 & 0.00438 & -0.00160 & -0.397 \end{bmatrix} \tag{5.2}$$

It was deemed unnecessary to perform an *undistortion test*, i.e., using the calculated camera matrix and distortion coefficients to remove the distortion effect in a set of new images. This was because the camera did not seem to suffer from noticeable distortion; see sample images in Figure 5.12 and Figure 5.13.

Instead, a re-projection error was calculated for all images as part of the calibration procedure. The aim was to detect and remove apparent outliers to improve the estimate of the camera matrix.

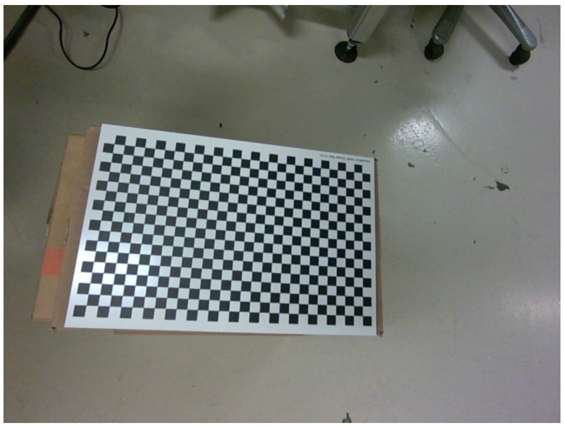


Figure 5.12: Sample image of the chessboard pattern captured with Intel RealSense D455 used in the calibration procedure.

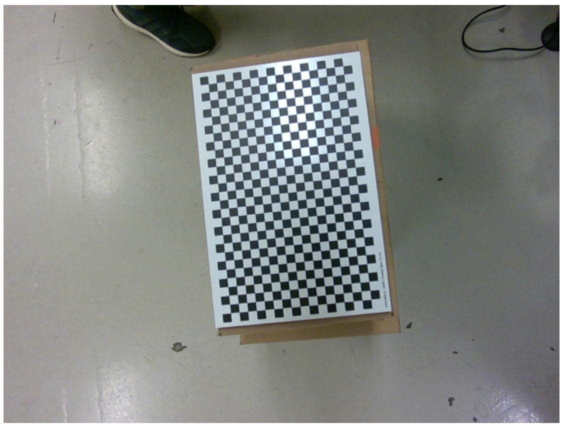


Figure 5.13: Sample image of the chessboard pattern captured with Intel RealSense D455 used in the calibration procedure.

Given the rotation and translation vectors associated with each chessboard, the camera matrix, and the distortion coefficients, the re-projection error was calculated for each image as the absolute norm between all corners in the re-projection and the original images. A lower average re-projection error means a more accurate estimate of the camera intrinsics, while the re-projection error for a single image describes how precise the estimated camera parameters are for the given image, possibly exposing it as a potential outlier. The results can be seen as a scatter plot in Figure 5.14 together with horizontal lines representing the mean error and the error plus/minus three standard deviations away from the mean.

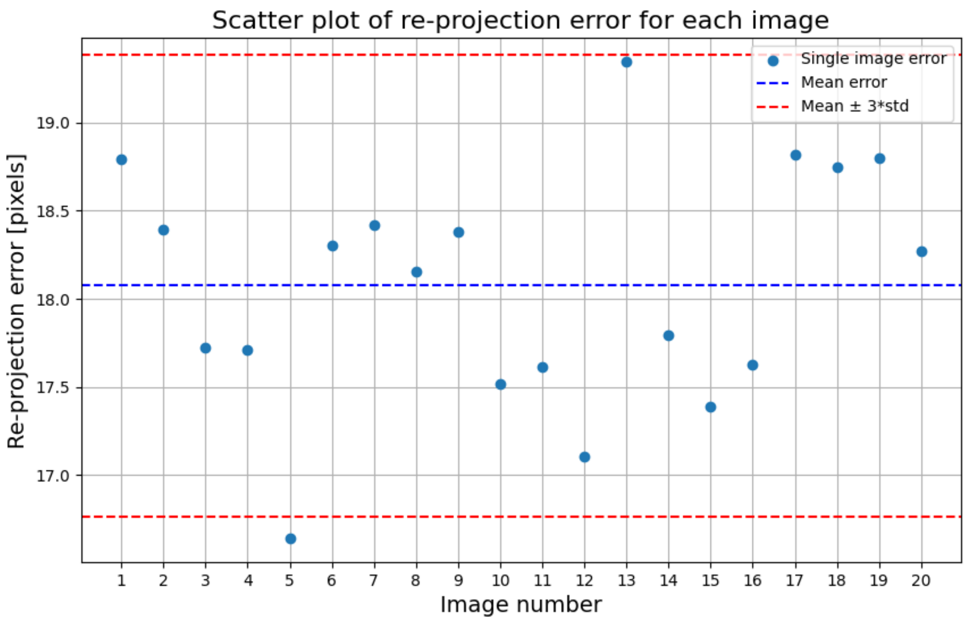


Figure 5.14: Scatter plot of re-projection errors after calibration, calculated as the average absolute norm between all corners in the original and re-projected images using the calculated transformation. Each dot represents the re-projection for one image. The blue line is the average re-projection error, and the red lines represent the mean ±3 standard deviations.

The re-projection error plot highlights the discrepancies between images. Using the standard deviation as a reference, however, it was concluded that there are no apparent outliers. Therefore, no images were removed from the dataset.

5.3.2 Extrinsic calibration

The extrinsic calibration aims to find the transformation matrix describing how the *hand* frame $\mathcal{H}$ of the robot should be translated and rotated to align with the *camera* frame $\mathcal{C}$, i.e., finding

$$\mathbf{T}_{hc} = \begin{bmatrix} \mathbf{R}_{hc} & \mathbf{t}_{hc}^{h} \\ \mathbf{0} & 1 \end{bmatrix} \tag{5.3}$$

The extrinsic calibration was performed on the same images as the intrinsic calibration, now with a corresponding robot pose for each image. Here, algorithm 3 was used for computing the extrinsic parameters and was implemented in a Python script together with the intrinsic calibration, as mentioned.

The results were as follows:

$$\mathbf{T}_{hc} = \begin{bmatrix} -0.66 & -0.048 & 0.75 & 0.090 \\ 0.75 & 0.031 & 0.67 & 0.11 \\ -0.054 & 0.99 & 0.015 & 0.056 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

As a sanity check, the hand-eye transformation matrix was estimated by hand to allow for comparison. The translations $x = 0.070m$, $y = 0.070m$, and $z = 0.055m$ were roughly measured using a ruler. Next, visually evaluating the Euler angles of sequential unit axis rotations led to an estimate of the rotation matrix. Using a XYZ Roll-Pitch-Yaw convention, the angles given in radians were $\theta_x = \frac{pi}{2}$, $\theta_y = \frac{3pi}{4}$ and $\theta_z = 0$, yielding the following transformation matrix:

$$\mathbf{T}_{hc} = \begin{bmatrix} -0.71 & 0.00 & 0.71 & 0.070 \\ 0.71 & 0.00 & 0.71 & 0.070 \\ 0.00 & 1.0 & 0.00 & 0.055 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.5}$$

Comparing this with the result from the calibration seen in Equation 5.4, discrepancies can be seen in both rotation and translation. However, these differences are reasonable to accept due to the inaccuracy of using a ruler and visual inspection to estimate translations and rotations. In general, they are pretty similar, and the conclusion is that the sanity check was passed, and the calibration was successful.

Chapter 6

Experimental results and discussion

This chapter covers the introduction, method, results, and discussion for each experiment conducted in this master thesis. The introduction aims to motivate each experiment's relevance to potential use cases. The methodology, as documented for each test in this chapter, should be seen as an extension to the description of the experimental setup in chapter 5. This means that core elements in the setup, such as the UR10 CB3 robot with corresponding control box and interface and Intel RealSense D455 camera together with its interface, are used and are not described in detail for each test. As mentioned in a previous chapter, the results are to be discussed concerning technical details and whether the results indicate feasibility or in-feasibility of the methods for visual servo schemes, prompt-to-task systems, or other applications in manufacturing settings.

The experiments are separated into two groups: (1) **visual servoing as a potential means of interacting with cobots**, and (2) **mapping language prompts to robotic tasks** using LLMs and object detection for real-world grounding.

In the first group, the initial experiment implements a visual servo scheme exploiting an ArUco marker for tracking with corresponding discussions on potential applications and limitations. Next, MediaPipe's hand detection solution was evaluated as a candidate for introducing machine learning to the hand tracking problem. Limitations of this solution were highlighted by employing it in challenging workspace environments involving tools, work gloves, and hand occlusion. In combination with a depth camera, MediaPipe's hand detector was used to implement a device-less visual servo system, allowing discussions on the expanded possibilities of such a system.

In the second group of experiments, LLMs were employed to map natural language prompts into a predetermined set of robotics tasks. Next, the open-source MediaPipe object detector was re-trained for new object categories using a custom dataset and the transfer learning technique supported in the MediaPipe framework. Further, the resulting model was evaluated in workspace situations by attempting to generate a list of objects based on a workbench scene. Lastly, the LLM and the object detector were put into the broader prompt-to-tasks system (see Figure 6.15), allowing discussions on the potential and limitations of such a system.

Thus, the research consists of a total of seven experiments.:

- Experiments involving hand tracking methods for visual servo systems:
  - Visual servo with ArUco marker tracking - Section 6.1.1.
  - Hand detection in typical workspace situations using MediaPipe - Section 6.1.2.
  - Visual servo using MediaPipe's hand detection solution coupled with the Intel RealSense depth camera for hand tracking - Section 6.1.3.
- Experiments on how language prompts can be mapped to robotic tasks using LLMs and an object detector:

– Sequencing robotic tasks based on natural language prompts using LLMs - Section 6.2.1.
– Re-training MediaPipe's object detection model for new object categories using a custom dataset - Section 6.2.2.
– Generating an item set based on visual inspection of a workbench scene using the re-trained object detector - Section 6.2.3.
– Prompt-to-tasks demonstrator system: leveraging the knowledge in an LLM to map language prompts into robotic tasks while being grounded in the real world using object detection - Section 6.2.4.

Some of the experimental results are presented in video format and have been uploaded to YouTube. Along with clickable links provided in this chapter, a summary of full-text links can be found in Appendix C.

6.1 Hand tracking for visual servoing

The experiments and discussions in this section are related to hand-tracking techniques for visual servo systems. In particular, the ArUco marker and MediaPipe's hand detector combined with a depth camera are researched as potential hand-tracking solutions. The corresponding discussions shed light on the potential and limitations of these techniques in real-world applications.

6.1.1 Experiment 1 - Visual servo with ArUco marker tracking

Disclaimer: this experiment (Section 6.1.1) is a derivative of work from the specialization project (Hagestuen, 2023*a*).

6.1.1.1 Introduction

ArUco markers are relevant for applications requiring an object's position in space to be estimated. An example is performance analysis in sports, such as attaching a marker to the shoes of an athlete runner on a treadmill and tracking their movement. Another example could be position estimation of a hand-held controller in AR/VR games.

Additionally, visual servo systems can exploit the tracking capabilities of an ArUco marker in settings where the operator is not required to have free hands. One example could be for lifting purposes where the operator needs to move an object around in space, for example, when inspecting a part from different angles. A different use case is performing supporting tasks for bricklaying in construction, e.g., handing the next brick to the worker from a pallet. At SINTEF Manufacturing, this is being researched in an ongoing EU project named HumanTech (SINTEF, 2023). As such, this experiment aims to implement a visual servo system using ArUco marker tracking.

6.1.1.2 Remarks from the specialization project on ArUco marker tracking

Before moving on to the visual servo system's implementation details, these remarks summarize key findings from a preliminary experiment conducted as part of the specialization project (Hagestuen, 2023*a*). The results obtained from this experiment influenced the choice of ArUco marker size to be used in the visual servo system. The objective was to determine the physical size of the ArUco marker for the system. Two candidate sizes were considered: $10cm$ x $10cm$ and $2.8cm$ x $2.8cm$, with selection criteria based on relative movement tracking accuracy and detection success rate. A natural hypothesis was that the larger marker would yield better results. However, research was still necessary to determine if a possible performance trade-off could be worth it for the physically smaller marker.

The experiment involved a stationary marker and a camera mounted on the robot's end-effector. Assuming a perfectly stationary marker and a precise robot trajectory, the relative movement between the camera and the marker would serve as a measure of the tracking method's performance for the two marker sizes. Additionally, the detection success rate, defined as the number of frames where a marker was detected as a fraction of the total images captured, was evaluated for both sizes.

The robot's end effector was moved 20cm along each axis, resulting in a total Euclidean distance traveled of

$$l = \sqrt{(0.2m)^2 + (0.2m)^2 + (0.2m)^2} \approx 0.346m. \tag{6.1}$$

The estimated Euclidean distance traveled by the ArUco marker in the camera frame was compared to the distance moved by the robot's end-effector in the robot base frame. Trajectories for both marker sizes are presented together in Figure 6.1 for easy comparison.

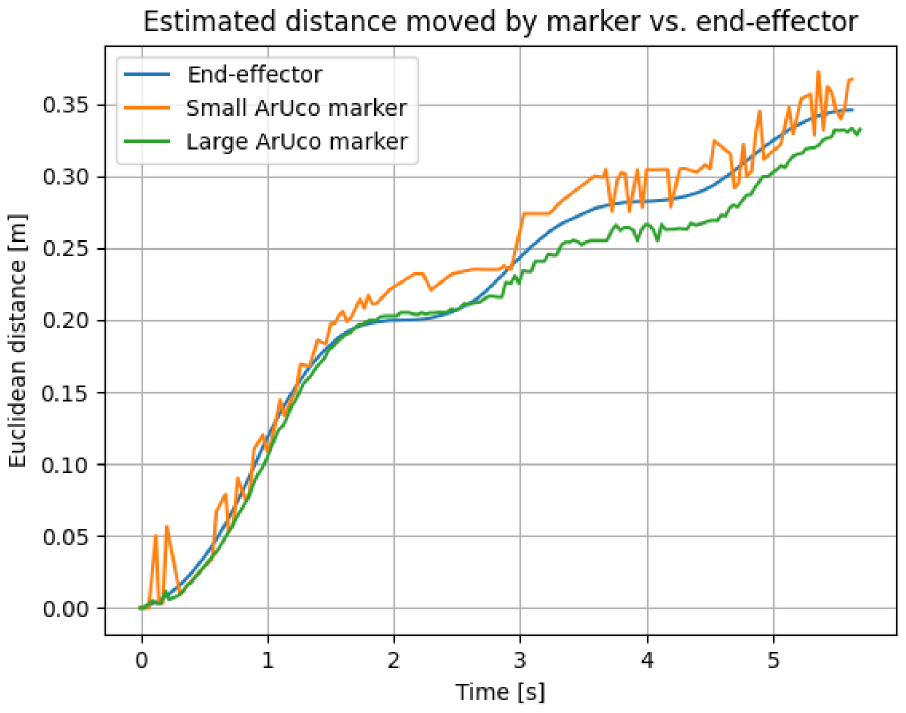


Figure 6.1: Estimated Euclidean distance moved by the ArUco marker in the camera frame plotted for each marker size together with Euclidean distance moved by robot end-effector in robot base frame.

For a movement of approximately 35cm in space, the results indicate an error of a few centimeters for both marker sizes, with the larger marker being less noisy. Meanwhile, the detection success rates for the two sizes denoted r_{big} and r_{small} respectively, were computed as

$$r_{big} = \frac{172}{172} = 1.00 \tag{6.2a}$$

$$r_{small} = \frac{112}{171} \approx 0.65, \tag{6.2b}$$

meaning that the big marker was detected in all 172 frames, while the small marker was detected in 112 of the total 171 frames.

Note that several potential error sources are involved, including inaccuracies in robot trajectory as received from the robot interface, inaccuracies in the estimated intrinsic camera parameters, and the possibility that the marker was not completely stationary during the experiment. A stationary marker is a crucial assumption because it allows the estimated movement of the marker to be compared with the movement of the robot end-effector. Quantifying these error sources is not of interest because the results are not evaluated quantitatively.

Based on the findings that the larger marker produced less noise and had a considerably higher detection success rate, it was decided to move forward with the $10cm$ x $10cm$ marker in the visual servo system.

Data processing and plotting were performed using Python, and the script is available on GitHub (Hagestuen, 2024*a*).

6.1.1.3 Methodology

A visual servo with an eye-in-hand scheme using ArUco markers was implemented in this experiment. The desired behavior of the visual servo can be summarized as:

1. Register position offset from camera to marker when the ArUco marker is detected.
2. Keep the offset constant by tracking marker movement in real-time and controlling the end-effector.
3. When the marker leaves sight, halt the robot.
4. Upon new detection of the marker, go back to point 1.

A Kalman filter was used to estimate the position of the marker. In addition to serving as a filter for the position estimates, it also provides estimates for the velocities $\dot{x}$, $\dot{y}$ and $\dot{z}$. As such, the state vector $\mathbf{x}$ is given as

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}, \tag{6.3}$$

with $\hat{\mathbf{x}}$ being the estimate of this state vector. Meanwhile, the measurement $\mathbf{y}$ used in the correction step of the Kalman filter is given as

$$\mathbf{y} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \tag{6.4}$$

where the measured coordinates x, y, and z are the coordinates of the ArUco marker acquired by solving the PnP problem in each frame in the video stream. Consequently, the measurement matrix $\mathbf{C}$ is

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{6.5}$$

Coordinates in the measurement and estimate were transformed to the robot base frame, i.e., the filter is implemented in the robot base frame. This was made possible by reading the robot pose and computing the base-camera transformation matrix $\mathbf{T}_{bc} = \mathbf{T}_{bh}\mathbf{T}_{hc}$ real-time, where:

- $\mathbf{T}_{bc}$ is the base-camera transformation matrix, used to transform points $\mathbf{x}^c$ in the camera frame to points $\mathbf{x}^b$ in the base frame.

- $\mathbf{T}_{bh}$ is the base-hand transformation matrix computed in real time as a function of the current robot pose.

- $\mathbf{T}_{hc}$ is the hand-camera (hand-eye) transformation matrix estimated through the extrinsic calibration procedure.

The model used in the Kalman filter is a white noise process in velocity, implying a random walk process in position. As such, the discrete $\mathbf{A}_d$ matrix in the model is given as

$$\mathbf{A}_d = \begin{bmatrix} 1 & 0 & 0 & \mathrm{dt} & 0 & 0 \\ 0 & 1 & 0 & 0 & \mathrm{dt} & 0 \\ 0 & 0 & 1 & 0 & 0 & \mathrm{dt} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{6.6}$$

with dt being the sampling time $dt = 1/f$ where $f = 30Hz$. The Kalman filter was tuned through simulating movements that the filter was expected to capture; noisy sine-waves were used in the measurement $\mathbf{y}$, and the process and measurement noise $\mathbf{Q}$ and $\mathbf{R}$ were tuned until the position and velocity estimates displayed acceptably low time-lag and noise simultaneously. The advantage of tuning the filter by simulation is that comparing the estimate to the true value is possible. Still, a potential drawback is that the filter is not tuned on real movements, which might differ vastly from the generated sine wave.

Errors in the tracking offset between the camera and marker were removed using a servo controller. Here, a built-in servo method from the RTDE API taking the desired end-effector position as an argument was called at each time step in the controller. This method offered no configuration concerning speed or acceleration. Therefore, a choice was made to wrap it in a more sophisticated custom method producing intermediate end-effector positions as a function of the desired aggressiveness of the servoing, which were passed to the built-in method.

The top-level algorithm for this ArUco marker-guided visual servo is seen in algorithm 6.

The algorithm describes how the desired behavior can be implemented. Note that a tracking state was incorporated in the algorithm; it was used to decide when to re-initialize the Kalman Filter and register a new desired tracking offset. This algorithm was implemented in a C++ project in the GitHub repository (Hagestuen, 2023*b*) developed during the summer internship with SINTEF Manufacturing.

6.1.1.4 Results

The results are presented as a YouTube video displaying the visual servo in operation; click **here** to access the video.

The saved data from the experiment also includes time series for tracking error and tool speed. Still, they are not included here because they are more relevant to discussions on the controller performance rather than the actual ArUco-tracking. Plots for estimated positions and velocities are also not included, as it was decided that they provide limited insights to the discussion. The main reasons for this are that ArUco marker tracking has already been discussed in the specialization project remarks, and this test was mainly meant to highlight the visual servo use-case of ArUco markers.

