Università Politecnica delle Marche

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Meccanica
Curriculum Meccatronica

Riconoscimento e anticipazione delle azioni umane in uno scenario di assemblaggio collaborativo uomo-robot

Recognition and anticipation of human actions in a human-robot collaborative assembly scenario

Candidate:
Giorgio Zoppi

Advisor:
Prof. Giacomo Palmieri

Coadvisor:
Prof. Pedro Neto

Academic Year 2023-2024

Abstract

The purpose of integrating robots into human environments is to simplify human life, by assisting in various activities and enhancing human capabilities. The current trend is shifting from autonomous robots to collaborative robots, commonly referred to as “cobots”, that share space and collaborate with humans. In manufacturing, cobots offer flexibility, improved occupational health, with both greater ergonomics and human factors, lower costs, and efficient use of space, by leveraging human experience alongside robotic precision, repeatability, and strength. A key challenge in this domain is improving human-robot interaction, specifically through the robot’s understanding of human actions.

This research aims to develop a computer vision system integrated with a collaborative robotic arm to recognize, predict, and anticipate human actions in a collaborative assembly scenario. The goal has been to create a lightweight autonomous real-time system that enhances human-robot collaboration, by investigating various AI-based technologies. The study utilized a pneumatic cylinder as an example component for the assembly, along with the KUKA LBR iiwa 7 R800 robotic arm, the SCHUNK Co-act EGP-C 64 gripper, and the Intel RealSense D435i camera. The core of the research involved developing algorithms for human action recognition, prediction, and anticipation, using neural networks, computer vision, and real-time robot control systems.

The results showed excellent performance of the offline models and very good performance in the autonomous real-time system implementation. In particular, the action recognition model, which has been designed and developed from scratch, as well as the dataset used to train it, showed an accuracy of 95% offline and, qualitatively, over 90% online. Moreover, the successful use of the action prediction model trained on real assembly sequence data, obtained from human operator demonstrations, enabled the creation of a system with no prior knowledge of the assembly sequence. This research demonstrates the potential of advanced computer vision and machine learning techniques in enhancing human-robot collaboration. By enabling cobots to recognize, predict, and anticipate human actions, significant progress is being made in creating more intuitive and efficient collaborative environments in manufacturing. The findings represent a major step toward a future where humans and robots collaborate safely and profitably across various sectors.

Sommario

L'obiettivo dell'integrazione dei robot negli ambienti umani è quello di semplificare la vita delle persone, assistendole in varie attività e migliorandone le capacità. L'attuale tendenza sta passando dai robot autonomi ai robot collaborativi, generalmente denominati "cobots", che condividono spazi e collaborano con gli esseri umani. Nel settore manifatturiero, i cobots offrono flessibilità, miglioramento della salute occupazionale, con una maggiore ergonomia e fattori umani, riduzione dei costi e uso efficiente dello spazio, sfruttando l'esperienza umana assieme alla precisione, ripetibilità e forza dei robot. Una sfida chiave in questo ambito è migliorare l'interazione uomo-robot, in particolare attraverso la comprensione da parte del robot delle azioni umane.

Questa ricerca si propone di sviluppare un sistema di computer vision integrato con un braccio robotico collaborativo per riconoscere, prevedere e anticipare le azioni umane in uno scenario di assemblaggio collaborativo. L'obiettivo è stato quello di creare un sistema real-time leggero ed autonomo che migliorasse la collaborazione uomo-robot, indagando su varie tecnologie AI-based per la sua realizzazione. Per lo studio, sono stati utilizzati: un cilindro pneumatico come componente assemblato di esempio, il braccio robotico KUKA LBR iiwa 7 R800, il gripper SCHUNK Co-act EGP-C 64 e la telecamera Intel RealSense D435i. Il cuore della ricerca ha coinvolto lo sviluppo di algoritmi per il riconoscimento, la previsione e l'anticipazione delle azioni umane, utilizzando reti neurali, visione artificiale e sistemi di controllo del robot in tempo reale.

I risultati hanno mostrato prestazioni eccellenti dei modelli offline e prestazioni molto buone per l'implementazione del sistema autonomo real-time. In particolare, il modello di riconoscimento delle azioni, che è stato progettato e sviluppato da zero, così come il dataset utilizzato per addestrarlo, ha mostrato un'accuratezza del 95% offline e, qualitativamente, superiore al 90% online. Inoltre, il successo nell'utilizzo del modello di previsione delle azioni, addestrato su dati di sequenze di assemblaggio reali, ottenute da dimostrazioni dell'operatore umano, ha permesso di creare un sistema senza alcuna conoscenza preliminare della sequenza di assemblaggio. Questa ricerca dimostra il potenziale delle tecniche avanzate di computer vision e machine learning nel migliorare la collaborazione uomo-robot. Permettendo ai cobots di riconoscere, prevedere e anticipare le azioni umane, si compie un significativo progresso verso la creazione di ambienti collaborativi più intuitivi ed efficienti nel settore manifatturiero. I risultati rappresentano un passo importante verso un futuro in cui esseri umani e robot collaborano in modo sicuro e proficuo in vari settori.

Contents

Chapter 1

Introduction

The purpose of the robotic existence among humans is to make the human life as straightforward as possible. The robots are supposed to assist humans in their activities and to improve their capabilities.

Today's paradigm is shifting, from robots that operate autonomously, with only human supervision, to collaborative robots, also commonly named as "cobots". These robots are able to share space with other robots and, most importantly, with humans. They are becoming increasingly more safe and intuitive to use, with simplified human-robot interfacing methods and decision making capabilities. cobots can interact physically and cognitively with humans, assuming different roles: learners, teachers, evaluators, collaborators, etc. There is no doubt that collaborative robots will be an integral part of our near future, where they will co-exist and actively collaborate with humans.

Talking about the field of manufacturing, the collaboration between human and robot leads to many benefits, the first of which is flexibility. Collaborative robots can be used to exploit the human potential, leveraging human experience and intellect combined with the precision, strength, and efficiency that characterizes robots. This translates into more occupational health for humans, with both greater ergonomics and human factors, lower costs, and efficient use of factory space [1].

Collaborative robotics is currently facing new challenges to make it easier for humans to interact with cobots. This includes finding better ways for humans to communicate with these robots. The main step towards effective communication is for the robot to understand what the human is communicating. This work focuses on the robot's understanding of the signals observable from the human body, in particular the actions performed with the hands, in order to develop more intuitive and natural human-robot communication methods.

1.1 Motivation

The services that a robot can provide are appreciated when they are offered at the right time and require little input from humans. If the interacting human needs to know prerequisites in order to interact with the robot then the level of interaction is less natural as compared to the one that does not demand any prerequisite for interaction [2].

In manufacturing industries, industrial robots are currently carrying out dangerous, dirty, and repetitive tasks excellently. These robots are isolated by cages or barriers, and human intervention is limited to the choice of settings and programming the working routines of the robots. On the other hand, there may be tasks that require both power and intelligent decision making, but the latter is usually hard to accomplish by the robot itself, even in tasks that are basic for humans [3]. Therefore, human and robot need to work together, coexisting and collaborating in the same operative space. This usually reduces the robot's level of autonomy, especially autonomous acting compared to industrial robotics applications, in order to improve the team's performance [4, 5].