Algorithm 6: Visual robot servo guided by an ArUco marker

```
Data: Extrinsic camera parameters, intrinsic camera parameters, robot IP address,
      Kalman Filter model and process and measurement noise matrices, ArUco marker
      dimensions, and type
Result: Perform visual servoing by tracking an ArUco marker
Load hand-eye transformation matrix from file;
Load camera matrix and distortion coefficients from file;
Connect to robot on IP address;
Initialize color stream for Intel RealSense camera;
Configure ArUco marker detector with marker dimensions and type;
Set Tracking state to SEARCHING;
while Perform visual servoing do
    Capture the next frame from the camera;
    Detect ArUco marker in captured frame;
    if ArUco marker detected then
        Estimate marker coordinates by solving PnP-problem using intrinsic parameters;
        Receive robot pose and compute base-hand transformation matrix;
        Transform marker coordinates to robot base frame;
        switch Tracker state do
            case SEARCHING do
                Initialize Kalman Filter at marker coordinates;
                Set new desired tracking offset;
                Set Tracker state to TRACKING;
                break;
            case TRACKING do
                Kalman Filter correction step using marker coordinates;
                Kalman Filter prediction step using random walk process;
                Compute control input to minimize desired tracking offset;
                Execute control command;
                break;
    else
        switch Tracker state do
            case SEARCHING do
                break;
            case TRACKING do
                Halt robot;
                Set Tracker state to SEARCHING;
                break;
Close the camera and disconnect the robot;
```

6.1.1.5 Discussion

The video displays that the desired behavior is indeed achieved, indicating that ArUco marker tracking combined with a Kalman Filter for position estimation is feasible for visual servo use cases and environments that can guarantee great marker visibility. As mentioned, such use cases include lift-by-guiding tasks for improved ergonomics and part inspection. Another potential use case is testing fixing and mounting points on parts during rapid movements by selecting an aggressive controller. To reduce the dependency on the hand-held surface, one could attach a small, stiff marker to the palm and/or back of the work gloves and perform visual servoing tasks with free hands. Such a strategy is thought to have a lower detection success rate due to smaller marker size, increased wear on the marker, and possible occlusion from fingers.

The implemented scheme can also extend to tolerate failed marker detection in a given number of consecutive frames before changing the tracking state back to *searching*. This would improve the system's feasibility in outdoor environments, such as on a construction site, while allowing smaller markers to be used, potentially also at more considerable distances from the camera. Upon failed detection, the prediction step in the Kalman Filter could be called without the subsequent correction step; a strategy often referred to as *dead reckoning*. This technique also serves as a countermeasure to the risk of a reduced detection success rate if ArUco markers are attached to work gloves. Additionally, the technique opens up the possibility of using other detection methods thought to be more unreliable, like hand detection using machine learning libraries such as MediaPipe. The upcoming experiments will discuss this possibility more extensively.

As a final note on this experiment, it is mentioned that having speed estimates of the tracked object by applying a Kalman Filter opens up the possibility of enhancing safety systems; increased velocity of the operator's hand toward dangerous machinery could increase the distance from the machine defined as hazardous because the hand can cover a greater distance while the machine is shutting down.

6.1.2 Experiment 2 - Hand detection in the workspace

Disclaimer: this experiment (Section 6.1.2) is a derivative of work from the specialization project (Hagestuen, 2023*a*).

6.1.2.1 Introduction

An essential part of this master's thesis was investigating hand-tracking techniques in visual servo systems. Therefore, it was interesting to see if hand detection using MediaPipe is feasible in typical work situations, which is the motivation for conducting this experiment. In the workspace, the conditions can be unfavorable, with the operator holding tools, wearing gloves, and having occluded hands.

Research on equipment-free detection opens up more use cases than earlier, where the operator depends on holding an ArUco marker. In particular, this is a crucial step in seeing if this technique for hand tracking is feasible for visual servo systems.

6.1.2.2 Methodology

A series of images was captured in work situations where the operator might be wearing gloves and holding tools. Note that all images in the series contain at least one hand. The robot was oriented so that the Intel RealSense D455 mounted on the end-effector was located above the working table and facing down.

After capturing a series of images, it was attempted to detect hands in the images using MediaPipe with the same confidence threshold for the entire series. The selected confidence threshold was 0.05, considerably lower than the default value of 0.5. Such a low value was chosen to increase the number of successful hand detections in the image series, exploring the technology in more challenging scenarios. The samples providing the most insight are presented in the results; these could be samples where MediaPipe was able to detect hands in challenging images or vice versa.

The entire image series with the corresponding detection algorithm looping over all images is available in the GitHub repository (Hagestuen, 2024*a*).

6.1.2.3 Results

Firstly, a subset of images was captured without wearing work gloves. A highlight from this test was that hands were successfully detected when the operator was holding a hammer and a screwdriver, as seen in Figure 6.2. Generally, wearing no gloves yielded successful hand detection in most images in the subset.

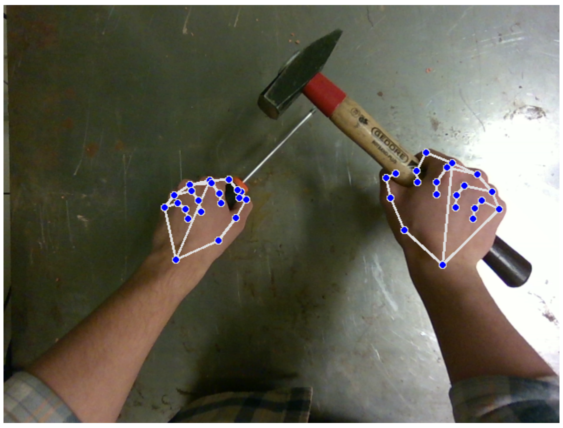


Figure 6.2: Successful hand detection when the worker was holding a hammer and a screwdriver without wearing gloves.

Next, a subset of images was captured when wearing orange gloves. This yielded varying results, as seen in Figure 6.3 and Figure 6.4; the hand was detected in just one of the two quite similar images. A screwdriver is present in the image where the detection failed.

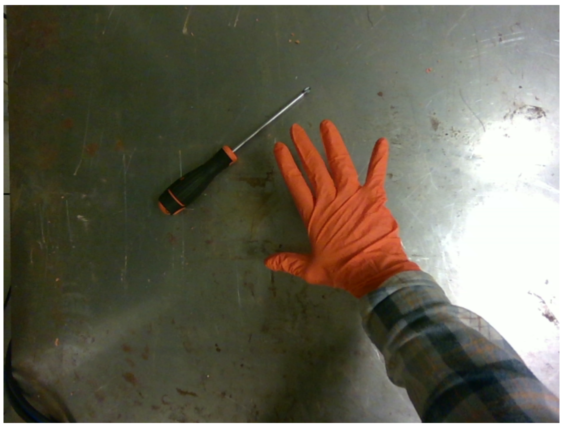


Figure 6.3: Failed hand detection when wearing orange gloves and holding no items. A screwdriver was present on the work table.

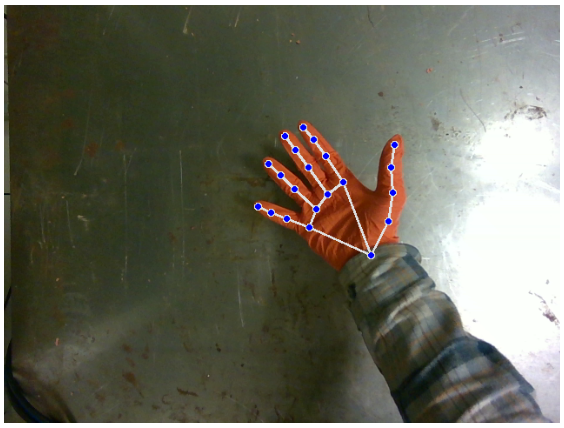


Figure 6.4: Successful hand detection when wearing orange gloves and holding no items.

Detection failed when holding a hammer, partially blocking the view of the hand; see Figure 6.5. However, the detection was successful when holding a brick sharing similar color tones with the glove, as seen in Figure 6.6.

Next, a subset of images was captured while wearing transparent disposable gloves. The hand was detected in most of these images, like when holding a screwdriver, as seen in Figure 6.7. The detection was also successful when holding a hammer in Figure 6.8. However, some landmarks are out of position; the index finger was wrapped behind the handle with the thumb and not stretched along the hammer, as indicated by the landmarks.

Furthermore, a subset of images was captured wearing black gloves in various situations, such as when holding tools or a brick. Detection failed in all this subset's images, even with free hands, such as in Figure 6.9.

Ultimately, a subset was captured when wearing oversized, white gloves. This yielded limited detection capabilities. The detection was successful when holding no items in a stretched-out pose, as seen in Figure 6.10, but it failed in Figure 6.11 when holding a hammer, despite the hand being quite stretched and mostly visible.

Note that the results are only a selection of the images in the series, which contains 60 images.

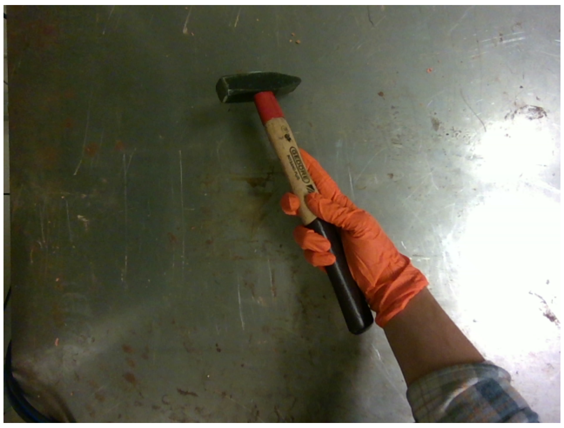


Figure 6.5: Failed hand detection when wearing an orange glove and holding a hammer.

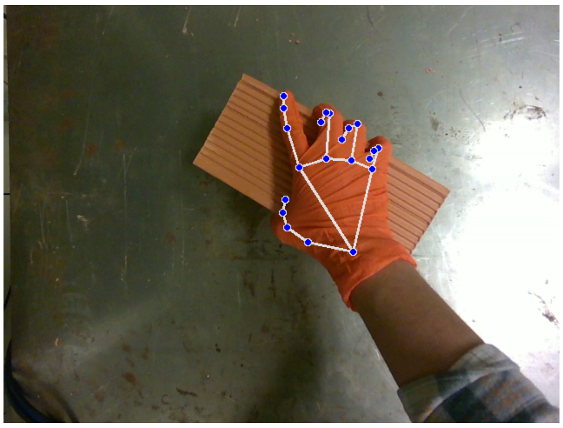


Figure 6.6: Successful hand detection when wearing orange gloves and holding a brick.

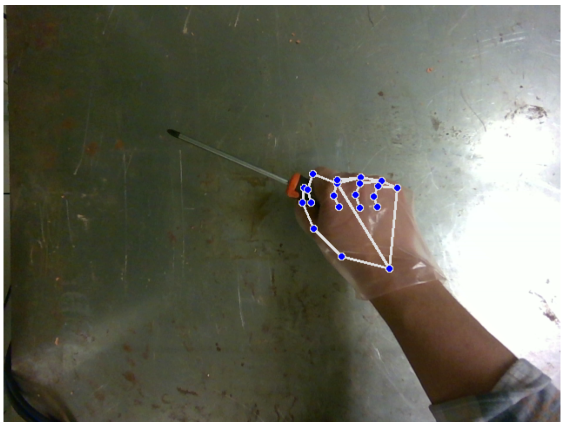


Figure 6.7: Successful hand detection when wearing a transparent glove and holding a screwdriver.

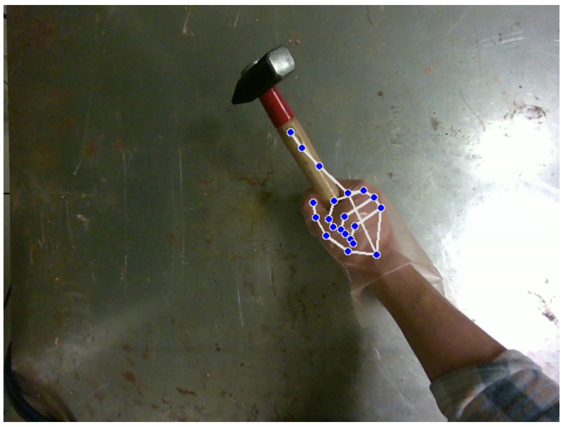


Figure 6.8: Successful hand detection when wearing a transparent glove and holding a hammer. However, the index finger landmarks were out of position.

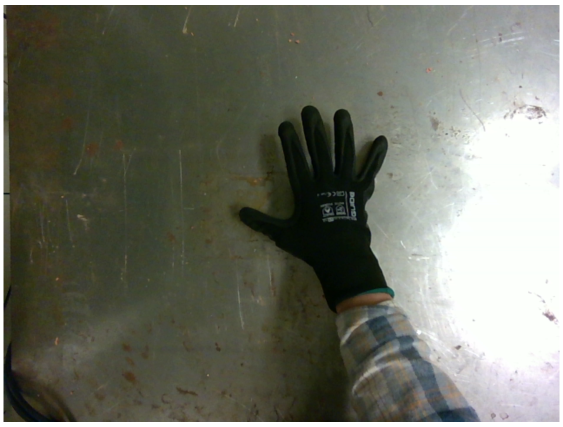


Figure 6.9: Failed hand detection when wearing black gloves and holding no items.

For the curious reader, it is re-emphasized that a script looping over the entire series to perform hand detection is available on GitHub (Hagestuen, 2024*a*).

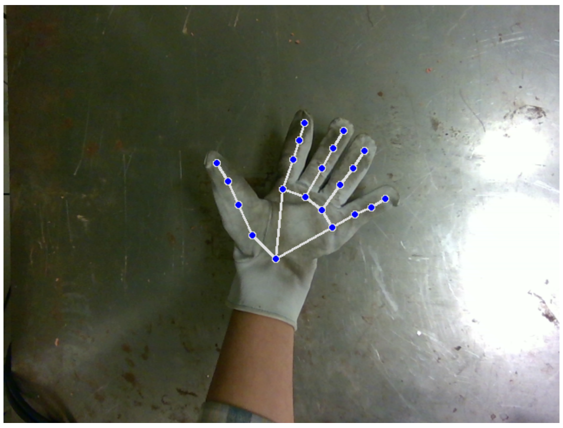


Figure 6.10: Successful hand detection when wearing white gloves and holding no items.

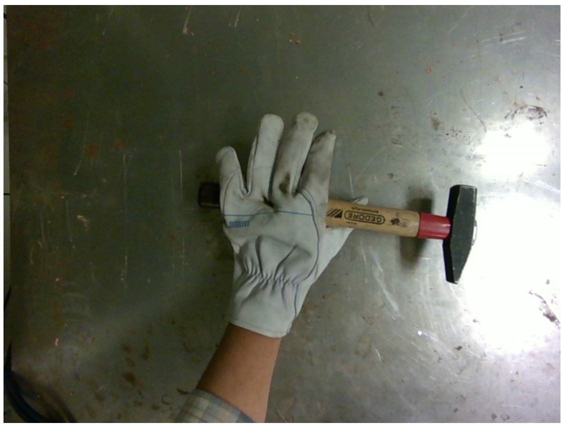


Figure 6.11: Failed hand detection when wearing white gloves and holding a hammer.

6.1.2.4 Discussion

The hand detection algorithm in MediaPipe employs a two-step process: a palm detector and a hand landmark model. Initially, the palm detector identifies a bounding box around the hand, leveraging the relative simplicity of detecting palms compared to entire hands with articulated fingers (Zhang et al., 2020). Next, the hand landmark model predicts the hand skeleton. As described by Zhang et al. (2020) the model is trained using three datasets. Firstly, it is trained on an *in-the-wild dataset*, which includes images of hands with varied backgrounds but simple hand articulations. Secondly, the model is trained on an in-house collected *gesture dataset*, featuring a wide range of hand gestures, but with limited background variation. Lastly, the model is trained on a *synthetic dataset* employed to better cover all possible hand poses. In contrast, the palm detection model is trained solely on the in-the-wild dataset.

The limited background variation in the gesture dataset may contribute to the poor performance in detecting black gloves. Although the example shown in Figure 6.9 is not a complex hand articulation, it is possible that other factors, such as the color contrast between the hand and the sleeve, negatively impact the model performance. Sufficient contrast between the hand/glove and the background is thought to be important for edge detection, despite successful detection in Figure 6.6 where the glove and brick share color tones and the hand is partially occluded. Additionally, the shape modification due to wearing gloves varies; for instance, the orange glove, being tighter than the black glove, maintains the hand's shape better. However, the looser glove in Figure 6.11 was still detected despite its oversized fit. Further, other objects within the image can potentially disrupt the palm detector by influencing the search area. For example, the presence of a screwdriver in Figure 6.3 could interfere with hand detection if it falls within the search region.

In summary, despite the information about the model architecture and training datasets provided by Zhang et al. (2020), several design uncertainties remain. Therefore, the effectiveness of the hand detection solution in challenging environments can best be evaluated based on its results.

The experiment yields varied results based on whether or not the operator is wearing gloves or holding tools. Because of this, hand detection using MediaPipe has limited value for a hand-controlled visual servo system employed in environments where it is natural for the worker to be wearing gloves or holding tools. On the contrary, results were promising when wearing orange gloves, suggesting that the correct selection of gloves can significantly improve the detection capabilities. However, this solves the problem from the wrong end; instead, a machine learning model should be trained on data made explicitly for construction and manufacturing settings, including image series of hands wearing various work gloves. Therefore, machine learning as a general concept for hand detection in visual servoing might still be feasible.

A key takeaway from this experiment is that hand detection using MediaPipe yields great results in controlled environments, particularly when not wearing gloves. Therefore, it is a feasible technology for visual servo systems operating in such environments and can be seen as a significant improvement in user-friendliness compared to using an ArUco marker to guide the robot. On the other hand, it has been uncovered that MediaPipe faces challenges when performing hand detection in complex environments. Despite its limited value in visual servo systems under such conditions, MediaPipe can still be useful for other applications within construction and manufacturing. For example, it can enhance safety by implementing a system similar to the "lawn-mower philosophy," where both hands must be placed on the handle for the machine to operate. Similarly, a system can be implemented in the workspace where two hands must be present in the image before starting a dangerous process nearby, such as a mill or lathe. Next, as indicated in Figure 6.2, handedness is determined as part of the detection algorithm. Handedness can potentially be used as a command to the collaborative robot; if the system is configured with the handedness of the worker, the presence of the opposite hand could be interpreted as a command, such as moving on to a new task. Understanding the intention of the worker by recognizing whenever the hand grasps new tools or items could also serve as a communication channel for human-robot collaboration. In general, if the robot can separate between tasks, efficiency can be improved by reducing dependency on teach pendants, buttons, or similar.

6.1.3 Experiment 3 - Visual servo using MediaPipe and RealSense for hand tracking

6.1.3.1 Introduction

Up until now, a visual servo scheme using an ArUco marker was displayed in Section 6.1.1, and hand detection using MediaPipe was demonstrated in Section 6.1.2. As such, a natural next step was to investigate the feasibility of hand tracking using MediaPipe combined with the Intel RealSense depth camera in a visual servo system. This experiment aims to implement such a scheme, discuss implementation challenges, and investigate its feasibility for applications in the manufacturing space.

As mentioned, Section 6.1.1 displaying a visual servo scheme using an ArUco marker for tracking was conducted as part of the summer internship with SINTEF Manufacturing. The scheme was implemented as a CMake project with C++ as the programming language. On the contrary, when redoing the experiment with MediaPipe, everything is implemented from scratch in Python. This decision was made because of the desire to stick with Python for all experiments in this master thesis. Additionally, the adaptation from ArUco marker tracking to MediaPipe-based tracking required substantial changes to the code, regardless of the transition to Python. As such, all parts of this experiment have been conducted as part of the master thesis. Naturally, similar challenges were encountered when redoing the experiment with the new tracking method, arguably reducing the time spent conducting this experiment.

6.1.3.2 Methodology

This experiment implemented an eye-in-hand visual servo scheme utilizing MediaPipe and the Intel RealSense camera. The fundamental idea for realizing 3D hand tracking was to make a depth request using the camera on pixel coordinates where a hand landmark has been detected. The desired behavior of the system is similar to that of the previously implemented visual servo, but one significant change has to be made; detecting the operator's hand in the camera view is insufficient to initiate hand-following behavior, the hand palm must also face towards the camera. This constraint allowed the operator to start and stop the robot by simply turning the hand instead of completely removing it from the camera view, possibly causing a rapid robot movement.

Detecting the direction of the palm can be done in a few different ways. A simple strategy is to compare the x-coordinates of a landmark on the pinkie with a landmark on the thumb in the camera frame. Note that determining palm direction based on this strategy requires knowledge about the handedness, i.e., if a left or right hand is detected. This knowledge is available from MediaPipe: upon detecting a hand, the hand-solution in MediaPipe indicates the handedness and a corresponding confidence $0.5 \leq p \leq 1.0$ associated with this handedness. Note that p cannot be lower than 0.5, which means MediaPipe detected the opposite hand. This is because $p_{left} + p_{right} = 1.0$. A clear weakness in comparing the x-coordinates of the pinkie and the thumb for determining handedness is that holding either the hand or camera upside down will produce wrong results.

An alternative method to improve this issue is to evaluate a cross-product between two vectors defined on the hand. For instance, one could evaluate the sign of the cross product between the 2D vector $\vec{\mathbf{a}}$ from landmark 0 to 5 and the 2D vector $\vec{\mathbf{b}}$ from landmark 0 to 17, see Figure 6.12. Pixel coordinates are used to construct these 2D vectors.

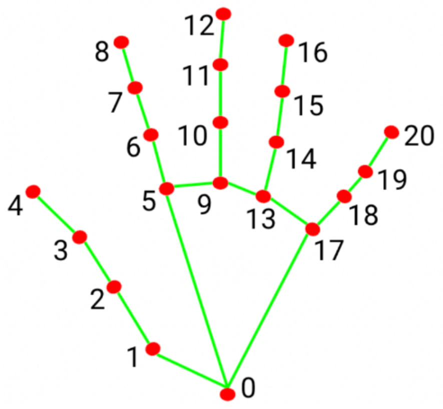


Figure 6.12: 21 hand landmarks produced by the MediaPipe Hand Landmarker task.
Source: Google (2023*a*), license: Creative Commons Attribution 4.0. The image has been cropped to include only visualization of hand landmarks; labels of landmarks have been removed.

The cross-product between two 2D vectors

$$\vec{\mathbf{a}} \times \vec{\mathbf{b}} = |\vec{\mathbf{a}}||\vec{\mathbf{b}}|sin(\theta) = a_1 b_2 - a_2 b_1, \tag{6.7}$$

is a scalar, where θ is the angle between the vectors. As such, the sign of the cross-product changes with the sign of θ. This property was exploited to determine whether the palm is facing the camera or not; as the hand is turned around, the angle θ between $\vec{\mathbf{a}}$ and $\vec{\mathbf{b}}$ changes signs and indicates a change in hand direction. Coupling this with knowledge about handedness made it possible to determine palm direction. This strategy is not prone to errors when holding the hand or camera upside down.

In summary, the desired behavior of the visual servo scheme is as follows:

1. Register position offset from camera to hand when the hand is detected with palm facing towards the camera.
2. Keep the offset constant by tracking hand movement and controlling the robot accordingly in real time.
3. When the hand leaves sight, or the palm is facing away, halt the robot.
4. Upon new detection of the hand with the palm facing toward the camera, go back to point 1.

When implementing the visual servo, it was essential to avoid double commanding the robot, i.e., sending a new command before the robot has executed the previous one. This was particularly relevant because the controller used in this scheme was a servo function blocking the robot for a certain amount of time. Sending a new command during this time caused an error message to be raised and the robot to stop. Intuitively, one could block the robot for dt with $dt = 1/f$, where f is the frame rate of the video stream. However, this was not a robust strategy because of varying processing times between receiving a new frame from the video stream and a finished computation of the following control input. In particular, if the processing time was unusually long in a given iteration and a control action was initiated to block the robot for dt, problems occurred if the processing time in the next iteration was shorter, as the system computed a new control action before the previous one was completed. Varying processing times arose due to the nature of MediaPipe's hand detector model; if segmentation information of the prior frame could be used when detecting a hand in the next frame, the detector required lower processing time.

There are a few strategies to solve the timing issue: the most elegant approach would be to create a thread-safe command queue, which only executes the next command when the previous is completed. A more straightforward method is to block the robot for the remaining time of dt only after processing is completed. This is visualized in Figure 6.13.

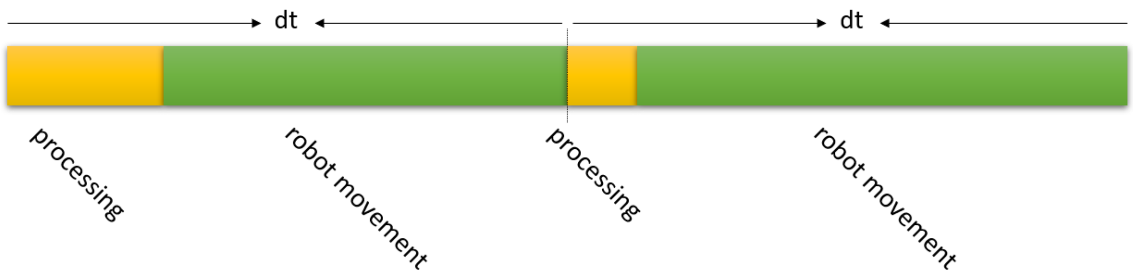


Figure 6.13: Timeline illustrating how the sample time dt is split between processing and robot movement. Relative lengths of processing and robot movement are not to scale.

The latter strategy was selected for simplicity. Note that a weakness concerning this strategy is that the robot is idle when processing a new frame, i.e., during the yellow part of the timeline in Figure 6.13. However, this was not noticeable when operating the robot, becoming apparent when seeing the system in action in the next section.

For improved safety, the system was configured with handedness to prevent unwanted intervention from the operator's other hand. This means that detecting the opposite hand was regarded as no hand detected and yielded no response from the robot. During initial testing, low gain values were used in the servo-controller to ensure correct system behavior before allowing more aggressive control. As the different components of the system seemed to work as intended, the gain was increased to achieve sufficient tracking during regular hand movements.

The top-level algorithm for this hand-guided visual servo scheme is seen in algorithm 7. The reader can fully explore the Python implementation on GitHub (Hagestuen, 2024*a*).

Note that depth was estimated solely based on the wrist landmark and used as the positional value for the entire hand. Preliminary testing showed that averaging the depth across all landmarks did not significantly reduce noise compared to using only the wrist landmarks. This conclusion was drawn during the specialization project, where a static-hand experiment was conducted to measure the noise in the depth estimate Hagestuen (2023*a*). In this experiment, a camera mounted on the robot's end-effector, which was stationary with mechanical brakes engaged, captured the data. The results consisted of two elements: the plot of the estimated depth over time and the corresponding standard deviation s, defined as

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}, \tag{6.8}$$

where:

- N is the number of observations in the sample,

Algorithm 7: Hand-guided visual robot servo

```
Data: Extrinsic camera parameters, intrinsic camera parameters, robot IP address,
      operator handedness
Result: Perform visual servoing by tracking hand
Load hand-eye transformation matrix from file;
Load camera matrix and distortion coefficients from file;
Connect to robot on IP address;
Initialize color and depth streams from the Intel RealSense camera;
Initialize MediaPipe hand detector object;
Set Tracking state to SEARCHING;
while Perform visual servoing do
    Capture the next frame from the camera;
    Align depth and color frames;
    Detect hand in the color frame and store 2D landmark coordinates;
    Estimate depth on wrist landmark using depth frame to acquire 3D hand coordinates;
    Determine palm direction using vector cross product;
    if Hand detected AND correct handedness AND palm facing camera then
        Receive robot pose and compute base-camera transformation matrix;
        Transform 3D hand coordinates to robot base frame;
        switch Tracker state do
            case SEARCHING do
                Set new desired tracking offset;
                Set Tracker state to TRACKING;
                break;
            case TRACKING do
                Compute control input to keep the desired tracking offset constant;
                Execute control command;
                break;
    else
        switch Tracker state do
            case SEARCHING do
                break;
            case TRACKING do
                Halt robot;
                Set Tracker state to SEARCHING;
                break;
Close the camera and disconnect the robot;
```

- x_i is each individual observation,
- $\overline{x}$ is the sample mean.

The wrist depth estimate and average depth estimate of all landmarks plotted against time can be seen in Figure 6.14.

Standard deviations $s_{wrist} = s_w$ and $s_{average} = s_{avg}$ calculated using Equation 6.8 were:

$$s_w = 0.00072m \tag{6.9a}$$
$$s_{avg} = 0.00051m, \tag{6.9b}$$

Although these results are uncertain due to potential error sources, such as the hand and robot not being perfectly stationary, it was decided to estimate the depth of the hand using only the

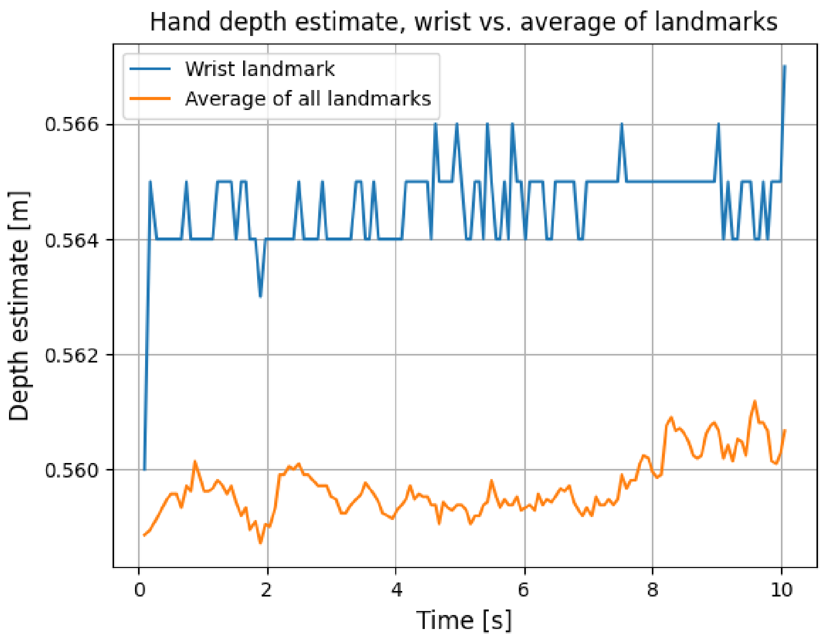


Figure 6.14: Wrist depth estimate and average of all 21 landmarks depth estimate plotted against time.

wrist in the visual servo system to reduce the number of landmarks processed in each iteration.

6.1.3.3 Results

The results are presented in video format displaying the system in action on YouTube. First, the experiment was conducted without wearing gloves; the video can be found **here**. Next, the experiment was conducted wearing the same orange gloves as in the previous experiment; click **here** to access the video. The videos show that the system demonstrates good tracking both with and without gloves. However, in the video with gloves, the system occasionally loses sight of the hand, and the robot halts momentarily before continuing to track with a new desired offset. This occurs despite the hand being visible to the camera.

6.1.3.4 Discussion

Hand tracking for visual servo using MediaPipe and Intel RealSense depth camera shows promising capabilities for gloveless hand tracking. However, the video showcasing tracking when wearing orange gloves reveals the main weakness concerning this method: when exposed to more challenging workspace situations, where the operator might be wearing gloves and holding tools, the tracking performance is reduced. It is clear from the video that the robot halts on a few occasions even though the hand is still in the camera view, with the palm facing the camera. The reason for this was undetermined; it could either be because the model detected the left hand, the wrong palm direction, or no hand at all.

An observation made during initial experiments when wearing the orange glove was that the model sometimes detected the opposite hand used to hold the robot teach pendant with higher detection confidence than the hand wearing the orange glove. This was even though the opposite hand was partially hidden behind the teach pendant, indicating that the hand detector was operating closer to the detection confidence limit.

This discussion should also mention that the experiment involving the orange glove yielded significantly better results than with other types of gloves, like the gloves worn in Section 6.1.2, which were a pair of white typical work gloves and a pair of black disposable gloves. Therefore, one can conclude that wearing gloves while operating the visual servo system will either reduce its performance or render it completely ineffective.

Despite its limitations, the device-less tracking showcased in this experiment introduces possibilities for deploying a visual servo system in new applications, expanding on the potential use cases already discussed in the first visual servo experiment (Section 6.1.1), where the reliance on an ArUco marker is evident. Firstly, the device-less tracking method is more appropriate in dynamic environments where the presence of the ArUco marker is impractical. Next, device-less hand tracking integrates better when the robot already needs to interact closely with humans or respond to human gestures, where hand tracking offers a more natural and user-friendly interface than fixed markers. Lastly, an argument for utilizing device-less tracking is simply removing the need to maintain a physical marker in the workspace, as it may be a substantial amount of time between every time the marker is used. Additionally, device-less tracking for visual servo can be used in the same applications as mentioned in the first visual servo experiment (Section 6.1.1), now offering improved user-friendliness by simply using the hand. Examples include lift-by-guiding tasks for enhanced ergonomics and testing fixing and mounting points on parts during rapid movements by selecting an aggressive controller.

6.2 Mapping natural language prompts to robotic tasks using LLMs and object detection

Now, shift the focus toward experiments and discussions revolving around how natural language can be mapped into robotic tasks while grounded in the real world using object detection. Figure 6.15 presents a module diagram for such a prompt-to-tasks robotic system. The system consists of the robot controller and camera modules interfacing with the two hardware components, the object detector for image processing, the LLM for requesting task sequences, and the user interface for communication with the operator. This module diagram should be used as support throughout the upcoming experiments to put components, such as the object detector and the natural language mapper, into the context of the broader prompt-to-tasks robotic system.

6.2.1 Experiment 4 - Sequencing robotic tasks based on natural language prompts

6.2.1.1 Introduction

An experiment was conducted to demonstrate the ability, or inability, of LLMs to leverage their extensive knowledge to produce a sequence of tasks based on a language prompt given by the user. In the context of the module diagram in Figure 6.15, the discussions in this experiment should be explicitly considered in relation to the "LLM" module within the broader prompt-to-tasks system. As mentioned in the literature study (Section 2.3.5), LLMs might be an enabling technology for *language instruction understanding* as defined by Liu and Zhang (2019). The main challenges mentioned in this paper include understanding language nuances, diverse contexts, complex relationships, and dealing with new information. These challenges represent areas where LLMs are thought to excel.

The thought scenario is a manipulator with an eye-in-hand camera setup serving as an assistant on a workstation. Tools and items are available on the workbench for the robot to equip, pick up, or hand over to the human operator. This scenario might resemble a collaborative robot in a production line that needs to hand items to an operator or a CNC machine requiring tool change between machining operations.

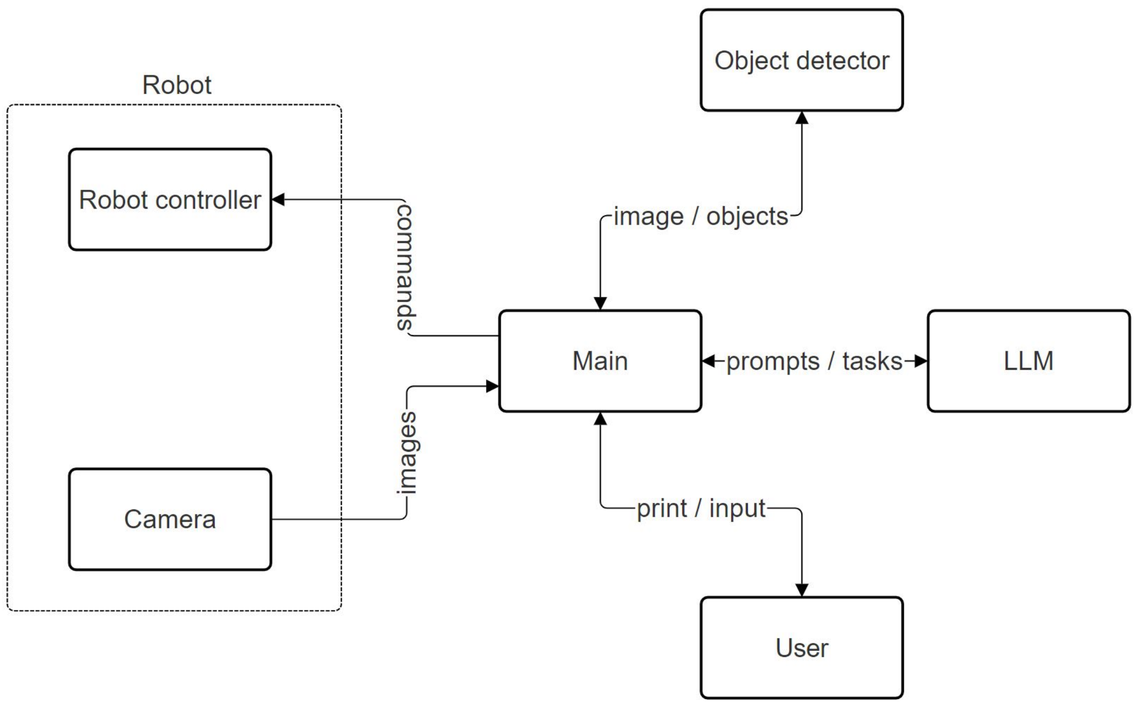


Figure 6.15: Module diagram for the prompt-to-tasks demonstrator. The diagram was made using SmartDraw.

6.2.1.2 Methodology

A set of available robotic actions is predefined and known to the LLM when served a language prompt. Typically, the language prompt represents a task that the operator requires the robot to perform, and the chosen sequence of actions should meet this requirement. Focusing only on the capabilities of the LLM, the set of available tools in this experiment was predefined instead of being based on visual input from the camera. Additionally, the selected tasks were not executed on a robot in a physical environment, as this experiment aims to highlight the capabilities of the LLM only. Integrating object detection in a physical environment is showcased in a later experiment.

The available actions were as follows:

A Pick up item 'X' from the workbench.

B Give item 'X' to the operator.

C Equip item 'X'.

D Move the end-effector to view the workbench from a new camera angle.

E Capture an image from the current camera angle.

Next, the pre-defined set of available tools was:

1. Hammer
2. Glue
3. Screwdriver
4. Caliper
5. Scissors

A *system message* was defined at the start of each session, providing the LLM with baseline instructions for the rest of the conversation. This system message was as follows:

- *"You are my assistant and are in control of a robot with a camera mounted on the end-effector. The camera can be used to scan for items on a workbench. The robot is able to perform the following actions: [A: Pick up item 'X' from the workbench. B: Give item 'X' to the operator. C: Equip item 'X'. D: Move the end-effector to view the workbench from a new camera angle. E: Capture an image from the current camera angle]. I, the operator, will tell you which items are visible on the workbench, and provide you with a task that I need done. Your job is to create a sequence of actions from the list above which fulfils the task. Your response should be formatted as a list of the selected actions, and which items from the workbench you decide to use (if relevant). If no actions apply, return 'None'."*

Next, a *user message* was passed containing a list of the currently visible items and the prompt from the user specifying the task that needs to be performed. Finally, a message from the language model termed the "assistant message," was received. This message contained a sequence of actions or was marked as "None" if no suitable actions were identified.

The prompts utilized in this experiment are summarized in Table 6.1.

Prompt	**Expected results**
Take a new look at the table to make sure no items are hidden. Next, give me something I can use to determine the diameter of this bolt.	The assistant should make sure the robot captures a new image of the table from a new angle. Additionally, the robot must pick up and hand the caliper to the operator to measure the bolt.
It is very slippery outside, can you help?	The given prompt is beyond the capabilities of the robot. As such the assistant should not suggest any actions and return "None".
Give me a tool for folding this piece of paper as many times as possible, I want to break the world record!	This prompt is somewhat ambiguous. While a hammer might be helpful for smoothing the paper between the folds, an acceptable outcome could also be "None." The assistant should refrain from selecting the scissors, despite its association with paper.

Table 6.1: LLM prompts and corresponding expected responses.

The LLMs used in this experiment were OpenAI's GPT model. In particular, three different models were tested:

- **GPT-3.5 Turbo**, specifically the *gpt-3.5-turbo* label which as of February 19th 2024 points to the flagship model *gpt-3.5-turbo-0125* in the GPT-3.5 Turbo family.
- **GPT-4**, specifically the *gpt-4*. This model family also includes the *gpt-4-32k*, offering a higher maximum quantity of text that the model can consider at any given moment while generating a response.
- **GPT-4 Turbo**, specifically the *gpt-4-turbo-preview* alias which as of February 19th 2024 points to the *gpt-4-0125-preview* model.