The design of many collaborative applications nowadays are not human-centered but, instead, they are robot-centered [6]. In this way, the human is part of the automation, not the actual controller. The opposite situation is when the robot is used only as a tool, assisting human movements or doing actions when commanded, i.e., teleoperated robots. In this case, the human is definitely the controller of the interaction and it can be very useful in some application fields, such as carrying heavy weights or precision surgery [7], but for many other applications a lot of the potential in terms of cognition and reasoning that could be made by the robot is being lost [8]. Even speaking in terms of costs, using a collaborative robot as a tool is very expensive unless the task is too challenging to be done by just humans. There exists other "less intelligent" machines that can be used instead of a robot is such tasks, for example devices that assist lifting weights on assembly line [9]. The main challenge in developing and integrating a human-in-the-loop collaborative process is to find novel approaches for robots to proficiently interact with humans in a safe, intuitive, and also low-cost manner.

Thinking about how humans interact with each other during manual jobs, like in manufacturing, the human voice and gestures could be a fundamental way to communicate. Although in the past years the interaction with smart objects through voice has been much pursued, for example in home automation, this does not seem to be the ideal solution in factory applications. This is due to the noise and sound interference of many kinds that could occur in such environments, as well as the difficulty in indicating specific locations to a robot through the voice. Another human sense that can be excluded is the physical touch, except for accidental touch in collisions, which should be avoided or limited, bringing into play the fundamental aspect of safety. The physical touch is, in general, unnatural even in a human-human interaction during manufacturing jobs and, in most cases, it only occur by error. A collaborative interaction between human and robot is defined “contactless” if there is not physical contact between them, while exchange of forces through objects or tools, is allowed and needed. On the other hand, an application where a collaborative robot should be guided manually is about using the robot as a bare tool. Freedrive operation, where the human operator moves the robot by hand in the desired position, is really common and useful during the programming/training phase, but should not be considered as an available option during the normal execution of the job. The only useful sense that remains in the list is the sight, which has been recently really appreciated in collaborative applications, especially in manufacturing scenarios, with robotic systems that include computer vision systems [10].

A crucial aspect for human-robot interaction is for the robot to estimate and recognize the human actions [11]. This means that the robot, using its sensors, should be able to understand the exact action performed by the human operator in every moment. Discerning human actions is a huge challenge on which much of the research in collaborative robotics is focusing lately.

If the robot can understand human actions and consequently react, the human can be the actual controller of the interaction and not being one of its parts, taking real advantage of the robot potential in collaborative applications. This correspond to emulate the way humans usually interact, as they infer one another’s intent from observed actions in order to efficiently and safely collaborate [12].

As just mentioned, recognizing actions is useful if followed by a reaction by the robot. The robot reactions can be based on data obtained up to the current moment or, in more advanced applications, they can be based on predictions of the future, resulting in even more natural behavior. A relative easy example of robot reaction is the replanning of robot’s motion paths to avoid collisions with human operators and the surrounding environment, that is known as “collision avoidance”. Methods and algorithms to execute it have reached a good level in research and successful applications [13]. Advanced algorithms are able to perform motion prediction and therefore avoid collisions more effectively. As will be highlighted in Sec. 2.5, the human movement prediction can be split into short and long time horizons. The latter group is the one related action prediction, as they represent goal-based human motions over extended time horizons.

The recognition of human actions has to be done in a proactive way to be effective. By recognizing actions quickly, predictions of future actions can be applied in time to generate consistent reactions from the robot. One of the most useful uses that can be made with actions predictions is to anticipate them, letting the robot complete actions instead of the collaborating human, even before he/she starts to perform them. This requires speed and effectiveness in data processing, reliable decision-making and safety in the execution of actions, all of which are complex aspects.

1.2 Objectives

The main objective of this work is to develop a computer vision system, integrated with a collaborative robotic arm, to recognize, predict and anticipate human actions in a collaborative assembly scenario, investigating among different AI-based technologies to see which one best fulfills the purposes. These components have then been combined into a single program to realize an autonomous real-time system. For this reason, all its parts must be sufficiently light and fast to allow optimal real-time implementation.

The focus has been on achieving the most natural, intuitive, and flexible human-robot collaboration possible. In this way, human capabilities that do not belong to a robot, such as domain experience, dexterity, and ability to deal with unexpected situations, can be exploited in a robotic application. By integrating the developed collaborative robotic system with these human capabilities, the working conditions of the human operator are improved, as well as efficiency and accuracy.

1.3 Problem definition

The work has been focused on the assembly of a pneumatic cylinder, which has been taken as a useful example component because of certain characteristics, described in the dedicated Sec. 3.1. Among the pneumatic cylinder's parts, some object classes have been identified and used to train an object detection system to recognize them in the real-time application. At the same time, a list of actions that a human operator would normally perform during the assembly of a component like the one under study has been defined. Appropriate tools and scripts have been developed to collect and process this data.

Then, a neural network has been developed to recognize the defined actions, based on the data provided by a pose detection model, which uses frames captured by the camera. Another neural network has been structured to predict future human actions, based on the recognized actions, objects data and assembly sequence data relative to the component. In this way, the robot will be able to anticipate these actions and effectively assist the human operator in the assembly process. The information about the objects provided by the object detector is used in combination with a specially developed object tracker. This information has also been used to investigate the performance of its inclusion in the action recognition model, with the goal of helping to classify actions based on the objects being used.

It is worth noting how recognition, prediction and anticipation of human actions follow a sequential dependency structure. For this reason, action recognition is of fundamental importance, as the other two steps are built upon it.

1.4 Overview

The current work is organized to start with a broad view of collaborative robotics and how robot perception is important for achieving optimal levels of naturalness and intuitiveness in human-robot interaction, before narrowing towards the main objectives of the work.

Chapter 2 presents the state-of-the-art, describing the current collaborative robotics applications and research, and the most common approaches to human action recognition, prediction, and anticipation. The most popular and innovative approaches to object detection and object tracking are also presented.

In Chapter 3, materials and methods used in the related work are described. Here the structures of the developed neural networks and the tools used to acquire and process the data will be discussed in detail.

In Chapter 4, the results of the work will be presented. Special attention will be paid to the key performance metrics related to the testing of neural networks, both offline and online. The performance of the overall system will also be analyzed.

Finally, Chapter 5 summarizes the work and the conclusions drawn from it, while also discussing some possible future developments.

Chapter 2

State of the art

In this chapter, the state-of-the-art of collaborative robotics is reported, with a special focus on robot perception, which is essential to enable conscious decision making by a robot. The key differences between industrial and collaborative robots will be pointed out. The state-of-the-art of human action recognition and object detection, which are two of the most challenging computer vision topics at the moment, will be described as well. Finally, particular attention will be paid to human action prediction and anticipation.

2.1 Industrial robotics

Industrial robots are designed to perform operations quickly, repeatedly, and accurately. They have been well established in the manufacturing industry for over thirty years. They are employed in repeated tasks, that can even be too heavy or dangerous for humans, with slight intellectual demands but where efficiency and continuity are needed. The most common tasks are: handling, assembling, casting, painting, sorting, and welding [14]. They can be defined as a programmable device, consisting of electronic and mechanical parts, that can perform complex actions autonomously. They are usually large, heavy and made of rigid materials. Normally, industrial robots are application specific, isolated from human workforce through protective barriers and possess their own working space. The isolation of an industrial robot is necessary for the safety of people inside the factory and is governed by specific regulations [15]. The setup of a cell with industrial robots can take a long time and it needs to be carefully tuned before real production can begin. This makes applications with industrial robots highly expensive and not flexible, useful just for mass production, where the large number of units that will be produced more efficiently allows the cost to be amortized [16].