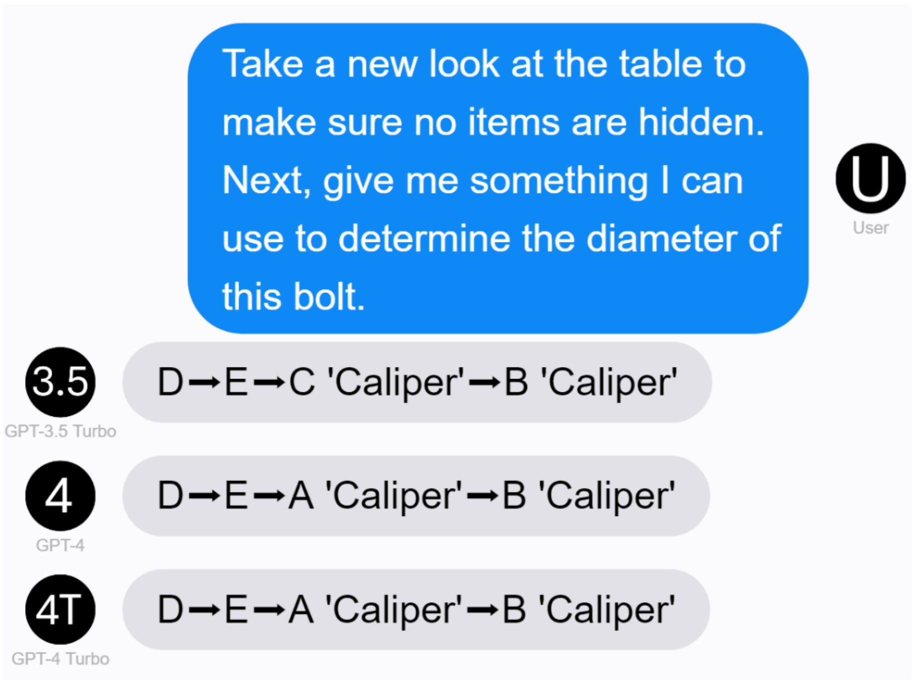
As stated by OpenAI, the GPT-4 Turbo model is the most powerful and is also offered at a lower price than the GPT-4 model as of February 2024, when this experiment was conducted.

Please note that the paid API service requires a unique key, which can be generated through an OpenAI account. The script developed for this experiment is open-sourced and can be seen on Github in the file *LLM.py* (Hagestuen, 2024*b*), but note the user must specify the API key because it is hidden. Alternatively, an environmental variable specifying the API key can be set on the local computer. The latter is considered the best practice for key-safety (OpenAI, 2024*a*) and is the approach used when conducting this experiment.

6.2.1.3 Results

This section covers the replies received by the LLM assistant for each user prompt and model. Note that, for the sake of convenience, the visualized chat conversations Figure 6.16, Figure 6.17 and Figure 6.18 do not include the system message with instructions and available actions which occurs before the user message. Therefore, it is important to consider that the system message was part of all conversations despite not being present in the visualization. Additionally, the list of items available on the workbench has also been excluded from the visualized conversation. This means the assistant was instructed about available actions and items before the messages seen in the visualized conversations.

Conversation 1
The results from the first prompt are seen in Figure 6.16. All three models produced a similar answer, but the GPT-3.5 Turbo model selected the *C - equip item 'Caliper'* action instead of *A - pick up item 'Caliper' from the workbench.*


Take a new look at the table to make sure no items are hidden. Next, give me something I can use to determine the diameter of this bolt.

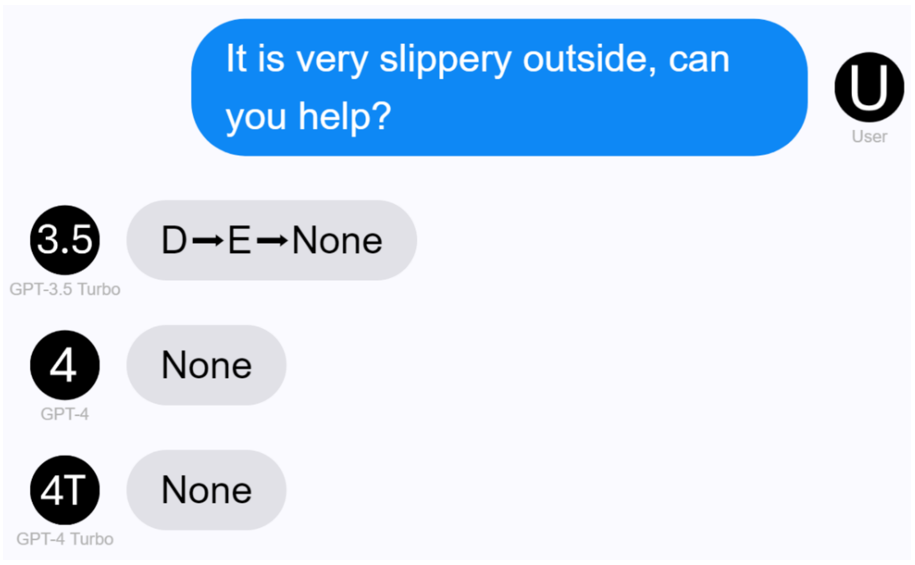
D→E→C 'Caliper'→B 'Caliper'

D→E→A 'Caliper'→B 'Caliper'

D→E→A 'Caliper'→B 'Caliper'

Figure 6.16: First prompt with corresponding answers from the assistant for each model. The conversation should be read in conjunction with the descriptions of the actions in the previous section (Section 6.2.1.2).
Chat visualization made using Chat Animator.

Conversation 2
Responses from the models to the second prompt are seen in Figure 6.17. While the GPT-4 and GPT-4 Turbo models selected *None*, the GPT-3.5 Turbo model resorted to capturing a new image of the worktable from a different camera angle.


It is very slippery outside, can you help?

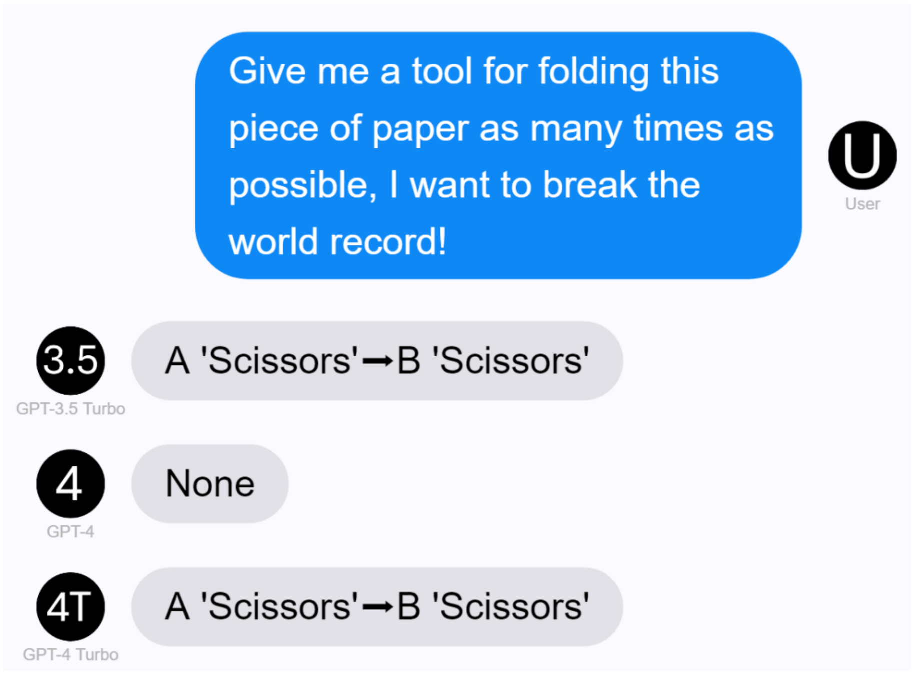
D→E→None

None

None

Figure 6.17: Second prompt with corresponding answers from the assistant for each model. The conversation should be read in conjunction with the descriptions of the actions in the previous section (Section 6.2.1.2).
Chat visualization made using Chat Animator.

Conversation 3
Responses from the model to the third prompt are seen in Figure 6.18. The GPT-3.5 Turbo and GPT-4 Turbo deemed the scissors a relevant object, picking it up and handing it to the operator.


Give me a tool for folding this piece of paper as many times as possible, I want to break the world record!

A 'Scissors'→B 'Scissors'

None

A 'Scissors'→B 'Scissors'

Figure 6.18: Third prompt with corresponding answers from the assistant for each model. The conversation should be read in conjunction with the descriptions of the actions in the previous section (Section 6.2.1.2).
Chat visualization made using Chat Animator.

6.2.1.4 Discussion

In the first prompt, all three models demonstrate good performance; however, observe the nuance between actions A, described as "pick up," and C, defined as "equip." Since the item is to be handed to the operator, the appropriate sequence of actions is to pick up the caliper and give it to the operator. As such, one can argue that the GPT-4 and GPT-4 Turbo models perform best here.

In the second prompt, the GPT-3.5 Turbo provided a sophisticated response by searching for new items that could contribute to problem-solving. However, since there was no interaction with a physical environment, and the list of available items remained static, the subsequent action selected was "None" as no new items were detected.

Regarding the third prompt, there is a clear association between paper and scissors, but it is unlikely that scissors would help fold a piece of paper as many times as possible. As such, this is an example of a conversation where the LLM falls short for the GPT-3.5 Turbo and GPT-4 Turbo models. Meanwhile, the GPT-4 model avoids this wrong association and performs well by producing the expected response.

It was noted that minor adjustments to the system message can significantly change the outcome of the conversations. For instance, modifying action E slightly (from *capture an image from the current camera angle* to *capture image*) alters the result of the initial conversation when employing the GPT-3.5 Turbo model. Specifically, the sequence of actions generated still involved moving the end-effector to a new camera angle, but it omitted the actual execution of action E, which captures the image. Additionally, it was attempted to include an example of a user-assistant conversation in the system message to demonstrate the desired behavior of the LLM, but this tended to bias the rest of the conversation towards the particular example. In conclusion, properly formulating the system message is important and highlights the value of prompt engineering.

In summary, this experiment has demonstrated that LLMs in the GPT family can map a text prompt to a sequence of actions given a set of available tools. However, based on the example conversations, none of the three models subject to testing have shown superior capabilities compared to the others.

On a final note, the *cumulative token problem* as addressed by hello9 (2023) on the OpenAI developer forum is mentioned as a possible limitation of utilizing an API to the GPT family for selecting robotic actions. As the conversation gets longer, the number of tokens required per new message increases greatly if the user provides the entire conversation history to the LLM in every message. This means that in scenarios where the assistant needs further instructions or feedback from the operator is incorporated and a more extended conversation occurs, the number of tokens used will increase significantly. However, the prices would have to increase by orders of magnitude from February 2024 levels before a typical conversation exceeds a few cents.

6.2.2 Experiment 5 - Re-training MediaPipe's object detection model for new object categories

6.2.2.1 Introduction

Object detection plays a vital role in collecting information about the robot's environment in modern robotic systems. This also applies to prompt-to-task robotic systems, where the robot must be rooted in the real world through object detection to generate an appropriate set of available actions. Depending on the application, existing models for object detection might not support the object categories necessary for the robot to be able to detect. As such, the following experiment was devoted to re-training MediaPipe's object detection model for new object categories using MediaPipe Model Maker.

6.2.2.2 Methodology

Inspired by the strategy employed by Lin et al. (2014) when creating the COCO 2014 dataset, Flickr was used to collect images for re-training the MediaPipe object detection model. As mentioned, roughly 100 images per new object are required to replace the classification layers of the model with the new layers using MediaPipe Model Maker.

It was attempted to avoid re-training the model with iconic images of the object categories. As such, the search strings in Flickr were formulated as "*object+scene*" and "*object+object*" to collect non-iconic images of the objects. If this strategy produced insufficient images, the search string would be modified to include only the object, filtering out iconic images. Additionally, images in supported classes from the Open Images V7 dataset (Google, 2022*a*) would be used to fill in if necessary. In this case, image credits were given to the original contributor because the non-labeled version of images from the Open Images dataset were utilized.

The dataset only contained images with a Creative Commons license allowing derivatives and adaptations, facilitated through Flickr's search engine which supports filtering for various license types. After experimenting with some example searches, it was noted that more than 90% of images are uploaded under licenses other than Creative Commons licenses allowing derivatives and adaptations, meaning that a significant portion of images on Flickr was unavailable for this work. The reader can explore the entire dataset and corresponding image credits to all image contributors on GitHub (Hagestuen, 2024*b*).

A decision was made to re-train the model for three new object categories:

- Hammer
- Screwdriver
- Wrench

Adjustable wrenches were excluded, focusing only on fixed wrenches. This strategy limited the scope of the object class as much as possible, aligning with the recommendations by Person (2023) in the Google Developer's blog.

After collecting and labeling images of hammers, the model was re-trained, and qualitative testing was conducted on new images of hammers using the MediaPipe Demo web application, which facilitates quick testing of custom models using the computer's webcam. The aim was to research if moving forward with the other object categories was worth the time, as collecting and labeling data is a relatively time-intensive task. A key takeaway was that some of the images in the dataset might have been too challenging. Therefore, iconic images were included in the dataset when collecting images for the next object categories. For illustration, an example of iconic and non-iconic images can be seen in Figure 6.19 and Figure 6.20.

The following list includes searches conducted on Flickr to collect images of hammers for the dataset:

- hammer+tool
- hammer+wood
- hammer+work
- hammer+metal
- hammer+construction
- hammer+nail
- hammer+wrench
- hammer+screw

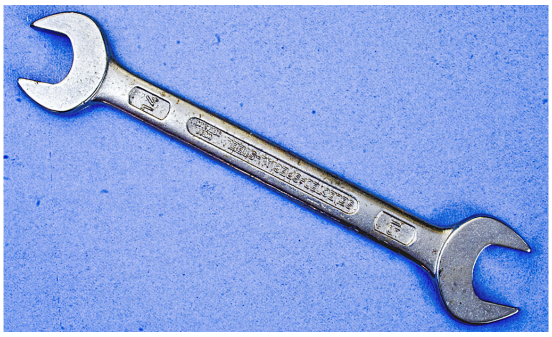


Figure 6.19: Iconic image of a wrench with a background of uniform color.
Image source: Tudor Barker via Flickr, license: CC BY-NC-SA 2.0 DEED.

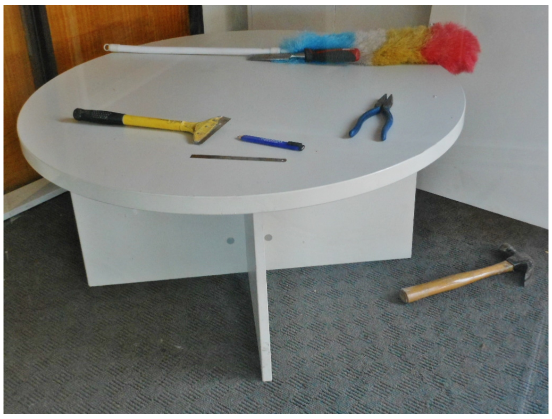


Figure 6.20: Non-iconic image of a hammer.
Image source: Michael Coghlan via Flickr, license: CC BY-SA 2.0 DEED.

- hammer+claw
- hammer

Additionally, the OpenImages dataset provided some images of hammers.

Next, the following list provides an overview of searches conducted on Flickr to collect images of screwdrivers:

- screwdriver+tool
- screwdriver+work
- screw+driver
- screwdriver+wood
- screwdriver+hammer
- screwdriver+screw
- screwdriver+construction
- screwdriver+mech
- screwdriver+garage
- screwdriver+home
- screwdriver+assembly
- screwdriver+assembling
- screwdriver

Finally, the equivalent overview of searches for images of wrenches is as follows:

- wrench
- spanner
- wrench+tool
- wrench+crescent

- wrench+screwdriver
- wrench+home
- wrench+mech
- wrench+garage
- wrench+assembly
- wrench+assembling

In addition to the images found in the searches listed above, additional images emerged when looking at the profiles of the image creators. For some image creators, this strategy discovered several useful images not found in the original search.

Next, the collected images were annotated with bounding boxes using Label Studio, as MediaPipe Model Maker does not support segmentation functionality. Labeling data is an ambiguous task for two reasons. Firstly, some objects nearly resemble specific tools, e.g., the middle screwdriver in Figure 6.21, which was not labeled as a screwdriver in this dataset. Secondly, the bounding box in Label Studio can be placed at different angles, especially when the object to be marked is at the edge of the image or sticks out of the image. This issue arises from the requirement that the bounding box must remain within the image while encompassing as much of the object as possible. See Figure 6.22 and Figure 6.23 for two examples of how the bounding box can be oriented.

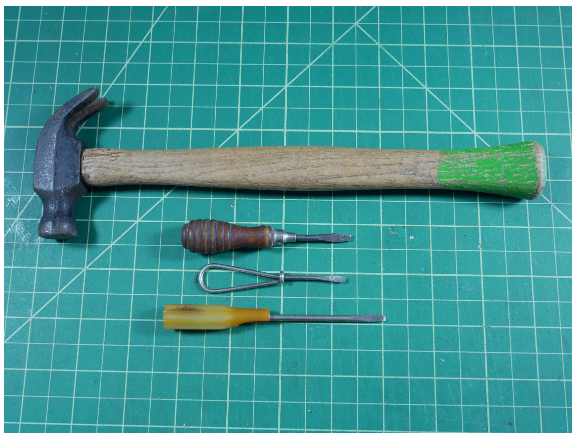


Figure 6.21: The middle screwdriver was not included in the dataset due to distinct visual differences.
Image source: el cajon yacht club via Flickr, license: CC BY 2.0 DEED.

Ambiguous orientation is problematic when evaluating on the dataset because the bounding box orientation in a detection might not coincide with the orientation of the annotation in the test split. However, the orientation of the bounding boxes is restricted by the COCO and PASCAL VOC formats accepted by MediaPipe Model Maker; these formats assume image-aligned bounding boxes, meaning that tilted bounding boxes are not supported. Using the COCO format as an example, the bounding box information is stored as $[x_{min}, y_{min}, width, height]$, where x_{min} and y_{min} are the coordinates of the top left corner of the bounding box. When interpreting this data, the bounding box is assumed to be aligned with the axis of the image with the coordinates (x_{min}, y_{min}) of the top left corner and with width and height as specified. Providing tilted bounding boxes produces wrongful results, and the correct placement of the bounding box in the earlier example is seen in Figure 6.24.

After all occurrences of objects in the dataset were labeled, the dataset was exported to COCO format. Using FiftyOne in Python, the dataset was separated into test, validation, and test splits using a 80/10/10 split. The Python script also made some necessary adjustments to the datasets to ensure they were in the format that MediaPipe Model Maker accepted. These adjustments include adjusting the JSON files so that the *background*-category had $ID = 0$, ensuring correct

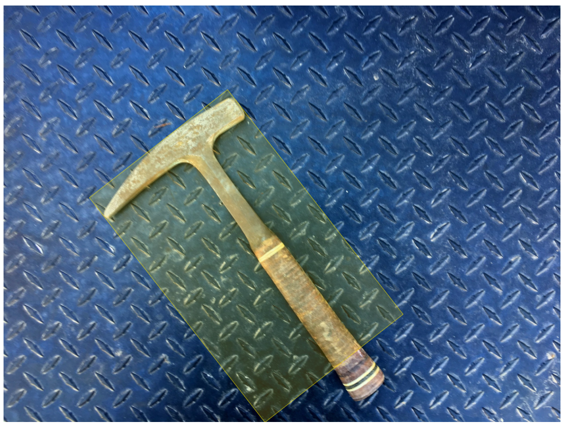


Figure 6.22: Bounding box placed parallel to the handle, arguably representing the natural and intuitive orientation. The bounding box reached the edge of the image and did not encapsulate the entire hammer.
Note: yellow color was used for better illustration in this image; the hammer object class was labeled with green bounding boxes in the rest of the dataset.
Image source: Alan Levine via Flickr, license: CC0 1.0 DEED. Annotated using Label Studio.

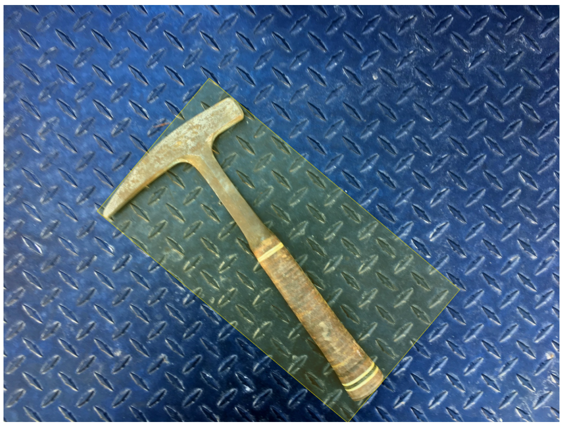


Figure 6.23: Bounding box tilted compared to the handle, allowing the entire hammer to be encapsulated while keeping the bounding box as small as possible. This example represents a possible ambiguity in the labeling process.
Note: yellow color was used for better illustration in this image; the hammer object class was labeled with green bounding boxes in the rest of the dataset.
Image source: Alan Levine via Flickr, license: CC0 1.0 DEED. Annotated using Label Studio.