2.2 Collaborative robotics

Collaborative robots, also referred as cobots, as opposed to industrial robots, are designed to work along with humans and share the same working space, as co-workers would do. These type of robots are, in general, smaller and lighter in weight as compared to industrial robots, so they offer great mobility. However, the real benefit brought by collaborative robots is their high flexibility [17]. Collaborative robots can be used to perform a variety of tasks, as they are easy and intuitive to be re-programmed, even by non-expert users. In an industry that is evolving from mass production to mass customization paradigm, the flexibility to change the tasks to be performed will be the fundamental element, even more than reduced production times [18].

2.2.1 Human-robot collaboration

Collaboration between humans and robots is usually defined in literature as Human-Robot Collaboration (HRC). HRC is sometimes distinguished from Human-Robot Interaction (HRI), even if they are similar, which is referred to the pure interaction between human and robot, not necessarily with a common task to complete [19]. This distinction is not important for the purpose of the present work, so they will be considered the same and will be referred to as HRI/C. HRI/C can be classified in many different ways. For instance, in [20] a classification in eleven categories is proposed, considering task type, robot morphology, interaction roles, space, and time. More recent classification, proposed a nested framework consisting of three levels of interaction between a human and a robot, where any level of engagement requires that the features of lower levels of interaction are guaranteed. The three levels of interactions, ordered from the internal to the external in the nested framework, are: safety, coexistence, and collaboration [21].

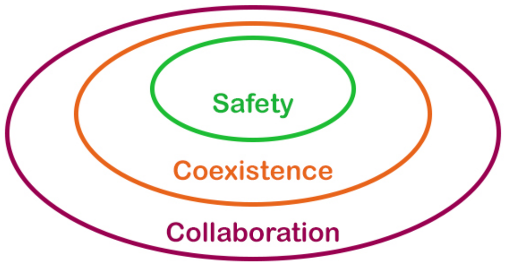


Figure 2.1: Nested levels classification for HRI/C [21]

- Safety: this is the most leading criteria to guarantee in a human-robot close coexistence in an industrial environment. Afterwards, there is a section dedicated to the state of the art of safety regarding collaborative applications, also reporting the current challenges;

- Coexistence: it is defined as the robot capability of sharing the workspace with humans and other entities, without any cage or barrier separating them. This does not include simultaneous interaction on the same object or coordination of actions and intentions between robot and human;
- Collaboration: it is the robot peculiarity of being able to perform complex tasks with direct human interaction and coordination.

An additional crucial aspect of HRI/C is the trust that the human places in the robot. Indeed, establishing trust between humans and robots is a prerequisite for building a society where humans and robots collaborate safely and productively in various contexts. The perceived trust in HRI/C depends on human, robot, and environmental factors, with greater relevance of the robot factors, especially its performance [22]. The goal of achieving an adequate level of trust in HRI/C has also been considered in the present study.

2.2.2 Applications

Currently, collaborative robots are being used in different fields of application. For example, in medical applications they are used to carry out or help doctors in performing complex surgical tasks with high accuracy. It has been seen how this results in less invasive and more safe surgeons [23]. The technology should also permit automation of some surgical tasks in the near future, thanks to the acquisition of the performed movements [24]. In rehabilitation applications, cobots can be used as therapeutic systems that help to exercise various range of muscle and joint movements. The flexibility of cobots plays an important role in allowing adaptation to the specific patient, enabling targeted help [25]. Also in education field some collaborative robots are finding employment, as described in this interesting article about a face-to-face computer supported collaborative learning for developing graph construction skills [26]. There are implementations even in military applications, such as exoskeletal systems to assist soldiers carrying military loads [27].

Manufacturing industry represents most of the collaborative robotics applications, especially automotive manufacturing industries and assembly lines in general. There, cobots are implemented to perform numerous tasks ranging from pick-and-placing, packing, palletizing, assembling, welding, product inspection and much more [28, 29]. It helps increasing health and safety of human workers, reducing operating costs and speeding up production cycles.

2.2.3 Research areas

In a collaborative application, collisions should be prevented. A wide field of research, known as collision avoidance, studies algorithms and modalities to avoid the physical contact between robot and human. In almost all tasks robots have to safely navigate from one waypoint to another in the presence of moving entities. Robots need to be able to predict where these dynamic obstacles will be moving in the near future and to avoid them with sufficient speed [30]. Using a human motion detector capable of providing predictions of movements with time parameterization, planners can further optimize robot trajectories in space and time in order to avoid dynamic obstacles with improved economy of motion. This behavior require the ability to reason about obstacle avoidance within both high-dimensional robot configuration space and time domain, which is computationally challenging [31].

In some working modes, like the the Speed and Separation Monitoring (SSM) that will be described in the next section, it is important to determine the robot workspace at any time in order to shape it around the actual robot position and minimize it, leaving more operational space to the human. For example, in [32] it is presented a novel projection-based safety system for ensuring hard safety in human-robot collaboration while maximizing the efficiency, by establishing minimal and well-shaped safety spaces around the robot at any time.

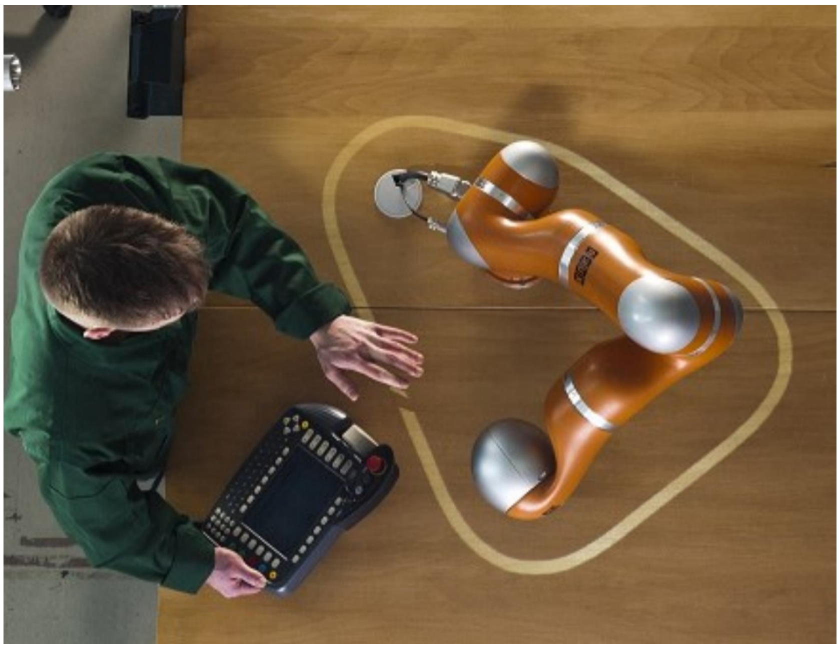


Figure 2.2: Dynamically established safety area based on robot position [32]

2.3 Safety in collaborative applications

In recent years, research in collaborative robotics has intensified, with the focus on making cobots increasingly safer to share workspace with humans while maintaining their characteristic agility and flexibility. As mentioned earlier, collaborative robots are lighter in weight compared to industrial robots. The materials, shape, and surface finish are designed to reduce harm to a human operator in the event of accidental contact. Collisions are prevented through various algorithms of collision avoidance but they could occur, due to the limits of sensors and robot motion capabilities. In this case, the contact has to be detected by the robot's sensors and it should react in a proper way, reducing forces at the impact and absorbing shock [33], or counteracting, depending on the specific application.

International standards consist of IEC standards for electrical fields and ISO standards for non-electrical fields (machinery, management, etc.). ISO/IEC Guide 51:2014 [34] is a set of guidelines for the development of safety standards, where they are classified in three types:

- Type A: basic safety standards, common to all machinery;
- Type B: group safety standards, for wide range of machinery;
- Type C: product safety standards, for detailed safety requirements for a particular machine or group of machines.

2.3.1 Collaborative working modes

Going into detail about safety for collaborative robots, there are four collaborative operative modes identified by robot safety standards 10218-1/2:2011 [15] and the technical specification ISO/TS 15066:2016 [35]. Descriptions and a graphic illustration, both extrapolated from [36], follow:

1. Safety-rated Monitored Stop (SMS): it is the simplest type of collaboration. The working area is shared between the human and the robot but they cannot work at the same time, since the robot is not allowed to move if it detects any entity in the operative space. This working mode is suitable for manual placement of objects to the robot's end-effector, in visual inspection, for finishing operation or complex operations when human presence is required, or when the robot can assist the operator to position heavy parts. Compared to traditional safety stop functions, SMS requires the additional retaining stopping function named "Stop

Category2", which is a safety-rated monitored stop leaving power available to the machine actuators after the movement ends [37]. Accordingly, when the human enters the collaborative area, the robot undergoes a safe standstill mode and its movement is paused through dedicated redundant software and electronics-based safety technology. At the same time, the robot automatic cycle remains active and the program continues from interruption point after the worker has left the collaborative area. These functionalities are integrated in cobots.

2. Hand Guiding (HG), also known as "direct teach" or "freedrive": it is a collaborative mode where the operator can teach the robot positions by using physical HRI/C, without the need for an intermediate interface. During the hand guiding of the robot, its weight is compensated to hold its position. This collaborative working mode requires robots equipped with both safety-rated monitored stop and safety-rated monitored speed functionalities. While the robot is inside the collaborative area, it executes the program in automatic mode; if the operator approaches this area, the robot program and movements interrupt. As the operator activates the hand guiding device, the robot state switches to safety-rated monitored speed functionality to allow direct movement of robot. When the operator releases the hand guiding device, the robot returns in safety-rated monitored stop and resumes previously interrupted program as soon as the operator leaves the collaborative area.

3. Speed and Separation Monitoring (SSM), also indicated as Speed and Position Monitoring (SPM): it allows the human presence within the robot's space through safety-rated monitoring sensors. The surrounding space near to the robot is divided into three concentric zones with different dimensions, centered on the robot. The robot operates at full speed when the human is in the the largest zone, at reduced speed in the medium zone, and it stops stop when the human moves into the smallest zone. These areas are inspected with scanners or a vision system. The smallest zone is always larger than the reachable area of the manipulator, so the medium zone is out of the robot's range. However, the robot is slowed down to a safe speed in the this zone as the operator could be endangered with a dropped manipulated object or some of its separable parts.

4. Power and Force Limiting (PFL): it is a collaborative mode where the motors power and force are limited in order to allow a side-by-side collaboration between human and robot. This working mode requires dedicated equipment and control models for handling collisions between the robot and the human with no harmful consequences. In this scenario, collisions might happen, so it is important to carry out an in-depth risk assessment [38] and to investigate the injury levels

caused by a possible collision with the robot [39]. These topics will be described in detail in the next section. Another important aspect is the reaction of the robot in response to the contact. The most obvious solution is activating robot's brakes after collision with immediate stop. Torque control mode with gravity compensation and admittance reflex are improved strategies, which result in a safer behaviour such as decreasing the impact energy through counter-motion in the opposite direction. Many other types of reactions, some of which are really complex, that try to reduce the injury level caused by the collision, are being studied recently. It is worth noting that a combination of PFL and SSM could used in order to exploit efficiency and productivity, while still preserving safety of the human operators, as mentioned in [40].

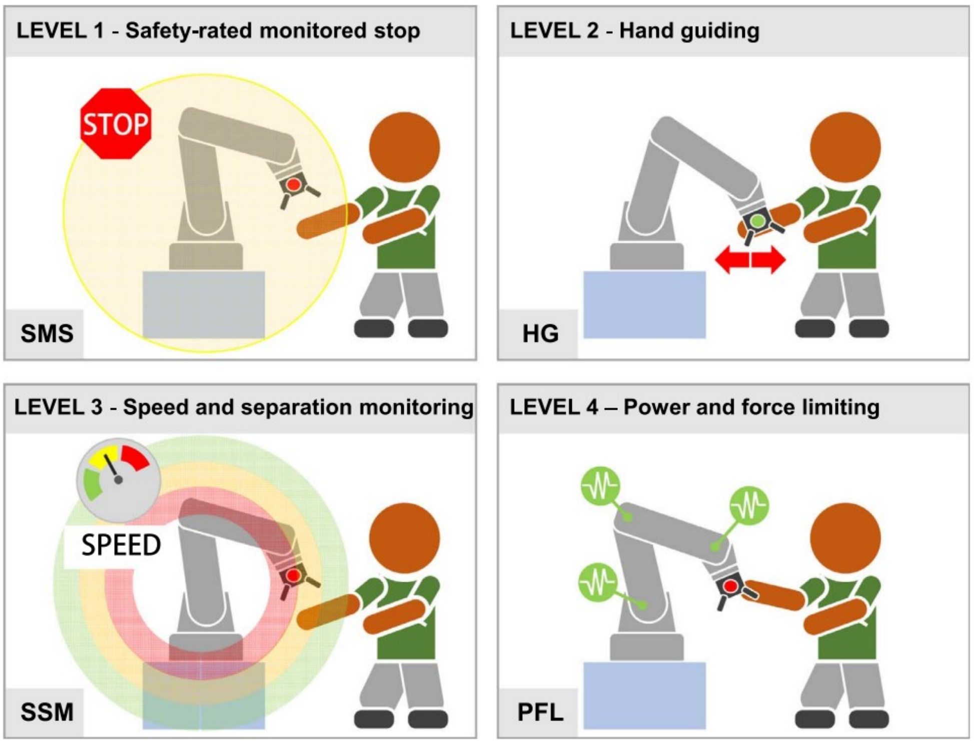

LEVEL 1 - Safety-rated monitored stop
STOP
SMS
LEVEL 2 - Hand guiding
HG
LEVEL 3 - Speed and separation monitoring
SPEED
SSM
LEVEL 4 – Power and force limiting
PFL

Figure 2.3: Collaborative working modes identified by robot safety standards [36]

The PFL collaborative working mode is recently the preferred mode to be used in factories, as it allows for the closest collaboration between humans and robots. This operative mode has been used in the related work.