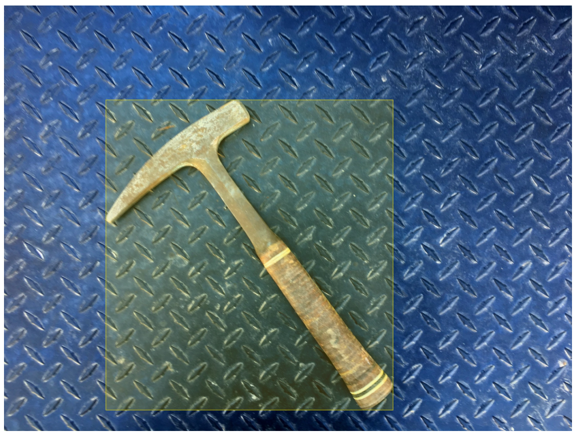


Figure 6.24: Bounding box aligned with image axes, which is the correct orientation when using the COCO or PASCAL VOC formats.
Note: yellow color was used for better illustration in this image; the hammer object class was labeled with blue bounding boxes in the rest of the dataset.
Image source: Alan Levine via Flickr, license: CC0 1.0 DEED. Annotated using Label Studio.

names for the images folder, and adding a license field. Refer to GitHub for fully exploring the script (Hagestuen, 2024*b*).

Next, the MediaPipe Model Maker module in Python was used to re-train the object detector for the new object categories. This step involved selecting hyperparameters and the desired model architecture. The selected model architecture *MobileNet MultiHW AVG I384* uses a 384x384 input image size, which was the highest resolution of the available architectures as of May 2024. Additionally, this architecture achieved the best performance out of the four available architectures when benchmarked on an android figurines dataset. On the downside, it suffered a CPU latency in the $200 - 300ms$ range, which was higher than the other models. This was something to consider when selecting the appropriate architecture for the application. For this experiment, latency was not an issue, and the MobileNet MultiHW AVG I384 architecture was considered the best choice (Google, 2023*c*).

With MediaPipe Model Maker, re-training the object detection model using the dataset developed in this thesis took roughly 20 minutes on a laptop with an AMD Ryzen 7 3700U processor and AMD Radeon Vega 10 GPU. Note that the time needed depends on several factors, such as the number of images, processing power, and hyperparameter epochs. As such, it is relatively easy to perform several iterations to improve model performance. As suggested by Person (2023), making incremental changes between each iteration is a recommended strategy for easily tracking the effect of the adjustments when aiming to enhance performance. Two key hyperparameters are learning rate and epochs, and adjustments to these can significantly affect the resulting model.

After realizing that re-training the model only required about 20 minutes, a brute force method was adopted in tuning the hyperparameters epochs and learning rate, i.e., the strategy explored all combinations within chosen ranges. For epochs, the candidates 10, 20, 30, 40, 50, and 60 were considered, while for learning rate, the values 0.01, 0.02, 0.04, 0.06, 0.10, 0.15, 0.20, 0.30, and 0.50 were examined. A Python script was developed to iterate through each combination overnight, conducting training and recording average precision and loss data for each combination in a text file for evaluation the following day. One should note that increasing the number of epochs without limits is not a good strategy. A suggested approach is to consider the epoch range at which the loss stabilizes to avoid overfitting the model to the training data.

To summarize the methodology employed for re-training the object detector using MediaPipe Model Maker, a step-by-step guide is provided:

1. **Decide on the object categories** based on the application. Simplify the model by limiting the number of classes if possible. Also, do not train for edge cases, as suggested by Person (2023). For example, do not include images where half the object is outside the frame. Finally, narrow down the classes by, e.g., avoiding images of sledgehammers in the hammer object category.

2. **Collect images** of the selected object categories using Flickr's search engine. Limit the search by selecting "Modification allowed" under the license filter. The images should be named after the object category, e.g. "hammer01", "hammer02", ..., "hammer99", "hammer100", "screwdriver01, screwdriver02", etc. Keep a text file updated with image credits in a list format, allowing easy association between the image and the entry in the contributions list. Aim for 100 images per object category.

3. **Label the images in Label Studio** using the *object detection with bounding boxes* template. List all desired object categories, with the default category "background" listed first. Import the images and start the process of labeling all occurrences. Ensure that bounding boxes are vertically oriented, as the COCO format does not support tilted boxes and will be wrongfully interpreted.

4. **Export the dataset** to COCO format. Name the .zip file "LS_export" and unpack it. Clone the Git repository to your local computer and place the exported dataset folder in the "object detection" folder in the Git repository.

5. **Split into train/validate/test** by running "split_dataset.py" in the repository, which randomly splits the dataset into three separate COCO datasets using FiftyOne. If a different split than 80/10/10 is desired, modify the script accordingly. A weakness in this step is that the random splits train/validate/test may contain an unbalanced amount of object categories *within each class*, as the random split only generates balanced splits based on the dataset as a whole. Therefore, functionality has been added to count occurrences of images for each object in each split and print it to the console so that one can re-run the script if an imbalance has occurred. This functionality relies on correct file names for the images in the dataset. Next, the script adds the CC BY-SA 4.0 DEED license to the exported datasets. This can be changed to a different license by adjusting the script, but keep in mind that images used in the dataset might be published under a *ShareAlike*-license, meaning that if you remix,

transform, or build upon the material, you must publish the work under the same license, or a license from the list (link) of compatible licenses (CreativeCommons, n.d.).

6. **Re-train the model** by running ”retrain.py” in the Git repository. Allow roughly 20 minutes per re-training. A recommendation is to execute this step on Ubuntu for easier installation of MediaPipe Model Maker, as mentioned in the experimental setup chapter. An essential part of this step is selecting the appropriate hyperparameters for re-training the model. The Python script calculates the validation loss and average precision using the validation dataset, prints them to the console, and exports the resulting model to a new folder named ”exported_model”.

7. **Prototype using MediaPipe Studio**, a web-based application for evaluating the exported model, which is uploaded as a TensorFlow Lite model file. One can evaluate new images not seen by the model, or a web camera can be used to qualitatively evaluate the model's performance.

8. **Make incremental changes** to the dataset, annotations, and hyperparameters and re-train the model. This allows for easier tracking of the effect of the adjustments. When the model yields satisfying performance, deploy the exported model to the application. Also, the model should be evaluated quantitatively using the test dataset not seen during the training procedure, allowing for a neutral model benchmark compared to other candidate models. Only minor adjustments are needed to the ”retrain.py” script to perform this evaluation on the test split instead of the evaluation split.

The step-by-step guide was developed using the following package and software versions:

- Python version 3.11.9
- FiftyOne version 0.23.7
- Label Studio version 1.11.0
- MediaPipe Model Maker version 0.2.1.3
- TensorFlow version 2.16.1

As a final note, all work related to re-training the model was tracked using Clockify to get an overview of how much time one must expect to devote to customize MediaPipe's object detection solution using MediaPipe Model Maker. Four categories, known as ”tasks” on Clockify, were defined in the project to track how much time was consumed on the various parts of the work:

- Collecting images.
- Labeling data.
- Export, formatting and splitting.
- Iterations and training.

6.2.2.3 Results

Quantitative results are presented using the average precision metric with combinations of values for the epochs and learning rate hyperparameters, calculated using the evaluation dataset. See Table 6.2 for an overview of the performance of each model.

Observe that the average precision was low across all combinations of hyperparameters. Although some combinations such as $epochs = 40$ and $learning\,rate = 0.20$ gave a higher AP of 0.0339, these values were magnitudes away from what would be considered satisfying.

	Learning Rate								
Epochs	**0.01**	**0.02**	**0.04**	**0.06**	**0.10**	**0.15**	**0.20**	**0.30**	**0.50**
10	0.0000	0.0003	0.0006	0.0007	0.0056	0.0030	0.0047	0.0057	0.0371
20	0.0008	0.0044	0.0011	0.0057	0.0237	0.0175	0.0115	0.0041	0.0068
30	0.0006	0.0006	0.0014	0.0026	0.0153	0.0032	0.0037	0.0064	0.0049
40	0.0008	0.0027	0.0023	0.0015	0.0051	0.0093	0.0339	0.0227	0.0027
50	0.0016	0.0027	0.0037	0.0075	0.0066	0.0102	0.0004	0.0122	0.0085
60	0.0022	0.0029	0.0014	0.0057	0.0084	0.0061	0.0169	0.0303	0.0156

Table 6.2: Average precision for different combinations of epochs and learning rates. The reader can find the exact train/validation/test splits by checking out the commit with hash 37d4a84c4919ccd1cfc7ea0b01e8426a6621b5dd in the GitHub repository (Hagestuen, 2024*b*).

Next, several models were trained only using images with instances of the wrench object category, as this part of the dataset included more iconic samples and was thought to have overall better quality. Table 6.3 provides an overview of average precision scores for different combinations of epochs and learning rates.

	Learning Rate								
Epochs	**0.01**	**0.02**	**0.04**	**0.06**	**0.10**	**0.15**	**0.20**	**0.30**	**0.50**
10	0.0030	0.0334	0.0860	0.1563	0.1747	0.2075	0.2016	0.1765	0.0603
20	0.0446	0.0382	0.1665	0.1749	0.2177	0.2553	0.1809	0.1901	0.2224
40	0.0684	0.1813	0.1642	0.1615	0.2307	0.2371	0.1701	0.1322	0.1762

Table 6.3: Average precision for different combinations of epochs and learning rates when trained using only images of the wrench object category. The reader can find the exact train/validation/test splits by checking out the commit with hash 33e31d88c944dc4ec73f270dc7c79096e0c89520 in the GitHub repository (Hagestuen, 2024*b*).

As seen in the overview, including only images of the wrench object category yielded higher AP values across all combinations, with $epochs = 20$ and $learning\,rate = 0.15$ having the highest AP. The upcoming section presenting qualitative results utilizes this combination of hyperparameters.

Before moving on to qualitative results, one should note that attempts were made to retrain the model using an adaptive learning rate. A cosine decay schedule was employed, with user-selected *cosine_decay_epochs* and *cosine_decay_alpha*, defining over how many epochs the cosine decrease happens, and to which final value the learning rate converges to as a fraction of the initial learning rate. Promising combinations of values for the epochs and learning rate parameters were used as candidates, and an adaptive learning rate scheme was employed without any noticeable consistent improvements in performance. Therefore, performance results from models trained with an adaptive learning rate scheme are not included.

Qualitative results are presented by showing model predictions on images from the test split. The best-performing models were used to generate these results from each of Table 6.2 and Table 6.3.

Figure 6.25, Figure 6.26 and Figure 6.27 show predictions on three new images using the model trained with $epochs = 40$ and $learning\,rate = 0.20$. As observed, the model consistently failed across all images by predicting the wrong label or not predicting any label, effectively defaulting to the "background" category.

Qualitative testing was also conducted using the wrench-only model, i.e., the model solely trained on images containing the wrench object category with $epochs = 20$ and $learning\,rate = 0.15$, on images from the test split. Three images were evaluated, see Figure 6.28, Figure 6.29 and Figure 6.30. This model performed substantially better than the model with three labels, with the only errors being one false positive and incorporating two wrenches in the same bounding box in the last image.

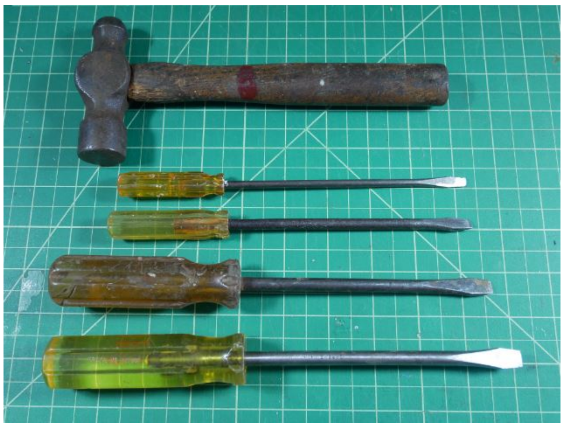


Figure 6.25: Hammer and screwdrivers wrongfully predicted as the default category. The prediction score threshold was set to 25%.
Image source: el cajon yacht club via Flickr, license: CC BY 2.0 DEED. Bounding box prediction was made using MediaPipe Studio.

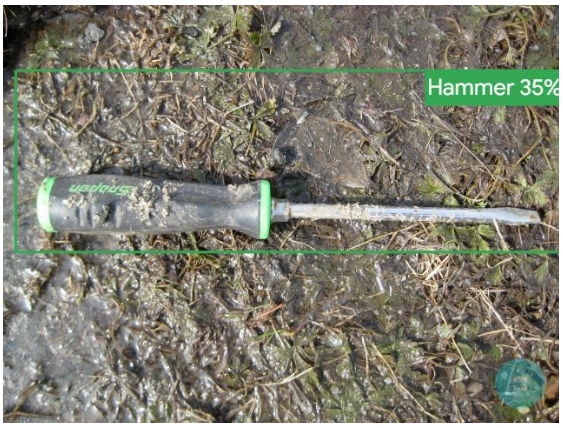


Figure 6.26: Screwdriver wrongfully predicted as a hammer with a 35% confidence score. The prediction score threshold was set to 25%.
Image source: Eugene Peretz via Flickr, license: CC BY-SA 2.0 DEED. Bounding box prediction was made using MediaPipe Studio.

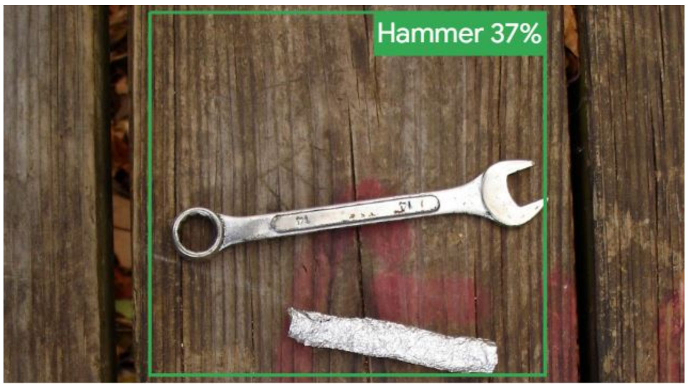


Figure 6.27: A wrench wrongfully predicted as a hammer with a 37% confidence score. The prediction score threshold was set to 25%.
Image source: Kevin via Flickr, license: CC BY-NC-SA 2.0 DEED. Bounding box prediction was made using MediaPipe Studio.

The total time spent re-training the model for the three new object categories—hammer, screwdriver, and wrench—was 68.69 hours, with the time distributed as follows:

- Collecting images: 14.92 hours.
- Labeling data: 13.00 hours.
- Export, formatting, and splitting: 9.61 hours. Most of these hours were devoted to writing the Python script for formatting and splitting the dataset. This script was developed to minimize manual and tedious work for similar projects in the future. With this script in place, future work can be streamlined by simply calling the script, significantly reducing the time and effort required.
- Iterations and training: 31.16 hours.

Note that this is an overview of *effective* working hours on customizing the object detector, meaning that time spent on related work like writing this report and reading papers on similar work are excluded. The same applies to other activities like lunch breaks and attending meetings.

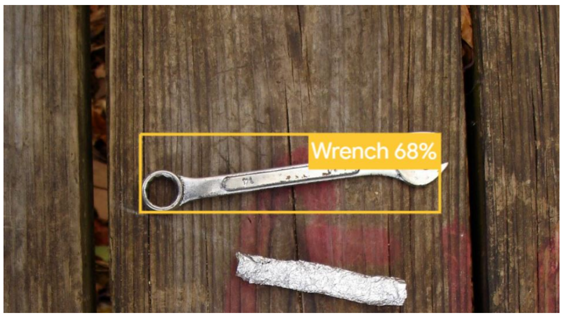


Figure 6.28: Correct wrench prediction with 68% confidence score on the same image where the three-label model wrongfully predicted a hammer. The prediction score threshold was set to 25%.
Image source: Kevin via Flickr, license: CC BY-NC-SA 2.0 DEED. Bounding box prediction was made using MediaPipe Studio.

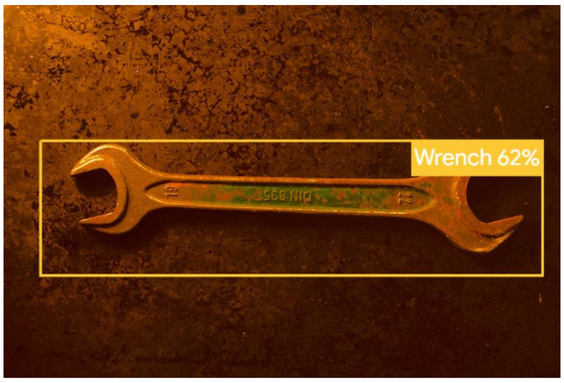


Figure 6.29: Correct wrench prediction with 62% confidence score. The prediction score threshold was set to 25%.
Image source: Alessandro via Flickr, license: CC BY-NC-SA 2.0 DEED. Bounding box prediction was made using MediaPipe Studio.

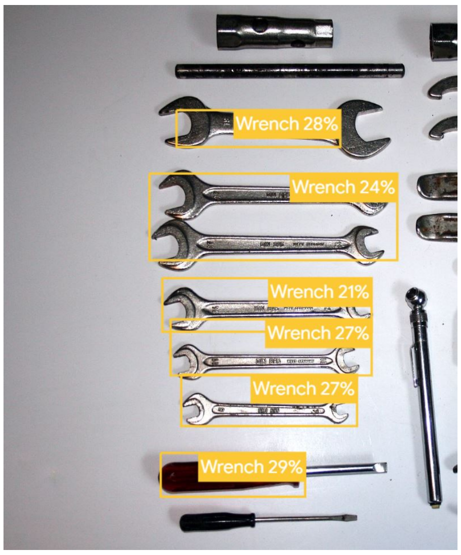


Figure 6.30: Correct wrench predictions, with a generally lower confidence score. One screwdriver is wrongfully predicted as a wrench. The prediction score threshold was set to 20%.
Image source: Noel Hankamer via Flickr, license: CC BY-NC-SA 2.0 DEED. Bounding box predictions were made using MediaPipe Studio.

6.2.2.4 Discussion

The discussion can be started by addressing the poor model performance after re-training for new object categories. When aiming for an AP close to 1.0, it is disappointing that the best-performing model, trained to recognize all three categories—hammer, screwdriver, and wrench—only yielded an AP of 0.0339. However, after employing the step-by-step method to re-train the MediaPipe object detection model, several insights have been gained that would be useful if the procedure were repeated to improve performance. Towards the end of this discussion section, a bullet point list summarizes these insights. Additionally, some issues are discussed in more detail before the summary.

An important issue is that most images used in the dataset have considerably higher resolution than the input image size of the model architecture (384x384). This aspect should have been considered more when collecting images for the dataset. Some images in the dataset were too challenging when downscaled to a 384x384 resolution, negatively impacting model performance.

Another issue is how the orientation of the objects in the training dataset affects the resulting model's performance. For example, the model might yield better results if all hammers in the dataset were oriented vertically, meaning all hammer instances would have the same orientation relative to their bounding boxes. However, such details are not available from the MediaPipe Model Maker documentation. Additionally, the resulting model must be able to detect hammers that are not oriented vertically to be useful in a practical setting, meaning such a discussion is of limited value when collecting the dataset because it could decrease the model's versatility. Instead, revealing such information would be more useful for determining the effectiveness of MediaPipe Model Maker as a tool for the rapid development of object detection models.

Next, the initial strategy of collecting non-iconic images for the dataset may have included too many challenging images. This issue is highlighted when comparing the performance of the wrench-only model to the model that includes categories for hammers, screwdrivers, and wrenches. The images of wrenches are more iconic due to an adjustment to the search technique to include more iconic images after the hammer images were collected. However, this comparison is not entirely fair because the wrench-only model has an inherently lower risk of incorrect predictions when evaluated on the validation set used to calculate the AP; with only one label beside the default "background" category, the likelihood of mislabeling is reduced compared to a model with multiple object categories. As a reminder to the reader, the entire dataset is available in COCO format on GitHub in the "object detection/LS_export" folder (Hagestuen, 2024*b*). It can be evaluated using tools such as FiftyOne. Alternatively, the reader can inspect the non-labeled version of the images on a local computer without any software tools.

A possible drawback when employing this technique in manufacturing settings is a low number of available images of an object of interest, for example, in part production. An alternative is to create a synthetic dataset using design files from computer programs, such as point clouds and computer-aided design (CAD) models. This would also narrow the scope of this object category to the selected design, possibly enhancing the performance of the object detector. One can project images from the computer program through different camera models to simulate the effect of building a dataset with various cameras.

Despite the poor quantitative results, this method for creating object classifiers can still be viable for manufacturing and industrial applications if its performance is improved to match that of the wrench-only model. It can be beneficial for detecting the *presence* of particular objects, such as on a conveyor belt or at a worktable. However, for applications requiring *separating* different objects piled together or spotting objects under challenging environments, this method for rapid development of an object classifier might be of limited value.