2.3.2 Risk assessment and contact hazards

The general risk assessment procedure of machinery is described in ISO 12100:2010 [41] and more specific standard for robots in ISO 10218-1/2:2011 [15]. This involves three steps that have to be repeated until the residual risk of hazards is under an acceptable level [38]. The first step within a risk assessment is a risk analysis, which is the identification of all risks or hazards that may occur during all kind of operation modes combined with all possible activities. Unintended behaviour of the operator or unpredictable behaviour of the robot system must be taken into consideration. Most typical hazards are mechanical ones like bruising, cutting, clamping or impacts. There can also be electrical or thermal hazards, or additional risks due to noise, vibration, radiation, chemicals and many others. The risk analysis is followed by a risk evaluation of detected hazards. Here, probabilities and extents of damages or injuries are assessed. From this, the last step is the risk reduction, which is operated with diverse protective measures or safeguards. The obtained residual risk is compared to the starting risk from the risk analysis and the process is repeated until significant improvements can no longer be made.

Talking about the contact hazards, the technical specification ISO/TS 15066:2016 [35] introduces the allowable collision peak pressure which should not be exceed in the case of the human-robot collision. It is well-known that the peak pressure varies depending on the shape of the robot part the impact (edges, corners, rounded fillet, etc.). The smaller the contact area, the higher the pressure, for the same collision speed and effective mass of the robot [42]. Also the dynamic of the impact is relevant, depending on the intensity of inertial effects during the contact, so the depending on the mass and the speed [43].

2.4 Human action recognition

Understanding human motion is a challenging task for robots since it is highly non-deterministic, uncertain, multi-modal and influenced by various factors in the environment [44]. Usually, human movements are aimed at performing an action, which therefore can be seen as a set of movements with a certain pattern and a specific goal. In this case, we can speak of primitive actions [45], even if they are composed of multiple movements, to distinguish them from more complex actions composed of multiple primitive actions. Examples of everyday life human primitive actions are: walking, picking an object from the table, opening a door, writing with a pen, etc. Each of these actions is composed of movements of various parts of the human body. By correlating these movements in space and time it is possible to determine the human action that is being performed. In manufacturing environments, we are more interested in primitive actions related to the specific production activities carried out by the factory. For example, in an assembly scenario, which is the one considered in related work, notable primitive actions could be: pick, place, push, pull, screw, etc.

In literature, the term "human action" is sometimes associated with "human intention", meaning sets of human movements over extended time spans aimed at specific goals [46], which coincides with the definition of human action described so far and which will continue to be referred to by this name. Another commonly used term is "human motion intention", defined as the trajectory of a human body part [3, 47]. This is related to motion-level analysis over short time horizons and it is not relevant for the related work.

To assist humans in collaborative tasks, it is crucial for robots to understand their actions, based on their perceptions [48]. Humans can naturally discern actions performed by other humans when interacting with them, while it is a very difficult task for robots. Sensors are the means by which robots can perceive. In particular, computer vision systems can be used in conjunction with a robot to provide it with vision, which, as mentioned above, is the preferred mode for carrying out human-robot interaction in recent years.

2.4.1 Perception

Human action recognition can be classified as part of the robot's perception. Perception is one of the main elements in making the robot collaborate with humans. The robot must be able to perceive surroundings by understanding the data it receives from sensors. This is essential for the human-robot interaction, as the robot must be able to recognize the objects involved in the task, i.e., object detection, detect the human it is collaborating with and human actions, and avoid operator confusion when several humans are present in the workspace [49].

In the present study, the perception of the computer vision system integrated with the robot is used to detect human poses and objects, which are the two classes of elements most commonly used as data for the models that actually enable robots to effectively understand the environment and interact with it.

2.4.2 Approaches

As explained previously, human actions are specific sets of human movements, so there is a need to start from the acquisition of human movements, and then elaborate them to recognize the actions. This is a very complex problem, which falls into the category of video analysis. Video analysis involves image analysis for elaborating single frames and memory mechanisms to link information temporally, i.e., sequential data analysis or time-series data analysis.

Classical approaches to action recognition, and time series analysis in general, include Dynamic Time Warping (DTW), which is used for its robustness to variations in the speed or style of the action being performed [50]. Another approach is to use Gaussian Mixture Models (GMMs), as they are able to cluster data into different groups as a collection of multivariate Gaussian distributions, with a Hidden Markov Model (HMM) used to capture temporal relationships [51].

When analyzing sequences, neural networks with memory mechanisms are usually used, namely the Recurrent Neural Networks (RNNs). In general, basic RNNs are not performing well in capturing medium-long term dependencies in sequences, as they suffer from gradient vanishing problem [52]. For this reason, the two RNN's main variations are used: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). It has been experimentally proven that LSTM and GRU have higher validation accuracy and prediction accuracy than the standard RNN. The list below describes the architecture, capabilities, and use cases, including Transformers, which are not properly recurrent neural networks, but they are still designed to handle sequential data:

- Recurrent Neural Network (RNN): they are fundamental sequence models that process sequences iteratively, using the output from the previous step as input for the current step. The recurrent connections allow for the retention of memory from previous time steps, being able to learn features and long term dependencies from sequential and time-series data [53]. Plain RNNs present a limited memory span due to their simple structure, making it hard to learn long-term dependencies. This is why they are less common in modern applications.

- Long Short-Term Memory (LSTM): they are an extension of standard RNNs, designed to better capture long-term dependencies in sequences [54]. They use gates (input, forget, and output) to regulate the flow of information and they have a cell state in addition to the hidden state to carry information across long sequences. They are really appreciated for long-context tasks such as intention recognition, speech recognition, sentiment analysis, etc.

- Gated Recurrent Unit (GRU): they are a variation of LSTMs with a simplified gating mechanism. They present only two gates (reset and update gates) and merge the cell state and hidden state [55]. They might be less efficient than LSTMs in capturing some long-term dependencies but they are faster to train and with a simplified model structure. Usages are the same of LSTMs.

- Transformers: they cannot be properly classified as recurrent neural networks, but they focus on self-attention mechanisms to process data in parallel and to weight the importance of different parts in the input data [56]. They are particularly appreciated for machine translation and text summarizing. One use among all is for the famous GPT.

Speaking of image analysis, Convolutional Neural Networks (CNNs) are the first choice when it comes to image processing and performing predictions on images, as they excel at such tasks. CNNs analyze the image pixel-level, working with kernels, called filters, that go over the image and generate feature maps to find graphic correlations between images belonging to the same class or linked to a particular detail to extract [57]. CNNs are recently being used to process 2D and 3D human pose estimation from RGB and RGB-D images, with great results [58].

CNNs usually process single images, so there is a need to link the information obtained from each of them in the sequence, i.e., the video, and to consider the temporal correlation. By using just CNNs, without any other network with memory mechanisms, these are the most used approaches to do it:

- Late fusion: it is the simplest and most commonly used fusion method. The CNN process every frame of the video and the results are sent to another layer, called the fusion layer, where they are merged. This layer is responsible for the final prediction. The fusion layer can be convolutional too or can contains easier merging approaches such as summation or averaging [59].

- Early fusion: it also rely on extraction of features, but they are merged into a multimodal representation before being processed by the CNN. This approach is more powerful since, if the data is properly aligned, cross-correlations between data items may be exploited, thereby providing an opportunity to increase the performance of the system [60].

- 3D CNN: spatial and temporal information are merged throughout the whole network. This approach is also termed as the slow fusion approach. This is powerful but, at the same time, it is computationally really expensive, which could results in really slow processing [61].

CNNs are also used in pose detectors, which are machine learning models specifically designed to estimate human body pose. Human pose is used in some works for action recognition by detecting local pose features and modelling the spatial configuration between them and also between objects [62, 63]. Of special interest for the present work is MediaPipe, which is a collection of state-of-the-art machine learning solutions for vision, text, and audio tasks, managed by Google [64].

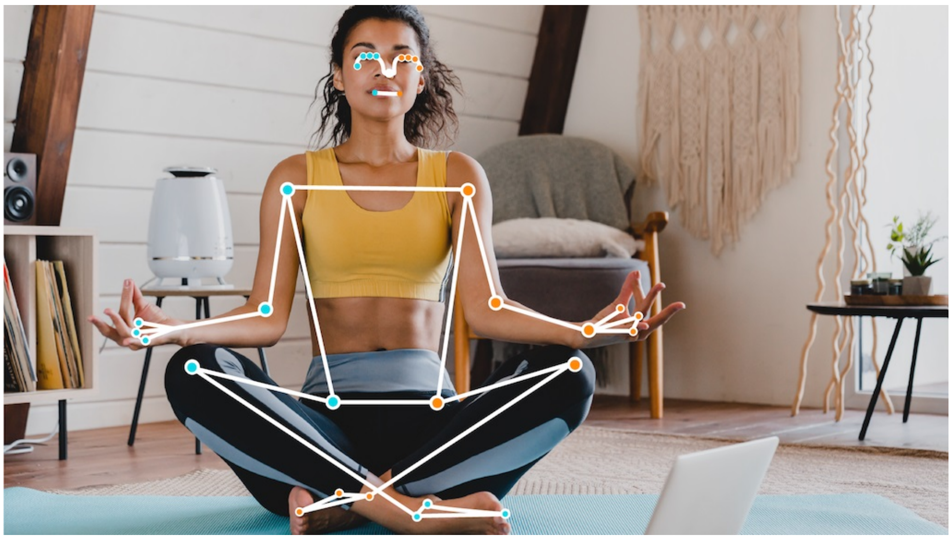


Figure 2.4: MediaPipe pose detector example from official documentation

In particular, the hands tracking solution has been implemented in the related work [65]. More information about this will be provided in Sec. 3.3.1, discussing the data underlying the action recognition model that has been developed.

The human gaze tracking could also be considered for a collaborative application. A study using an open source platform and Pupil Glasses by Pupil Labs, used to acquire 3D estimates of the human's gaze locations could be found in [66]. However, gaze tracking has been excluded from the current study because the intention was to create a system that enhances the natural interaction between human and robot, and the mandatory use of special glasses would not be plausible.

2.5 Human action prediction and anticipation

In literature, the terms "action prediction" and "action anticipation" are often used alternately to refer to the same concept of estimating future actions. In fact, the meaning of the two terms is very similar. In the present study, it was decided to distinguish the two terms more clearly, considering action prediction as the part of estimating future human actions, based on recognized current actions and objects, and action anticipation as the application of those predictions by the robot, which involves estimating predictions deep into the future, decision making logic, and communicating with the robot to actually perform the actions. Therefore, action anticipation is based on action prediction and is always subsequent to it. Together with action recognition, these two operations compose the sequential dependency structure of the robot program: action recognition, action prediction, and action anticipation.

2.5.1 Action prediction

Action perception and recognition allow the prediction of human movement, which, as mentioned, is closely related to human actions. Human movement prediction can be split into two main groups, based on the length of the prediction time horizon. The first group is about short time horizons, usually in the context of reaching motions and collision avoidance, so the robot can predict where the human will be in the immediate future. The second group is related to goal-based human motion prediction over longer time horizons, i.e., human action prediction [67].

Activities often have a hierarchical structure where an activity is composed of a sequence of sub-activities and involve interactions with certain objects. This means that, observing the sub-activities performed and the objects being used by people around us, we are able to predict their future movements and actions. The same behavior is expected by a collaborative robot for an intuitive and natural interaction [62].

Action-level prediction, regarding real-time classification of the action in the early stages of the execution, possible trajectories and targets predictions are common in research [68, 69]. With just this level of prediction, the only anticipation that could be done by the robot is at the single action level, which in most cases means collision avoidance. A common approach to obtain a distribution over possible human navigation trajectories from visual data is the maximum entropy inverse optimal control [70].

Another kind of prediction is at sequence-level, meaning prediction of future actions [71, 72]. This is known in literature as time-series forecasting or time-series prediction. It allows for future trends prediction by analyzing historical data patterns. This type of prediction strictly depends on the action-level one, but also requires data about possible action sequences for the analyzed activity. With this level of prediction, an actual anticipation of future possible actions could be achieved, exploiting the benefits of collaborative robots in a manufacturing environment.

2.5.2 Action anticipation

Anticipation is the other key aspect in human robot collaborative interaction. This is allowed by prediction and it leads to a more natural human-robot interaction. Not every human-human interaction contains anticipations, but in every context where humans interact repeatedly, therefore a routine is implicitly established, anticipation is an expected behavior. For example, when we pay in a store and see that the seller is about to give us the change, we might anticipate his action by holding out an open hand. If we do not do it, therefore we do not use anticipation, the seller will invite us to take the change through his gesture of handing it to us. The latter is not an actual unnatural behavior but the overall time of the interaction is increased and requires a stop that could be avoided. In manufacturing applications, it is essential to reduce working time, so anticipation not only makes interactions more natural but also more efficient. The effectiveness of anticipation is strictly correlated to reaction time of the robot, depending on the analysis and processing time; if the robot reacts to human too late, it might be unable to exhibit anticipatory behavior despite the existence of predictions that include the desired anticipatory information [67].

Works as [62] address action anticipation, but they reduce the robot's action to a binary classification problem on whether or not to perform a predefined action. In other works, such as [73, 74], the robot's anticipation of human actions is extended to a set of possible actions. In this work, only the sequence-level action prediction will be addressed, and it has been done with the aim of giving the robot more decision-making power on action anticipation, making it able to theoretically choose from the same set of recognized actions.

2.5.3 Approaches

Concerning the prediction of actions, a common approach is related to Finite State Machines (FSMs). They are defined as computation models used to model and generate distributions over sets of possible infinite cardinality of strings, sequences, words, etc. [75]. In the case of human action prediction, each unique human action is usually represented by a distinct FSM and the sequences of action are taken into consideration [76].

Another approach particularly used to accomplish human action prediction is the time-series analysis wherein each time step of the motion is encoded as a multivariate Gaussian distribution over the degrees of freedom of a human body part or full body [68].

Also, reinforcement learning, especially reinforcement learning from human feedback (RLHF), is really interesting for robot decision making in predicting actions. It poses different challenges due to the limitations of reinforcement learning in physical real-world environments for inefficiency and safety in long-horizon tasks, as it is impractical to allow robots to engage in unbridled trial-and-error interactions with the physical environment for extended periods [77]. Novel frameworks, such as MAPLE [78], aim to increase deep reinforcement learning by using a pre-defined library of behavior primitives, where the focus is in determining the correct parameters of these primitive skills already known by the robot. This exhibit a high degree of robustness and re-usability for achieving certain manipulation goals, such as picking up objects and moving the robot to a target configuration.