Using the overview of the time required to re-train the object detection model for new object categories, one can estimate the time a consultant would need for this task and the associated cost for a thought manufacturing firm to hire such a consultant. The total time spent was 68.69 hours. By using the step-by-step framework provided, the time spent on export, formatting, and splitting (approximately 10 hours) can almost be neglected, as this task is now replaced by simply calling the "split_dataset.py" script available on GitHub (Hagestuen, 2024*b*). Consequently, the number of hours can be approximated to 60 for three new object categories. Based on the experience gained, the time required is roughly proportional to the number of new object categories for collecting and labeling data and re-training the model. Therefore, an average time and cost can be calculated per new object category introduced to the model. However, one should note that the time spent on iterations and re-training can vary, depending on factors such as computing power, strategies employed for tuning, and the performance requirements of the model. Ultimately, with a per-hour rate of NOK 1200 for hiring a consultant, one can estimate the cost per new object category to:

$$cost\,per\,new\,category = 1200\,NOK/hour * 20\,hours = 24000\,NOK \tag{6.10}$$

Note that this number does not include other parts of the consultant's project, such as reading and adjusting the system specification from the manufacturing firm, integration with other systems, and documentation. On the other hand, the estimated time and cost might be reduced when considering the capabilities expected by an experienced consultant within this field.

Although the resulting model after re-training for new categories yielded poor results, valuable experiences have been gained from this work. As mentioned earlier, these are summarized in a list and should be regarded as supplements to the discussion up until this point:

- Disregard the original strategy of creating a dataset with rich contextual information and instead focus on collecting iconic images of the objects.
- Consider the model resolution of 384x384 pixels when collecting images for the dataset because objects are more challenging to spot in downscaled images. Crop images before they are labeled to counteract this, potentially creating iconic images from non-iconic images.
- Avoid images where the objects are partly occluded or outside the frame.
- Avoid images where the objects are in a crowd, e.g., many screwdrivers piled together, as Label Studio does not support the "isCrowd"-parameter in the COCO format. Alternatively, look into other labeling tools that support this functionality.
- Aligning bounding boxes with the image axes is required when working with the COCO format. Therefore, labeling objects that are tilted might include significant portions of the surrounding area in the bounding box. As such, consider which objects are present in this area and whether or not they represent a disturbance. Alternatively, rotate the images to fit the bounding box better to the object.

Despite generally poor results after re-training the model, note that the existing object detection model's performance demonstrates the *potential* of this method, seeing as the design of the neural network remains unchanged. As such, a conclusion after this experiment is that although there is potential for MediaPipe's object detection mode, re-training it for new object categories proves difficult, and the list provided above should be read carefully when working through the steps in the step-by-step guide.

6.2.3 Experiment 6 - Generating an item set based on visual inspection of scene

6.2.3.1 Introduction

In this experiment, the re-trained model from the previous experiment was employed on images of a workbench containing objects from the object classes for which it was re-trained. Using object detection to determine what is present in the scene is a form for *passive scene description* (Huang et al., 2022) as discussed in the literature review, which is used to collect information about the robot's environment, enabling the task sequencing to be grounded in the real world. As such, this experiment and the associated discussions should be explicitly considered related to the "Object detector" module in Figure 6.15.

6.2.3.2 Methodology

Images of the workbench were captured with the Intel RealSense D455 camera used in earlier experiments. This camera has a 640x480 resolution, which is closer to the model's resolution of 384x384 pixels, but downscaling the images is still necessary. It should be noted that the workbench represents a more detection-friendly scene than many images in the dataset, as the bench is almost uniform in color.

Due to poor model performance after re-training for new object categories and to add insights to the discussion, it was also decided to utilize MediaPipe's original object detector, i.e., the existing detection model, before re-training and replacing the classification layer. This website provides a complete list of all 80 supported labels in this model. The work included easily accessible items from this list, such as frisbee, tie, knife, fork, scissors, bottle, and toothbrush, to qualitatively evaluate MediaPipe's existing object detection model.

On May 13, 2024, OpenAI gave ChatGPT Free users access to previously paid services such as support for uploading and analyzing images (OpenAI, 2024*b*). After initial website testing and consultation with supervisor Mathias H. Arbo, it was decided to also conduct tests using the GPT model family, despite the transition from on-device, free models to remote, paid models. Note that image analysis through OpenAI's API still has an associated cost; only requests made through the website are free. To decrease token usage, image file sizes were reduced by converting the images to .webp format before sending requests through the API.

The two MediaPipe models were tested using MediaPipe Studio, while the OpenAI solution was prompted through the OpenAI Python API. In summary, the experiment qualitatively evaluated three models on images of objects on the workbench:

- The re-trained model from the previous model, trained to recognize hammers, screwdrivers, and wrenches.
- OpenAI's object detection solution, accessible for free on OpenAI's website and for a cost through the API.
- MediaPipe's existing object detection model, trained to recognize 80 labels. In particular, the *EfficientDet-Lite2 float32* model was utilized for evaluation.

6.2.3.3 Results

First, OpenAI's object detector and the re-trained model were evaluated on images containing a hammer, a screwdriver, and a wrench. These results are presented in pairs, where each pair includes the predictions made by the two models on the same image.

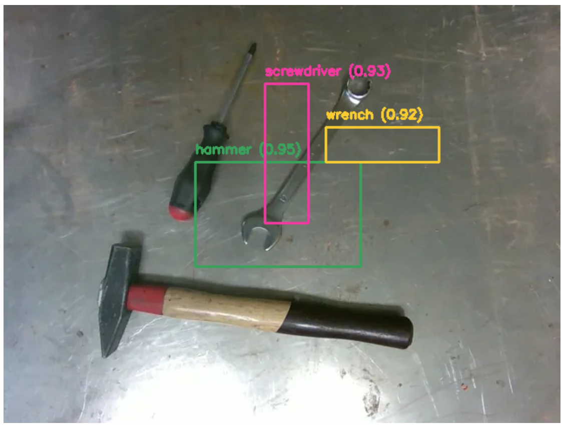


Figure 6.31: Correct object predictions with imprecise bounding box placements. None of the bounding boxes had a satisfying IoU score and would contribute negatively to a thought AP@[0.50:0.95] score. Bounding box predictions were annotated by analyzing text responses in JSON format from GPT-4o through the Python API.

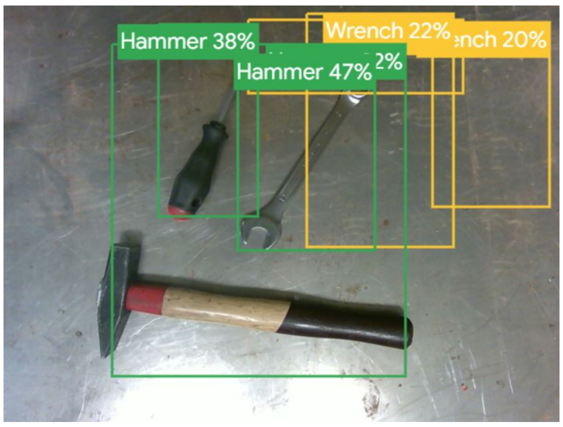


Figure 6.32: Incorrect object predictions. One hammer and one wrench prediction were correct, but bounding box placements were imprecise with an IoU < 50%. The prediction score threshold was set to 15%, and the bounding box predictions were made using MediaPipe Studio.

In the first image, OpenAI's object detector accurately labeled all objects with a high confidence score, as seen in Figure 6.31. However, it misplaced the bounding boxes for all three tools, particularly the hammer. Meanwhile, the re-trained model produced several false positives in Figure 6.32, especially in the top right area of the image. One hammer and one wrench prediction were correct but with inaccurately positioned bounding boxes.

Moving on to the second picture, OpenAI's object detector again successfully categorized the objects in Figure 6.33 and arguably achieved slightly better bounding box placements. However,

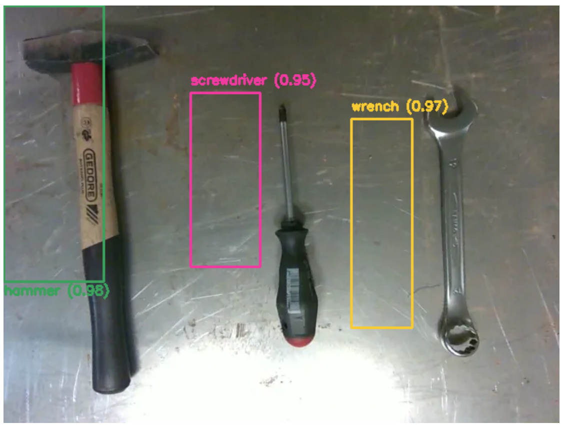


Figure 6.33: Correct object predictions with relatively precise bounding box placements for the hammer. However, the IoU scores for the screwdriver and the wrench are equal to zero, which would contribute negatively towards a thought AP@[0.50:0.95] score. Bounding box predictions are annotated by analyzing text responses in JSON format from GPT-4o through the Python API.

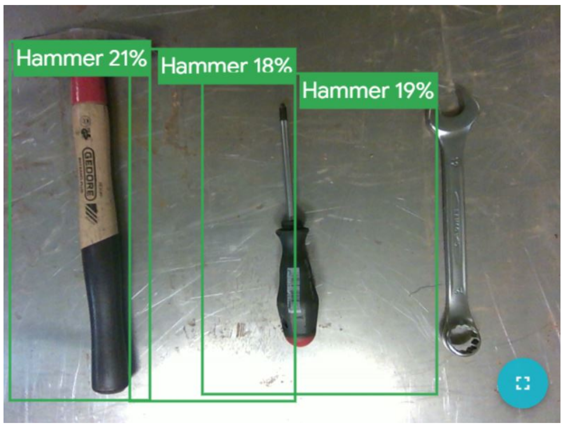


Figure 6.34: Correct prediction of the hammer and incorrect predictions of the screwdriver and the wrench. The correct hammer prediction had a high IoU score. The prediction score threshold was set to 15%, and the bounding box predictions are made using MediaPipe Studio.

the boxes were still poorly positioned, and the IoU scores would not contribute positively to a satisfying AP score. Meanwhile, some of the predictions made by the re-trained model, such as the left-most hammer prediction in Figure 6.34, were correct with a high IoU score. However, the confidence score was relatively low. Finally, the screwdriver and the wrench were wrongfully predicted as hammers.

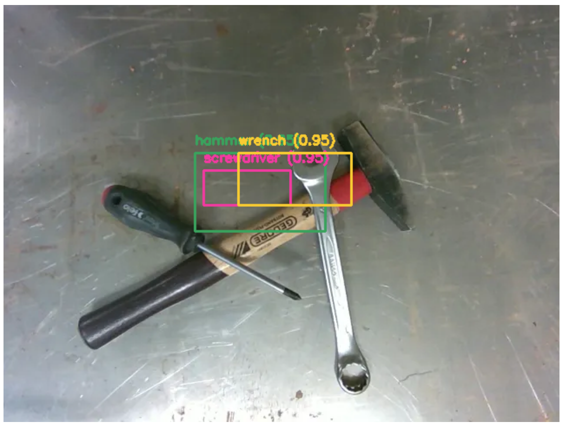


Figure 6.35: Correct object detection predictions with imprecise bounding box placements. None of the bounding boxes had a satisfying IoU score and would contribute negatively to a thought AP@[0.50:0.95] score. Bounding box predictions were annotated by analyzing text responses in JSON format from GPT-4o through the Python API.

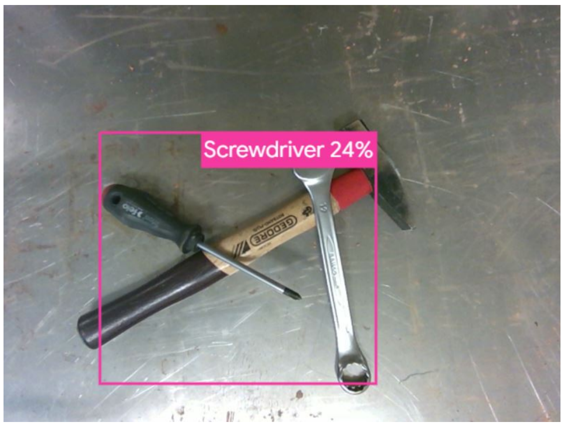


Figure 6.36: One single screwdriver prediction, placed on the pile of objects with an acceptable level of precision. The hammer and wrench went undetected. The prediction score threshold was set to 15%, and the bounding box predictions were made using MediaPipe Studio.

In Figure 6.35, OpenAI's solution successfully detected all objects but failed again to place bounding boxes correctly, yielding low IoU scores. On the other hand, the re-trained MediaPipe model detected only the screwdriver in Figure 6.36, with an IoU score of roughly 50%.

Next, the existing MediaPipe model is evaluated on images of items from the list of labels, see

Figure 6.37, Figure 6.38 and Figure 6.39.

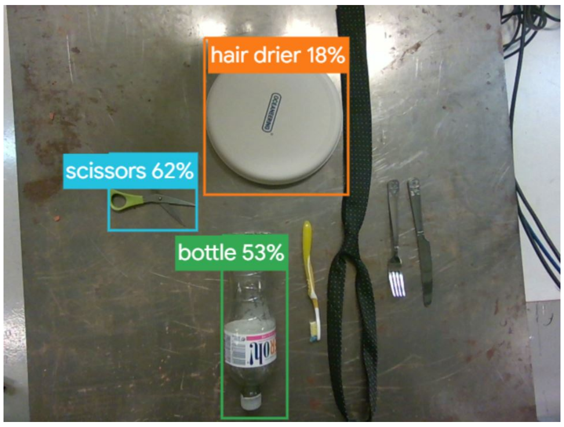


Figure 6.37: Scissors and bottle correctly predicted with great bounding box placements. The toothbrush, tie, knife, and fork were undetected, while the frisbee was categorized as a hair dryer. The prediction score threshold was set to 10%. Bounding box predictions were made using MediaPipe Studio.

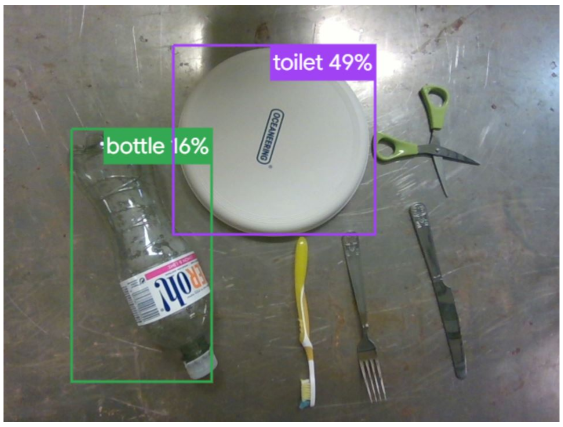


Figure 6.38: Correct prediction of the bottle with a precisely placed bounding box. The frisbee was wrongfully categorized as a toilet, and the other objects in the image were undetected. The prediction score threshold was set to 10%. Bounding box predictions were made using MediaPipe Studio.

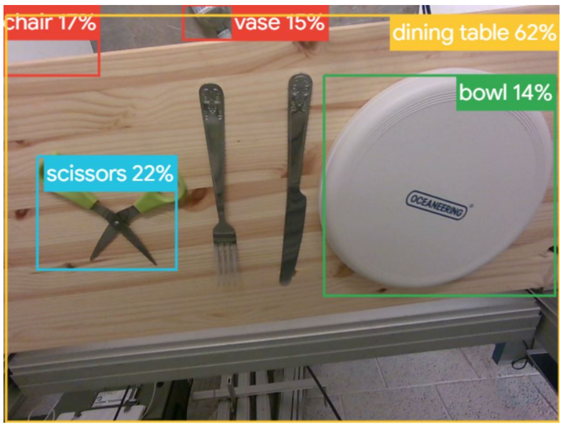


Figure 6.39: Correctly predicted scissors with a great bounding box prediction. The frisbee was wrongfully labeled as a bowl, and the wooden bench was categorized as a dining table. In the background, two unrecognizable shapes were detected as a chair and a vase. The prediction score threshold was set to 10%. Bounding box predictions were made using MediaPipe Studio.

As can be seen, the existing MediaPipe model performed better than the re-trained model for its respective object categories. Some predictions, especially in the first two images, were correct with relatively well-placed bounding boxes. However, this model also produced false positives and wrong labels on objects in the images, such as the toilet prediction in Figure 6.38 and the chair and vase predictions in Figure 6.39. Lastly, the results show several false negatives, i.e., objects in the image that remain undetected. Examples include the toothbrush, tie, fork, and knife.

6.2.3.4 Discussion

Although the re-trained model occasionally makes correct predictions, it cannot be considered accurate because of the large number of incorrect predictions. Given the high volume of predictions, some are bound to be correct by chance; the high presence of errors reduces the reliability of the accurate predictions. Compared to OpenAI's solution, the object detection capabilities are poor and unpredictable, often producing false positives, false negatives, and incorrect labels. OpenAI's detector, included in the experiment for reference, produced accurate labels but suffered from poor bounding box placements in all three images.

Despite its superior performance, a clear disadvantage regarding OpenAI's object detector is the dependency on making requests to a remote server running the model, limiting the scope of applications for such systems. Some issues that arise are dependency on a network, slower response times compared to on-device systems, increased operating costs, and possible sharing of sensitive information. Rough estimates made during testing showed that a request for image analysis through OpenAI's Python API takes $5 - 10$ seconds, while the on-device MediaPipe model lies in the $200 - 300$ milliseconds range.

During testing, it was observed that object detection requests made through the OpenAI API produced different bounding box placements with almost every new call. While precise bounding box placement wasn't the primary focus of this research, the variability in placements could be an issue for applications where object localization is critical. Additionally, the variation in responses from the model raises concerns about how such systems can be safely verified for use in industrial applications, considering their inconsistency. With the increased usage of LLMs in research and development, the topic of varying responses is highly relevant, and careful consideration is necessary when developing systems that rely on such technologies.

Examining the existing MediaPipe detection model, the predictions in Figure 6.39 indicate that detections within the same image were contextually dependent on each other. The model appears to use contextual information and object relations when evaluating the image, leading to false positive detections related to each other, such as a dinner table, vase, bowl, and chair. Since the neural network's design remains unchanged during re-training, this issue will likely persist as the number of labels increases in the new model. However, carefully selecting training images where the context of the objects varies across the dataset can possibly mitigate this drawback to some extent.

A conclusion after qualitative testing is that MediaPipe's object detection solution struggles to deliver consistently accurate results. Additionally, it displays a possible bias towards contextual relations, causing false positive detections to be related to each other category-wise. Finally, the re-trained model with new object categories potentially inherits these issues but is highly dependent on the quality of the training dataset.

6.2.4 Experiment 7 - From language prompt to robotic tasks

6.2.4.1 Introduction

This experiment combines task sequencing using LLM with real-world grounding using object detection to implement a prompt-to-tasks demonstrator. After discovering that OpenAI's object detector yielded better results than the re-trained MediaPipe model, OpenAI's solution was utilized when implementing the prompt-to-tasks demonstrator.

6.2.4.2 Methodology

As demonstrated in the previous experiment, OpenAI's object detector displayed poor localization capabilities. However, for this experiment, the most essential aspect of the object detector is its ability to detect the presence of objects, with accurate localization being less critical. In particular, this is because object grasping was not included in this demonstrator, meaning localization is less critical.

Three scenarios were developed to demonstrate the prompt-to-task system with object detection. Each scenario includes items visible on the workbench and a prompt from the operator containing a desired task for the system to execute.

Scenario 1

- Visible items on the workbench:
  - Wrench
  - Pliers
  - Toothbrush
  - Screwdriver
  - Hammer
  - Bottle
- Operator prompt: "Give me a tool for tightening this bolt."

Scenario 2

- Visible items on the workbench:
  - Wrench
  - Pliers
  - Toothbrush
  - Screwdriver
  - Hammer
  - Bottle
- Operator prompt: "The workbench will be used for assembly tomorrow morning. Remove all items not necessary for assembly."

Scenario 3

- Visible items on the lower workbench:
  - Toothbrush
  - Bottle
- Visible items on the upper workbench:
  - Screwdriver
  - Hammer
- Operator prompts:
  1. "I need to fix my bed, are there any items here that I can use? Make sure you inspect all workbenches." Followed up by:
  2. "Great, now give me the items I can use to fix my bed."