2.6 Object detection and object tracking

A collaborative robot should be able to recognize the objects present in the workspace. Indeed, it has to distinguish between the obstacles to be avoided and the objects that will be manipulated during the collaborative tasks. In addition, thanks to the object detection, the robot can proactively perform collaborative assistance without specific location commands for the objects to be manipulated [49].

The famous object detection algorithm YOLO (You Only Look Once) is considered the state of the art for real-time object detection. This is a complex neural network, with the backbone made of several convolutional neural networks, as usual for image elaboration networks, but with the peculiarity of not being recurrent, as the name suggest. Prior detection systems apply the model to an image at multiple locations and scales, considering detections for high scoring regions, which is computationally intensive. Instead, YOLO apply the network to the full image, dividing it into regions and predicting bounding boxes and probabilities for each region. It is highly efficient and accurate, with an outstanding speed compared to recurrent convolutional neural network (R-CNN) [79, 80]. Since version 6, YOLO exceeded the 400 FPS in the most popular datasets [81]. This means that, with the correct hardware, it can be practically used in any application that requires real-time object detection. In the related work, the most recent version of YOLO was used: YOLOv9 [82], that with his new lightweight network architecture, GELAN, can achieve superior results compared to previous versions.

A key aspect of YOLO, as it represents the model of a neural network and not a closed software, is that it can be trained on custom dataset, even from scratch or fine-tuning a model pre-trained on big datasets of objects in various context. This allow for custom object detection and, with fine-tuning, great results can be reached even with a relatively small dataset of custom objects.

Although YOLO is an exceptionally effective object detector, it does not come with an object tracker by default, as it is a model specialized on the object detection task rather than a closed software. Some object detection methods make use of the temporal information computed from a sequence of frames to reduce the number of false detections [83]. This corresponds to a tracking method, but this is not the case with YOLO, which instead works on the single frame. More in general, object tracking in the realm of video analytics is the critical task that not only identifies the location and class of objects within the frame but also maintains a unique id for each detected object as the video progresses. It is challenging due to factors like abrupt appearance changes and severe object occlusions. In particular, we refer to the Multiple Object Tracking (MOT), or Multiple Target Tracking (MTT), to differentiate it from Single Object Tracking (SOT), which is related to sophisticated appearance models and/or motion models to deal with challenging factors regarding the single object detection, such as scale changes, out-of-plane rotations and illumination variations [84], which are YOLO's concern. MOT is of great interest in the field of computer vision and its tasks are generally partitioned into locating multiple objects, maintaining their identities, and yielding their individual trajectories given an input video [84]. The applications are limitless, ranging from surveillance and security to real-time sports analytics. One famous application is pedestrian tracking, where much of the research in the field is focused given its huge number of practical applications.

The MOT is very complex, and the performance of trackers, even state-of-the-art ones, can vary drastically depending on the application. Numerous researches are focused on defining standardized benchmarks that can compare the performance of multi-object trackers with each other [85]. At the time of this writing, two trackers are directly usable with YOLO with the `Ultralytics` library, which are `BoT-SORT` [86] and `ByteTrack` [87], both with great focus on pedestrians tracking. The library makes them really easy to implement and the performances on the current application have been investigated. After running several tests, the trackers were found to be unsuitable for the application, as they showed deficiency in re-identification after complete obstruction of the object view, which occurs very often with hand interaction in assembly operations. Furthermore, additional logic is needed to handle unseen objects that are presumably being held in the hand. For this reason, a dedicated custom multi-object tracker with the described features has been developed. This will be described extensively in the related Sec. 3.3.5.

Chapter 3

Materials and methods

3.1 Component description

The component chosen for the collaborative assembly scenario to study is the Festo's pneumatic cylinder ADN-40-60-A-P-A. The choice fell on it because of it can be assembled in many different ways, due to very few mandatory assembly precedences. This is useful to properly test the robot program flexibility in predicting the correct next actions. Moreover, it is composed of enough pieces to be valuable for the case study but not too much to become too complex. Pneumatic cylinders such as this are common in research concerning collaborative robotics assembly applications, e.g., [88, 89].

Total parts to be assembled for the considered pneumatic cylinder are 17 and they can be conceptually divided as follows:

- Main body: consists of four parts and four screws. The composing parts are: cylinder barrel, lower cap, upper cap, piston rod with cushion (in one not-divisible piece), and piston nut. Total parts: 9;
- Cylinder supports: double symmetrical L-shape support, upper and lower, with two screws each. Total parts: 6;
- Screwable air connectors: to screw into both cups to connect the air circuit. Total parts: 2.

Fig. 3.1 shows the fully assembled pneumatic cylinder, while Fig. 3.2 shows all the parts and used tools, numbered.

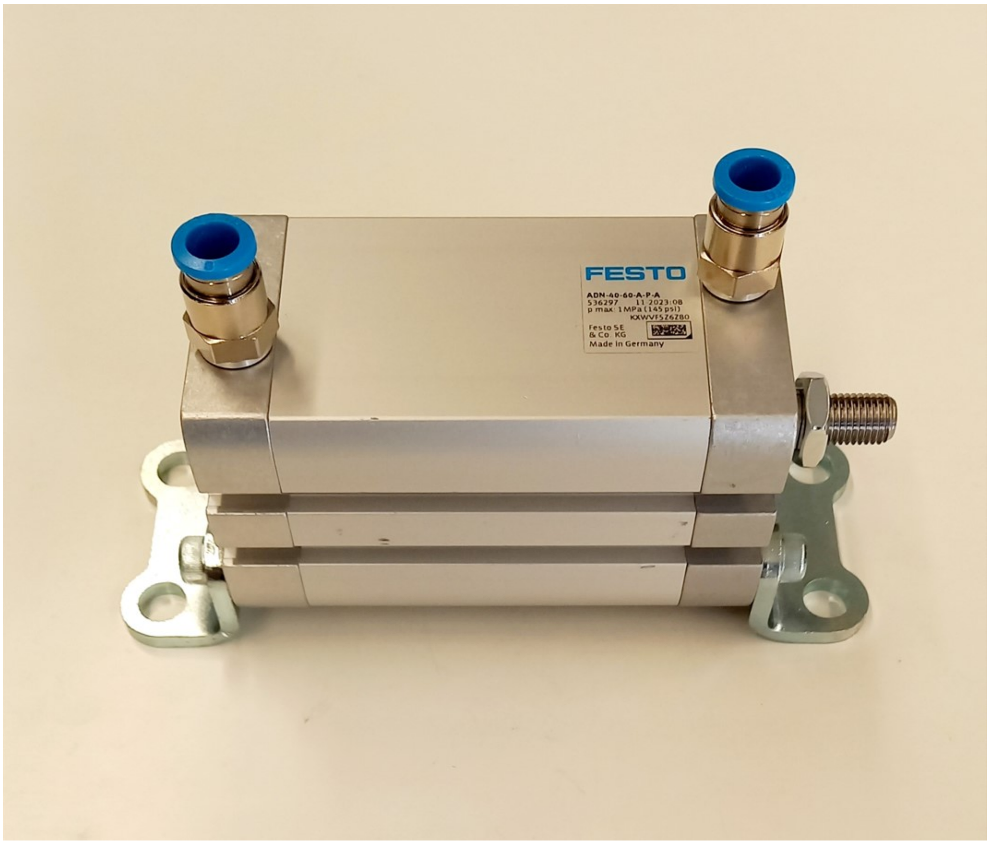


Figure 3.1: Pneumatic cylinder considered in the related work

These parts have been grouped into 10 different object classes, listed below, in which the two wrenches used for screwing have also been added, differentiated by the adjective "big" and "small":

1. Small screw;
2. Big screw;
3. Small wrench;
4. Big wrench;
5. Cap (not differentiated into superior and inferior);
6. Barrel;
7. Piston;
8. Support;
9. Air connector;
10. Nut.