See algorithm 8 for a top-level description of the desired behavior of the prompt-to-tasks robotic system. The repeat condition was set based on the objects detected in the robot's view; detecting a different set of objects indicated a scene change, and the loop would repeat.

```
Algorithm 8: Prompt-to-tasks Robotic System
  Data: Robot IP address, camera camera calibration data, OpenAI API key
  Result: Executed sequence of tasks
  Load calibration data;
  Initialize camera;
  Initialize robot;
  Initialize language model with system message;
  Move the robot to the initial position;
  repeat ← True
  while repeat do
      Capture an image of the workbench;
      Detect and classify objects on the workbench;
      Receive user prompt from operator;
      Generate a sequence of tasks using the language model;
      Execute tasks in the sequence;
      Check if new tasks are needed and update repeat condition;
```

As mentioned, OpenAI's object detector was utilized in this demonstrator, meaning that requests were made through OpenAI's API in both the object detector module and the LLM module in Figure 6.15. However, these two pipelines were separate and not connected, meaning the object detector module could be switched out for other solutions if needed.

The actions available to the robot were very similar to those in an earlier experiment:

A Pick up item 'X' from the workbench.

B Give item 'X' to the operator.

C Equip item 'X'.

D Move the end-effector to view the alternative workbench.

E Capture an image from the current camera angle.

The results are presented in video format with a corresponding explanation of the events in the video, allowing the reader to follow the execution with ease. For demonstration purposes, picking up and handing items to the operator were simulated by moving the end-effector to the relevant position.

6.2.4.3 Results

For each scenario, a YouTube video with corresponding lists of events is presented. Consider these lists as supplements to the videos. Note that the videos were muted due to excess background noise from other work at the lab at the time of recording.

Scenario 1

Results from scenario 1 can be viewed on YouTube through this link, and the events in the video were as follows:

- 0:03, the Python script is run through the terminal.
- 0:13, the RealSense camera is successfully initialized.
- 0:17, connection to the robot is established.
- 0:19, the robot moves to its initial position.

- 0:24, an image of the workbench is captured, and the object detector starts analyzing the image.
- 0:32, the list of visible items is received from the object detector and printed to the console, awaiting the user to enter the desired prompt. All objects in the scene were detected:
  - Hammer
  - Screwdriver
  - Toothbrush
  - Wrench
  - Pliers
  - Bottle
- 0:40, the operator enters the prompt, which is sent to the LLM with a request for generating a task sequence.
- 0:42, a task sequence is received from the LLM and printed to the console:
  1. ['A', 'wrench']. Pick up item *wrench* from the workbench.
  2. ['B', 'wrench']. Give item *wrench* to the operator.
- 0:44, the robot starts executing the tasks in order, starting with picking up the wrench from the workbench. As a reminder, the robot movements are purely for demonstration purposes.
- 0:47, the wrench has been picked up, and the next action is initiated.
- 0:54, the wrench is handed to the operator, marking the end of the task sequence.

Scenario 2

Results from scenario 2 can be viewed on YouTube through this link, and the events in the video were as follows:

- 0:02, the Python script is run through the terminal.
- 0:11, the RealSense camera is successfully initialized.
- 0:15, connection to the robot is established.
- 0:17, the robot moves to its initial position.
- 0:22, an image of the workbench is captured, and the object detector starts analyzing the image.
- 0:30, the list of visible items is received from the object detector and printed to the console, awaiting the user to enter the desired prompt. All objects in the scene were detected:
  - Hammer
  - Screwdriver
  - Toothbrush
  - Wrench
  - Pliers
  - Bottle
- 0:43, the operator enters the prompt, which is sent to the LLM with a request for generating a task sequence.
- 0:44, a task sequence is received from the LLM and printed to the console:
  1. ['A', 'toothbrush']. Pick up item *toothbrush* from the workbench.

2. ['B', 'toothbrush']. Give item *toothbrush* to the operator.
3. ['A', 'bottle']. Pick up item *bottle* from the workbench.
4. ['B', 'bottle']. Give item *bottle* to the operator.

- 0:48, the robot starts executing the tasks in order, starting with picking up the toothbrush from the workbench.
- 0:50, the toothbrush has been picked up, and the next action is initiated.
- 0:57, the toothbrush is handed to the operator, and the next action is initiated.
- 1:01, the bottle has been picked up from the workbench, and the next action is initiated.
- 1:09, the bottle is handed to the operator, marking the end of the task sequence.

Scenario 3

Results from scenario 3 can be viewed on YouTube through this link, and the events in the video were as follows:

- 0:01, the Python script is run through the terminal.
- 0:12, the RealSense camera is successfully initialized.
- 0:16, connection to the robot is established.
- 0:19, the robot is at its initial position. An image of the workbench is captured, and the object detector starts analyzing the image.
- 0:23, the list of visible items is received from the object detector and printed to the console, awaiting the user to enter the desired prompt. Both objects in the scene were detected:
  - Toothbrush
  - Bottle
- 0:30, the operator enters the prompt, which is sent to the LLM with a request for generating a task sequence.
- 0:31, a task sequence is received from the LLM and printed to the console:
  1. ['D', '']. Move the end-effector to view the alternative workbench.
  2. ['E', '']. Capture an image from the current camera angle.
- 0:34, the robot starts executing the tasks in order, starting with moving the end-effector to view the upper workbench.
- 0:40, the robot now has a view of the upper workbench and captures an image of this workbench. This marks the end of the first task sequence. The object detector starts analyzing the image.
- 0:47, the list of visible items is received from the object detector. Both objects in the new scene were detected:
  - Hammer
  - Screwdriver

  Since the list differs from the previous one, the operator can enter another prompt.
- 1:00, the operator enters the prompt, which is sent to the LLM with a request for generating a new task sequence.
- 1:01, a task sequence is received from the LLM and printed to the console:
  1. ['A', 'hammer']. Pick up item *hammer* from the workbench.

2. ['B', 'hammer']. Give item *hammer* to the operator.
3. ['A', 'screwdriver']. Pick up item *screwdriver* from the workbench.
4. ['B', 'screwdriver']. Give item *screwdriver* to the operator.

- 1:04, the robot starts executing the tasks in order, starting with picking up the hammer from the upper workbench.
- 1:06, the hammer has been picked up, and the next action is initiated.
- 1:12, the hammer is handed to the operator, and the next action is initiated.
- 1:15, the screwdriver has been picked up from the workbench, and the next action is initiated.
- 1:21, the screwdriver is handed to the operator, marking the end of the task sequence.

6.2.4.4 Discussion

A central aspect of this demonstrator was the robot's ability to pick up tools based on text prompts from the operator. This functionality shares similarities with a thought tool-selecting system in CNC machines, where the operator specifies the desired task in text format. By utilizing a predefined set of actions and knowledge about the tool rack's content, the system exploits the capabilities of an LLM and its awareness of its surroundings to execute the task described by the operator.

Another aspect is the robot's role as an assistant to assembly workers, as showcased in scenario 2. Before the workers arrive at work in the morning, the robot can organize the assembly table based on the scheduled tasks for the day. Leveraging the knowledge in an LLM, the robot can determine which tools are necessary for the tasks and remove any unnecessary items from the table, ultimately improving the worker's efficiency as they start their shift. Such an assistant would be especially relevant for collaborative robots where the worker and the robot share the same workspace but do not necessarily work simultaneously, as illustrated in the middle case in Figure 6.40.

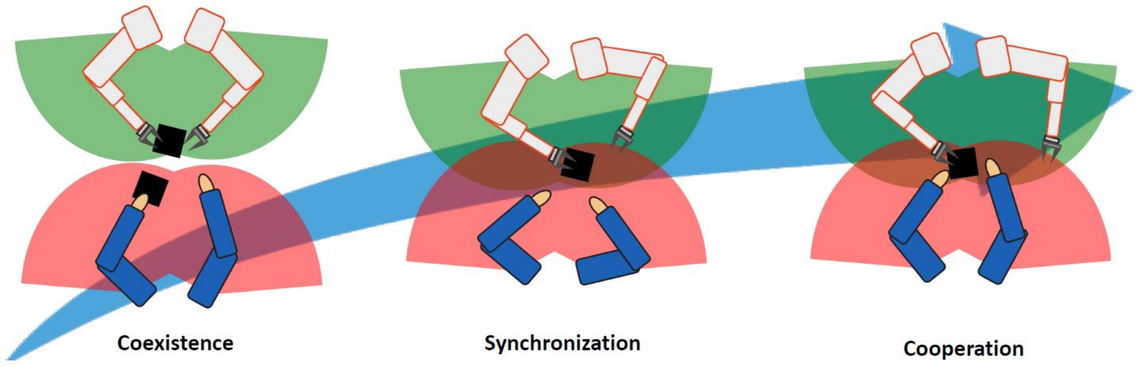


Figure 6.40: Different collaboration levels, from having distinct workspaces (coexistence) to working simultaneously in the same workspace with the robot (cooperation). Synchronization is the trade-off between the two, where only the robot or the worker is present in the workspace at any given time despite sharing it.
Image source: lecture slides by Prof. Paolo Rocco (Rocco, n.d.) in the course Control of industrial robots delivered at Politecnico di Milano (Rocco, 2024).

The ability to extend the list of available actions to the robot is critical to the potential of a prompt-to-tasks system, allowing it to generate a sequence containing more complex tasks. In manufacturing, such new actions could be welding in a production cell, moving items on a conveyor belt according to a sorting system, or even visual servoing using camera-based hand tracking for part inspection. Such flexibility in task planning and execution can reduce downtime and increase overall productivity in manufacturing environments. Additionally, using natural language processing for task input can make the system more accessible to non-technical operators, leading to more widespread use of advanced robotics in the workplace and reducing personnel training expenses.

As mentioned, object grasping was omitted from the experiment but is still a crucial aspect should such a system be implemented on the manufacturing floor. Unless the positions of items are known (e.g., using a tool rack), the robot relies on precise localization of objects through the visual system to successfully grasp objects with the end-effector. As discovered in the previous experiment, the OpenAI object detection model struggles with localization despite correctly categorizing objects in the scene. This highlights an important area for future work on this subject; research must investigate alternative object detection models with better localization capabilities and look into how these models can be integrated to allow object grasping through real-time visual feedback. This problem shares similarities with the visual servo schemes implemented in this master thesis; using real-time position estimation of a target in the camera's view, the system must control the end-effector to minimize a desired tracking offset. When grasping, this offset is constant and depends on the object to be grasped, introducing the challenging aspect of adapting to a broad range of object categories and their associated geometries.

One of the most challenging aspects of developing a prompt-to-tasks system is accurately gathering information about the environment through object detection. On paper, the open-source MediaPipe object detector might be useful for consumer applications due to its small model size and low resource requirements. However, it performs poorly in manufacturing settings like those highlighted in this thesis. As mentioned, there are indications that the model is biased toward contextual information, which can be a weakness in unpredictable industrial environments. Next, re-training MediaPipe's object detector for new object categories proves difficult due to several factors that add to the challenges present in the existing model. These factors include the need for a high-quality dataset, significant image downscaling in the model, and the unknown impact of object orientation in training images. In contrast, the OpenAI object detector demonstrates promising detection capabilities but struggles to precisely locate objects in the frame. Localization is critical for typical manufacturing applications like tool grasping due to the reliance on visual feedback. The model also struggles to respond consistently when prompted about object localization in images. Additionally, OpenAI's models are paid services running on remote servers, raising several issues mentioned earlier, such as dependency on a network, slower response times compared to on-device systems, increased operating costs, and possible sharing of sensitive information.

Next, LLMs show promise in prompt-to-task systems due to their extensive knowledge base, which is useful for parsing language inputs and mapping them to a predefined set of robotic actions. However, this thesis has shown that integrating LLMs presents its own set of challenges. One significant issue is the variability in the LLM's responses; small changes to the input prompt can lead to significantly different outputs, negatively affecting the reliability and predictability of the system, which are essential aspects in applications where consistency is important.

In conclusion, while integrating object detection models and LLMs into prompt-to-task robotic systems offers exciting possibilities for increased automation in manufacturing, significant challenges remain. Having an object detection model with great detection and localization capabilities and ensuring reliable LLM performance are critical steps toward realizing the potential of these technologies in manufacturing settings.

Chapter 7

Conclusion and future work

7.1 Conclusion

In conclusion, through a series of experiments, this thesis demonstrates the feasibility and potential applications of machine learning techniques for human-robot collaboration in manufacturing. Specifically, it focuses on hand-tracking using MediaPipe coupled with a depth camera for robotic visual servo systems, re-training MediaPipe's object detector for new object categories in manufacturing, and language model-driven robot task planning using the GPT model family.

The research investigates the feasibility of two hand-tracking methods for visual servo systems: ArUco marker, and MediaPipe combined with a depth camera like Intel RealSense. While the ArUco marker provides reliable tracking, it requires a physical marker, limiting its applicability in dynamic environments. On the other hand, the MediaPipe and depth camera method shows promising capabilities for gloveless hand tracking. However, it faces challenges in typical workspace scenarios, particularly when the user holds tools when wearing gloves. This highlights the need for further work on camera-based tracking technologies to improve their reliability in various working conditions.

Next, re-training the MediaPipe object detector for new categories using MediaPipe Model Maker highlighted several challenges. Despite being a fast, on-device model with low resource requirements for re-training, significant image downscaling requires the researcher to disregard the desire for a contextually rich dataset and instead focus on collecting iconic images of the objects. Additionally, the experimental results suggest a potential model bias towards contextual relations in test images, causing false positive, correlated category-wise detections. Despite the challenges of re-training for new object categories, following the step-by-step customization guide and experience-based recommendations in this thesis allows a researcher to customize MediaPipe's object detector in just 20 hours per new category, including the time needed to create the dataset.

Furthermore, integrating Large Language Models (LLMs) with object detection in prompt-to-tasks systems shows great promise. LLMs excel in understanding and natural language, facilitating robotic task planning based on simple verbal instructions from human operators. Meanwhile, object detection allows situation-aware task planning through real-world grounding. This approach is instrumental in dynamic environments where tasks frequently change and require flexibility, making the system more accessible to non-technical operators, potentially leading to more widespread use of robotics in the workplace. However, this thesis has shown that integrating LLMs presents the challenge of variability in the LLM's responses; small changes to the input prompt can lead to significantly different outputs, negatively affecting the reliability and predictability of the system.

The systems and methodologies presented in this thesis demonstrate their feasibility and highlight their potential applications in manufacturing and collaborative robotics. For instance, visual servo systems using device-less tracking methods could be employed in tasks such as ergonomic lift-by-guiding tasks or testing fixing and mounting points on parts during rapid movements by selecting an aggressive controller. Similarly, prompt-to-task systems utilizing LLMs and object detection can be employed in assembly lines or tool-selecting systems for CNC machines, where intuitive and efficient collaboration is crucial.

7.2 Future work

This section summarizes the future work mentioned throughout the thesis.

7.2.1 Machine learning dataset tailored for hand-detection in the workspace

Experimental results indicate significant variation in hand-detection capabilities in different workspace scenarios, particularly when the human operator is wearing work gloves. As discussed, performance can be improved by wearing certain types of gloves, but this solves the problem from the wrong end; instead, time should be devoted to creating a tailored dataset for training a machine learning model for hand detection in workspace conditions.

7.2.2 Quantify positional error of hand tracking methods

A sophisticated experimental setup can be implemented, including a system for tracking the hand or object's true position, allowing the positional error to be quantified in metric units. For object tracking using an ArUco marker, an idea is to use a cart-on-a-rail system with a high-precision encoder. Quantifying positional error in hand tracking is more challenging. Still, it can be realized by an indoor position system (IPS) or by using prostheses mounted on an encoder system instead of human hands.

Additionally, the tracking capabilities in the camera's x-, y-, and z directions should be investigated separately by isolating contributions from movements along each axis.

7.2.3 Kalman filter - extension to loss of frames

Utilizing a Kalman Filter and performing dead reckoning should be explored as a proposed countermeasure to detection systems with a lower detection success rate. This equates to executing the Kalman Filter prediction step without a follow-up correction step for iterations where hand detection failed.

7.2.4 Estimating full hand pose using MediaPipe and Intel RealSense

This thesis has demonstrated position estimation of a hand in space. A natural next step is to investigate the potential for estimating full hand pose using MediaPipe with a depth camera, such as Intel RealSense. By defining a coordinate frame at the hand using the estimated 3D coordinates of selected landmarks, relative depth information between these landmarks provided by MediaPipe's hand detector can be utilized. Only the depth to the wrist landmark needs to be estimated by the depth camera. The orientational information about the hand can then be used in the visual servo system to rotate the end effector, with a control objective that constrains the hand pose to a specific orientation in the camera frame.

7.2.5 Standardize prompt structures to improve consistency of LLM

Attention should be devoted to developing techniques to standardize prompt structures to minimize the variability in LLM outputs for robotic task planning. Reducing variability in LLM responses is essential to enhancing the reliability of the prompt-to-tasks mapping system.

7.2.6 Movement algorithm for scanning the robot's environment

The current study's scope is constrained by assuming all available items and tools are visible on the workbench. This assumption may not always hold, potentially requiring the development of a sophisticated algorithm allowing quick and reliable scanning of the robot's surroundings using object detection in an eye-in-hand configuration to generate a set of available actions.

7.2.7 Grasping of objects using visual guiding

The ability to grasp objects using visual localization feedback is crucial for the prompt-to-task system to execute tasks as the task planner directs. Future work should focus on integrating localization estimates from the object detector into the robot's movement algorithm via visual feedback, enabling the robot to grasp objects within its environment. This functionality relies on having a detection model that provides precise localization estimates of objects in the camera's view.

Appendix A

Literature search strings

This appendix summarizes the searches made in Scopus before the literature search strategy was adapted. Table A.1 covers the search strings when looking for research on (1) human-controlled visual robot-servo, and Table A.2 covers the search strings when looking for research on (2) mapping natural language prompts to robotic tasks.

As mentioned in the literature review, the methodology was altered after realizing that a unified search for articles encompassing all individual components of the problem could limit the search's reach. For instance, searching for articles addressing both hand tracking and visual servoing yielded limited results, especially when constraining the use case to manufacturing applications. This part of the appendix will summarize some of these searches, allowing them to be repeated later.

The keywords used in the search, with corresponding explanations, are as follows:

- "AND" & "OR" represents logical operators between keywords, with AND performed first in the order of precedence.
- "ABS" constrains the search to only the article's abstract.
- "PUBYEAR" specifies the year the article was published.

Note that if the searches are repeated later, the upper limit for PUBYEAR must be increased to allow for potentially new articles.

Some of the entries in the table produce a number of articles that are beyond manageable. Because of this, the articles were first sorted by *Relevance* and then by *Cited by (highest)* in the Scopus search engine. In both cases, the first 20 articles were considered after sorting. Note that the search results, when sorting by *Relevance*, seem to change significantly over time. Naturally, sorting results by *Cited by (highest)* also changes with the cite count.

As emphasized in the literature review, some of the articles produced from these searches were included in the study despite deciding to change the strategy to a broader search. As such, the strategy adaption can be seen as a supplement rather than a replacement to the searches seen in Table A.1 and Table A.2.

Search string	**Number of articles**	**Date of search**
((hand AND tracking) OR (hand AND detection) OR (hand AND gestures) OR (hand AND recognition)) AND ((visual AND servoing) OR (visual AND servo) OR (visual AND task AND control) OR (gesture-based AND control)) AND (manufacturing OR (robotic AND assembly)) AND PUBYEAR > 2017 AND PUBYEAR < 2025	5117	February 8th, 2024
ABS (((hand AND tracking) OR (hand AND detection) OR (hand AND gestures) OR (hand AND recognition)) AND ((visual AND servoing) OR (visual AND servo) OR (visual AND task AND control) OR (gesture-based AND control)) AND (manufacturing OR (robotic AND assembly))) AND PUBYEAR > 2017 AND PUBYEAR < 2025	19	February 8th, 2024
((hand AND tracking) OR (hand AND detection) OR (hand AND gestures) OR (hand AND recognition)) AND ((visual AND servoing) OR (visual AND servo) OR (visual AND task AND control) OR (gesture-based AND control)) AND PUBYEAR > 2017 AND PUBYEAR < 2025	29109	February 8th, 2024
ABS (((hand AND tracking) OR (hand AND detection) OR (hand AND gestures) OR (hand AND recognition)) AND ((visual AND servoing) OR (visual AND servo) OR (visual AND task AND control) OR (gesture-based AND control))) AND PUBYEAR > 2017 AND PUBYEAR < 2025	416	February 8th, 2024

Table A.1: Scopus-searches that were conducted to find literature and research on hand tracking in visual servo systems. In two of the searches, the use cases of the results are also constrained by additional keywords.

Search string	**Number of articles**	**Date of search**
((natural AND language) OR (language AND prompts)) AND ((robot AND task) OR (robot AND action) OR (robot AND planning)) AND (manufacturing OR (robot AND assembly)) AND PUBYEAR > 2017 AND PUBYEAR < 2025	3574	March 18th, 2024
ABS (((natural AND language) OR (language AND prompts)) AND ((robot AND task) OR (robot AND action) OR (robot AND planning)) AND (manufacturing OR (robot AND assembly))) AND PUBYEAR > 2017 AND PUBYEAR < 2025	32	March 18th, 2024

Table A.2: Scopus searches conducted to find literature and research on mapping natural language prompts to robotic tasks.

Appendix B

Articles used in the literature study

The literature review in chapter 2 is based on the literature seen in Table B.1 and Table B.2. The table was split into two due to length.

Article	**Main contribution**
Hutchinson et al. (1996)	Provides an introduction to visual servoing, focusing on the basic concepts. Refers to Hill and Park Hill and Park (1979), distinguishing the open-loop look-and-move systems from the closed-loop visual servo control.
Jiang et al. (2022)	Demonstrates a visual servo controller on a 7 DOF robot, allowing motions in the null-space of the manipulator satisfying human-like behavior of the robot arm by replicating the swivel angle of the operator's arm.
Gong et al. (2018)	Proposes a visual servo method for use without prior knowledge of camera parameters and hand-eye relationship, featuring a task function rooted in projective homography.
Wang et al. (2019)	Presents a practical approach that utilizes a wearable sensory system to recognize human hand-over intentions.
Wu and Cheng (2018)	Proposes an algorithm to estimate the depth of any face of a cuboid, with prior knowledge about the dimensions of the six faces.
Yuan et al. (2018)	Explores the current state of 3D hand pose estimation, one of the findings being that 3D volumetric representations outperform 2D CNNs. Also addresses the next challenges that need to be solved, like pose estimation for extreme viewpoints.
Xiao and Chen (2019)	Leverages partial knowledge of the target's dynamics to improve on challenges caused by low sampling rates and delay in the visual feedback loop.
Shirai (1971)	An early implementation of a closed-loop visual servo utilizing a commercial vidicon TV camera, implementing a system for placing a square prism block into a square block.
Oberweger et al. (2015)	Integrates a prior model into a neural network to improve the accuracy of hand pose estimation. Introduces an innovative method for locating finger joints.
Fang et al. (2023)	Provides a map of currently deployed head-mounted AR systems in the manufacturing industry, and related challenges.
Cuevas-Velasquez et al. (2019)	Demonstrates a hybrid system combining eye-to-hand and eye-in-hand visual input, where a master supervisor selects which input to use to exploit the advantages of both configurations.
Guo-Dong et al. (2010)	Utilizes image moments, referred to as a kind of global image feature, claiming that they provide a generic representation of any object. This is opposed to utilizing simpler geometric image features like circles, lines, and dots
Chaumette (2004)	Derives the analytical form of interaction matrix related to an arbitrary image moment. Proposes moments to be used in image-based visual servoing.
Muthusamy et al. (2021)	Solves challenges related to low sampling rate and latency by utilizing an event-based neuromorphic camera, a camera only recording changes in the scene by measuring light intensity.
Tahri and Chaumette (2005)	Derives the analytical form of the interaction matrix related to generic image moments, and presents improvements to image-based visual servoing using these moments.

Table B.1: First part of the overview of all articles used in the literature study in this master thesis.

Article	**Main contribution**
Hill and Park (1979)	An early introduction of the term ”visual servoing”, separating it from earlier open-loop approaches, where the vision system is deactivated when the manipulator is moving.
Ge et al. (2018*a*)	Presents a CNN-based method producing the full 3D hand pose from a 3D volumetric representation of the depth image of the hand
Tompson et al. (2014)	Real-time continuous pose recovery of markerless complex objects from a single depth image using 2D CNNs.
Cao and Wang (2019)	Presents a scheme where the fingertips of the operator are tracked using skin color segmentation. Bezier curves are fitted to control points generated through the Monte Carlo method.
Li and Zhang (2023)	Proposes an RL-scheme where the agent chooses the servo-gain λ adaptively at each time-step to improve visual servo performance.
Ge et al. (2018*b*)	Introduces a multi-view CNN-based approach for 3D hand pose estimation, which leverages multiple views of depth images to improve accuracy and robustness.
Wu et al. (2019)	Separates between device-based and camera-based hand detection. Proposes using multiple depth cameras to allow hand tracking without relying on a wearable.
Ahn et al. (2022)	Maps natural language instructions to robot actions that are not only feasible for the robot, but also contextually appropriate by using value functions associated with each pre-trained skill.
Huang et al. (2022)	Improves on Ahn et al. (2022) by incorporating three types of feedback to achieve better performance: *success detection*, *passive scene description*, and *active scene description*.
Vemprala et al. (2023)	Provides design principles exploiting knowledge contained by an LLM like ChatGPT to fundamentally change the way a robotic task is translated into code, mainly to decrease the dependency on a specialized engineer.
Uboweja et al. (2023)	Utilizes the MediaPipe Model Maker solution to adapt the hand gesture recognition solution with custom data.
Lin et al. (2014)	Creates COCO 2014, a dataset for training object recognition models. The work focuses on creating a dataset with objects in natural contexts instead of iconic images.
Liu and Zhang (2019)	Overview of methodologies employed for human-robot cooperation. The paper covers language instruction understanding, language-based execution plan generation, and knowledge-world mapping.

Table B.2: Second part of the overview of all articles used in the literature study in this master thesis.

Appendix C

Full-text YouTube links

The full-text YouTube links provided in this thesis are as follows:

- Visual servo with ArUco marker tracking: https://www.youtube.com/shorts/P0-OTJAlkfY
- Visual servo using MediaPipe and RealSense for hand tracking: https://www.youtube.com/shorts/297Iiu7ZTak
- Visual servo using MediaPipe and RealSense for hand tracking, wearing orange gloves: https://www.youtube.com/shorts/_bIRd0Ux0aA
- Prompt-to-tasks demonstrator, scenario 1: https://www.youtube.com/shorts/gP4o9aZAqC0
- Prompt-to-tasks demonstrator, scenario 2: https://www.youtube.com/watch?v=3rc1f9_7mcg
- Prompt-to-tasks demonstrator, scenario 3: https://www.youtube.com/watch?v=nWEUiikISkI

Bibliography

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Ho, D., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jang, E., Jauregui Ruano, R., Jeffrey, K., Jesmonth, S., J. Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Lee, K.-H., Levine, S., Lu, Y., Luu, L., Parada, C., Pastor, P., Quiambao, J., Rao, K., Rettinghouse, J., Reyes, D., Sermanet, P., Nicolas, S., Tan, C., Toshev, A., Vanhoucke, V., Xia, F., Xiao, T., Xu, P., Xu, S., Yan, M. and Zeng, A. (2022), Do As I Can, Not As I Say: Grounding Language in Robotic Affordances, Arxiv.

Andreff, N., Horaud, R. and Espiau, B. (1999), On-line Hand-Eye Calibration, *in* 'Proceedings of the 2Nd International Conference on 3-D Digital Imaging and Modeling', ResearchGate.

Burger, W. (2016), Zhang's Camera Calibration Algorithm: In-Depth Tutorial and Implementation, Technical Report HGB16-05, University of Applied Sciences Upper Austria, School of Informatics, Communications and Media, Hagenberg, Austria.

Cao, S. and Wang, X. (2019), 'Real-time dynamic gesture recognition and hand servo tracking using PTZ camera', *Multimedia Tools and Applications* **78**, 27403–27424.

Chaumette, F. (2004), 'Image moments: a general and useful set of features for visual servoing', *IEEE Transactions on Robotics* **20**(4).

CreativeCommons (n.d.), 'CC BY-SA 4.0 DEED', https://creativecommons.org/licenses/by-sa/4.0/. Date accessed: May 8, 2024.

Cuevas-Velasquez, H., Li, N., Tylecek, R., Saval-Calvo, M. and Fisher, R. B. (2019), Hybrid Multi-camera Visual Servoing to Moving Target, *in* '2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', IEEE, Madrid, Spain, pp. 1132–1137.

Daniilidis, K. (1999), 'Hand-Eye Calibration Using Dual Quaternions', *The International Journal of Robotics Research* **18**(3), 186–298.

Fang, W., Chen, L., Zhang, T., Chen, C., Teng, Z. and Wang, L. (2023), 'Head-mounted display augmented reality in manufacturing: A systematic review', *Robotics and Computer-Integrated Manufacturing* **83**.

Ge, L., Liang, H., Yuan, J. and Thalmann, D. (2018*a*), 'Real-Time 3D Hand Pose Estimation with 3D Convolutional Neural Networks', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **41**(4).

Ge, L., Liang, H., Yuan, J. and Thalmann, D. (2018*b*), 'Robust 3D Hand Pose Estimation From Single Depth Images Using Multi-View CNNs', *IEEE Transactions on Image Processing* **27**(9), 4422–4436.

Gong, Z., Tao, B., Yang, H., Yin, Z. and Ding, H. (2018), 'An Uncalibrated Visual Servo Method Based on Projective Homography', *IEEE Transactions on Automation Science and Engineering* **15**(2), 806–817.

Google (2022*a*), 'Open Images Dataset V7 and Extensions', https://storage.googleapis.com/openimages/web/index.html. Date accessed: May 19, 2024.

Google (2022*b*), 'TensorFlow Text - Text processing in Tensorflow', https://github.com/tensorflow/text#a-note-about-different-operating-system-packages. Date accessed: May 21, 2024.

Google (2023*a*), 'Hand landmarks detection guide', https://developers.google.com/mediapipe/solutions/vision/hand_landmarker. Date accessed: January 29, 2024.

Google (2023*b*), 'MediaPipe Model Maker', https://developers.google.com/mediapipe/solutions/model_maker. Date accessed: March 18, 2024.

Google (2023*c*), 'Object detection model customization guide', https://developers.google.com/mediapipe/solutions/customization/object_detector. Date accessed: February 6, 2024.

Guo-Dong, L., Guo-Hui, T. and Ying-Hua, X. (2010), Visual servoing using image moments and nonlinear state error feedback control law, Xuzhou, China.

Haavardsholm, T. V. (2023), 'A handbook in Visual SLAM', https://github.com/tussedrotten/vslam-handbook/releases.

Hagestuen, E. V. (2023*a*), Hand tracking for visual robot-servo, Specialization project, NTNU.

Hagestuen, E. V. (2023*b*), 'realsense_aruco', https://github.com/Endrevh/realsense_aruco.git. Commit hash: 74cacba731970f5d848e4dadc31212af5bf22ae4.

Hagestuen, E. V. (2024*a*), 'hand-tracking', https://github.com/Endrevh/hand-tracking. Commit hash: 04a7e70dc43d0b3261016da866a6aba94dde81cc.

Hagestuen, E. V. (2024*b*), 'NL2Robot-Mapping', https://github.com/Endrevh/NL2Robot-Mapping.git. Commit hash: d1f67903b5aebee0bac3f3b97816dac7965491d2.

hello9 (2023), 'The cumulative token problem and role = system usage, options?', https://community.openai.com/t/the-cumulative-token-problem-and-role-system-usage-options/84789/1. Date accessed: February 15, 2024.

Hill, J. and Park, W. T. (1979), Real time control of a robot with a mobile camera, *in* 'Proc. 9th ISIR', Washington, DC, pp. 233–246.

Horaud, R. and Fadi, D. (1995), 'Hand-Eye Calibration', *The International Journal of Robotics Research* **14**(3), 195–210.

Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Chebotar, Y., Sermanet, P., Brown, N., Jackson, T., Luu, L., Levine, S., Hausman, K. and Ichter, B. (2022), Inner Monologue: Embodied Reasoning through Planning with Language Models, Arxiv.

Hutchinson, S., Hager, G. D. and Corke, P. I. (1996), 'A tutorial on visual servo control', *IEEE Transactions on Robotics and Automation* **12**(5), 651–670.

Jawad, E. (2023), 'The deep neural network - a review', *Journal of Mathematics* **9**(9), 1–5.

Jiang, J., Wang, Y., Jiang, Y., Xie, H., Tan, H. and Zhang, H. (2022), 'A Robust Visual Servoing Controller for Anthropomorphic Manipulators With Field-of-View Constraints and Swivel-Angle Motion: Overcoming System Uncertainty and Improving Control Performance', *IEEE Robotics & Automation Magazine* **29**(4).

Kukil (2022), 'Introduction to MediaPipe', https://learnopencv.com/introduction-to-mediapipe/. Date accessed: December 15, 2023.

Li, M. and Zhang, X. (2023), RLC-Servo: a full-automatic hand-eye cooperative servo model based on reinforcement learning, Vol. 12705, SPIE, Nanjinh, China.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C. L. (2014), Microsoft COCO: Common Objects in Context, *in* 'Computer Vision – ECCV 2014', Vol. 8693 of *Lecture Notes in Computer Science*, Springer, Cham, pp. 740–755.

Liu, R. and Zhang, X. (2019), 'A review of methodologies for natural-language-facilitated human–robot cooperation', *International Journal of Advanced Robotic Systems* **16**(3).

Muthusamy, R., Ayyad, A., Halwani, M., Swart, D., Gan, D., Lakmal, S. and Zweiri, Y. (2021), 'Neuromorphic Eye-in-Hand Visual Servoing', *IEEE Access* **9**, 55853–55870.

Myhre, T. (n.d.), 'Robot camera calibration', https://www.torsteinmyhre.name/snippets/robcam_calibration.html. Date accessed: November 19, 2023.

Oberweger, M., Wohlhart, P. and Lepetit, V. (2015), Hands Deep in Deep Learning for Hand Pose Estimation, *in* 'Proceedings of 20th Computer Vision Winter Workshop', pp. 21–30.

OpenAI (2022), 'Introducing ChatGPT', https://openai.com/blog/chatgpt. Date accessed: February 9, 2024.

OpenAI (2024*a*), 'Best Practices for API Key Safety', https://help.openai.com/en/articles/5112595-best-practices-for-api-key-safety. Date accessed: February 15, 2024.

OpenAI (2024*b*), 'Introducing GPT-4o and more tools to ChatGPT free user', https://openai.com/index/gpt-4o-and-more-tools-to-chatgpt-free/. Date accessed: May 14, 2024.

OpenAI (2024*c*), 'OpenAI Pricing', https://openai.com/pricing. Date accessed: March 18, 2024.

Pan, S. and Wang, X. (2021), A Survey on Perspective-n-Point Problem, *in* '2021 40th Chinese Control Conference (CCC)', IEEE, Shanghai, China.

Park, F. C. and Martin, B. J. (1994), 'Robot sensor calibration: solving AX=XB on the Euclidean group', *IEEE Transactions on Robotics and Automation* **10**(5), 717–721.

Peeblesshire (2022), 'Erling Haaland helps himself to two goals as Manchester City crush FC Copenhagen', *Peeblesshire News* . Date accessed: November 21, 2023.

Person, J. (2023), '5 things to know before customizing your first machine learning model with MediaPipe Model Maker', https://developers.googleblog.com/en/5-things-to-know-before-customizing-your-first-machine-learning-model-with-mediapipe-model-maker/. Date accessed: May 8, 2024.

Pertuz, S., Pulido-Herrera, E. and Kamarainen, J.-K. (2018), 'Focus model for metric depth estimation in standard plenoptic cameras', *ISPRS Journal of Photogrammetry and Remote Sensing* **144**, 38–47.

Plazaola, A. and Agustín, J. (2016), Implementation of a Visual Servo Control in a Bi-Manual Collaborative Robot, Technical report.

Rocco, P. (2024), 'Control of industrial robots', https://rocco.faculty.polimi.it/cir/index.html. Date accessed: May 27, 2024.

Rocco, P. (n.d.), 'Control of industrial robots - Collaborative robotics', https://rocco.faculty.polimi.it/cir/Collaborative%20robotics.pdf. Date accessed: May 27, 2024.

Rublee, E., Rabaud, V., Konolige, K. and Bradski, G. (2012), ORB: An efficient alternative to SIFT or SURF, *in* '2011 International Conference on Computer Vision', IEEE, Barcelona, Spain.

Ruiz, P. and Tibthat, K. (2023), 'Introducing MediaPipe Solutions for On-Device Machine Learning', https://developers.googleblog.com/2023/05/introducing-mediapipe-solutions-for-on-device-machine-learning.html. Date accessed: February 7, 2024.

Sellat, Q., Bisoy, S. K. and Priyadarshini, R. (2022), Chapter 10 - Semantic segmentation for self-driving cars using deep learning: a survey, *in* H. K. Tripathy, P. K. Mallick, A. K. Sangaiah, G.-S. Chae and S. Mishra, eds, 'Cognitive Big Data Intelligence with a Metaheuristic Approach', Cognitive Data Science in Sustainable Computing, Academic Press, pp. 211–238.

Shirai, Y. (1971), 'Guiding a robot by visual feedback in assembling tasks', *Pattern Recognition* **5**(2), 99–106.

SINTEF (2023), ‘HumanTech’, https://www.sintef.no/prosjekter/2021/humantech/. Date accessed: November 30, 2023.

Solá, J. and Deray, J. (2018), ‘A micro Lie theory for state estimation in robotics’.

Srinivasan, P. P., Garg, R., Wadhwa, N., Ng, R. and Barron, J. T. (2018), Aperture Supervision for Monocular Depth Estimation, *in* ‘2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition’, IEEE, Salt Lake City, UT, USA, pp. 6393–6401.

Tahri, O. and Chaumette, F. (2005), ‘Point-based and region-based image moments for visual servoing of planar objects’, *IEEE Transactions on Robotics* **21**(6), 1116–1127.

Tompson, J., Stein, M. M., Lecun, Y. and Perlin, K. (2014), Real-Time Continuous Pose Recovery of Human Hands Using Convolutional Networks, Vol. 33.

TTK21 Introduction to Visual Simultaneous Localization and Mapping - VSLAM (2023), https://www.itk.ntnu.no/emner/fordypning/ttk21. Date accessed: October 24, 2023.

Uboweja, E., Tian, D., Wang, Q., Kuo, Y.-C., Zou, J., Wang, L., Sung, G. and Grundmann, M. (2023), On-device Real-time Custom Hand Gesture Recognition, *in* ‘2023 IEEE/CVF International Conference on Computer Vision Workshops, ICCVW 2023’, Paris, France, pp. 4275 – 4279.

UniversalRobots (2023), ‘USER MANUAL - UR10 CB-SERIES’, https://www.universal-robots.com/download/manuals-cb-series/user/ur10/315/user-manual-ur10-cb-series-sw315-english-international-en/. Date accessed: December 15, 2023.

Vemprala, S. H., Bonatti, R., Bucker, A. and Kapoor, A. (2023), ‘ChatGPT for Robotics: Design Principles and Model Abilities’.

Voxel51 (n.d.*a*), ‘Available Zoo Datasets’, https://docs.voxel51.com/user_guide/dataset_zoo/datasets.html#coco-2017. Date accessed: March 14, 2024.

Voxel51 (n.d.*b*), ‘COCO Integration’, https://docs.voxel51.com/integrations/coco.html. Date accessed: March 14, 2024.

Wang, W., Li, R., Diekel, Z. M., Chen, Y., Zhang, Z. and Jia, Y. (2019), ‘Controlling object hand-over in human-robot collaboration via natural wearable sensing’, *IEEE Transactions on Human-Machine Systems* **49**(1), 59–71.

Wu, J.-F. and Cheng, M.-Y. (2018), Depth estimation of objects with known geometric model for IBVS using an eye-in-hand camera, *in* ‘2017 International Conference on Advanced Robotics and Intelligent Systems (ARIS)’, IEEE, Taipei, Taiwan, pp. 88–93.

Wu, Y.-c. and Feng, J.-w. (2017), ‘Development and Application of Artificial Neural Network’, *Wireless Personal Communications* **102**, 1645–1656.

Wu, Y., Wang, Y., Jung, S., Hoermann, S. and Lindeman, R. W. (2019), ‘Towards an articulated avatar in VR: Improving body and hand tracking using only depth cameras’, *Entertainment Computing* **31**.

Xiao, H. and Chen, X. (2019), Following Fast-Dynamic Targets With Only Slow and Delayed Visual Feedback: A Kalman Filter and Model-Based Prediction Approach, *in* ‘ASME 2019 Dynamic Systems and Control Conference’, Park City, Utah, USA.

Yuan, S., Garcia-Hernando, G., Stenger, B., Moon, G., Chang, J. Y., Lee, K. M., Molchanov, P., Kautz, J., Honari, S., Ge, L., Yuan, J., Chen, X., Wang, G., Yang, F., Akiyama, K., Wu, Y., Wan, Q., Madadi, M., Escalera, S., Li, S., Lee, D., Oikonomidis, I., Argyros, A. and Kim, T.-K. (2018), Depth-Based 3D Hand Pose Estimation: From Current Achievements to Future Goals, *in* ‘2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition’.

Zhang, F., Bazarevsky, V., Vukanov, A., Tkachenka, A., Sung, G., Chang, C.-L. and Grundmann, M. (2020), ‘MediaPipe Hands: On-device Real-time Hand Tracking’, *ArXiv* **abs/2006.10214**.