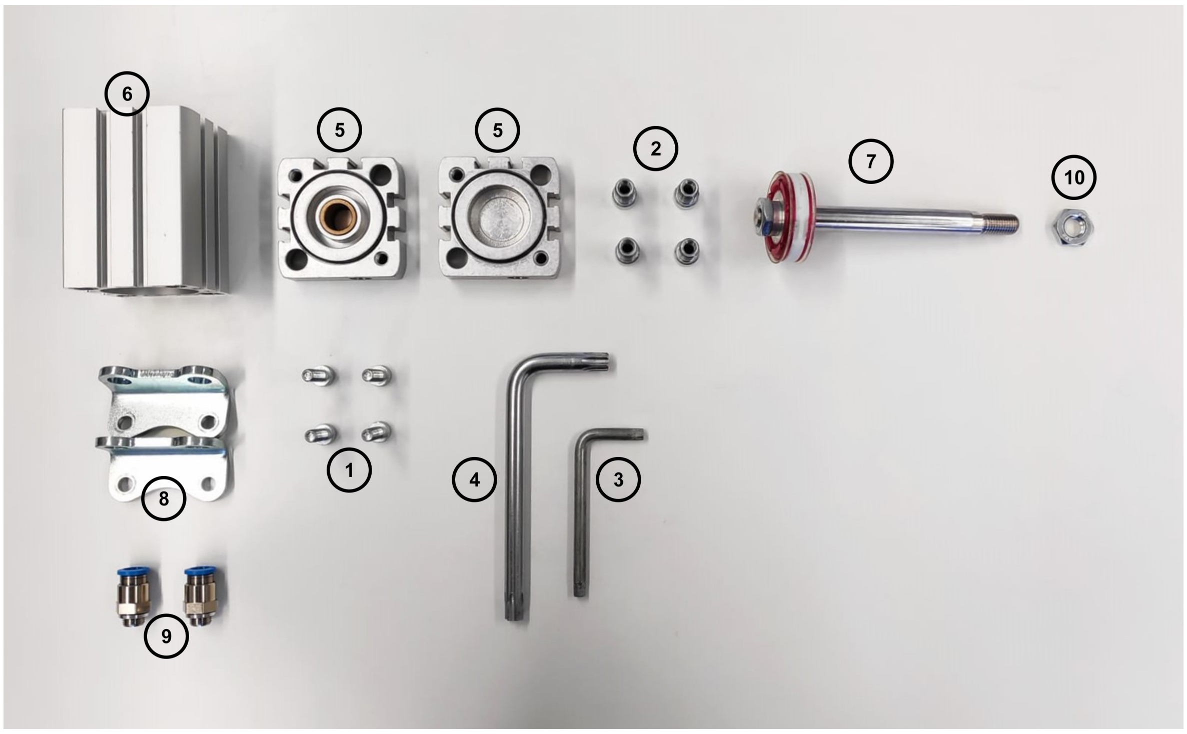


Figure 3.2: All the parts that compose the pneumatic cylinder and the two wrenches used, numbered as in the reported list

An arbitrary assembly sequence for the pneumatic cylinder is proposed below, with the list of every primitive actions to be carried out. This should be considered only as an example to understand the main steps of the assembly activity, since the sequence order is not fixed and there could be various techniques to be applied for assembling the cylinder, if the operator is left free to decide. Furthermore, the introduction of a collaborative robot in the assemble task necessarily modifies some steps of the sequence, introducing interactions with the robot.

1. Tighten the lower cap to the cylinder barrel:
    - Pick and place the cylinder barrel (the sides are equal);
    - Pick and place the lower cap;
    - Pick and place the screw (x2);
    - Tighten the screw (x2).
2. Insert piston rod into cylinder barrel:
    - Flip the assembly upside down;
    - Pick and place the piston rod on top of cylinder barrel;
    - Fully insert the piston rod into the cylinder barrel.

3. Tighten the upper cap to the cylinder barrel:
    - Pick and place the upper cap;
    - Pick and place the screw (x2);
    - Tighten the screw (x2).
4. Tighten the lower support to the lower cap:
    - Flip assembly on one side, so the barrel is horizontal, possibly with holes for air connectors on top;
    - Pick and place the lower support;
    - Pick and place the screw (x2);
    - Tighten the screw (x2).
5. Tighten the upper support to the upper cap:
    - Pick and place the upper support;
    - Pick and place the screw (x2);
    - Tighten the screw (x2).
6. Tighten the screwable air connections and the nut:
    - Pick and tighten the air connectors (x2);
    - Pick and place the nut;
    - Tighten the nut.
7. Remove the finished assembly.

3.2 Physical setup

The robot arm used in the related work is the KUKA LBR iiwa 7 R800 (Lightweight Robot Intelligent Industrial Work Assistant), a state-of-the-art collaborative robot designed for safe human-robot interaction. Key specifications of the KUKA LBR iiwa 7 R800 include:

- Number of axes (DOF): 7;
- Rated payload: $7\,kg$;
- Maximum reach: $800\,mm$;

- Pose repeatability (ISO 9283): $\pm 0.1\,mm$;
- Safety features: integrated torque sensors in each joint, allowing for precise force control and collision detection.

The end-effector used is a the SCHUNK Co-act EGP-C 64, a collaborating gripper for small components, featuring a set of collaborative fingers with overall opening range from 0 to $20\,mm$. The gripper is powered directly from the end of the robot arm through a dedicated cable. Key specifications of the SCHUNK Co-act EGP-C 64 include:

- Stroke per jaw: $10\,mm$;
- Minimum gripping force: $65\,N$;
- Maximum gripping force: $230\,N$;
- Power supply: $24\,V$ DC;
- Maximum total current: $2\,A$.

The camera used for the developed computer vision system is a Intel RealSense D435i camera, featuring RGB and depth sensors. The camera is mounted on a stand that is integral to the worktable and is positioned at a 45-degree angle to the horizontal. It captures the entire work worktable, which is at an optimal height for a standing operator. This type of configuration, where the camera is placed at a fixed point in the workspace, is commonly referred to as an eye-to-hand configuration [90]. The main features of the Intel RealSense D435i camera are:

- Depth technology: Stereo vision;
- Depth field of view (FOV): 87°x58°;
- RGB camera resolution: 1920x1080 pixels;
- Frame rate: up to 90 FPS for depth data, 30 FPS for RGB data;
- IMU: integrated inertial measurement unit for motion tracking.

It is anticipated that the camera has been configured for both RGB and depth channels with a resolution of 640x480 pixels at 30 FPS, so that frames can be aligned and 2D spatial information provided by the RGB sensor can be merged with depth information. More details on the actual parameters with which this camera has been used is given in Sec. 3.4, where the data acquisition setup is described.

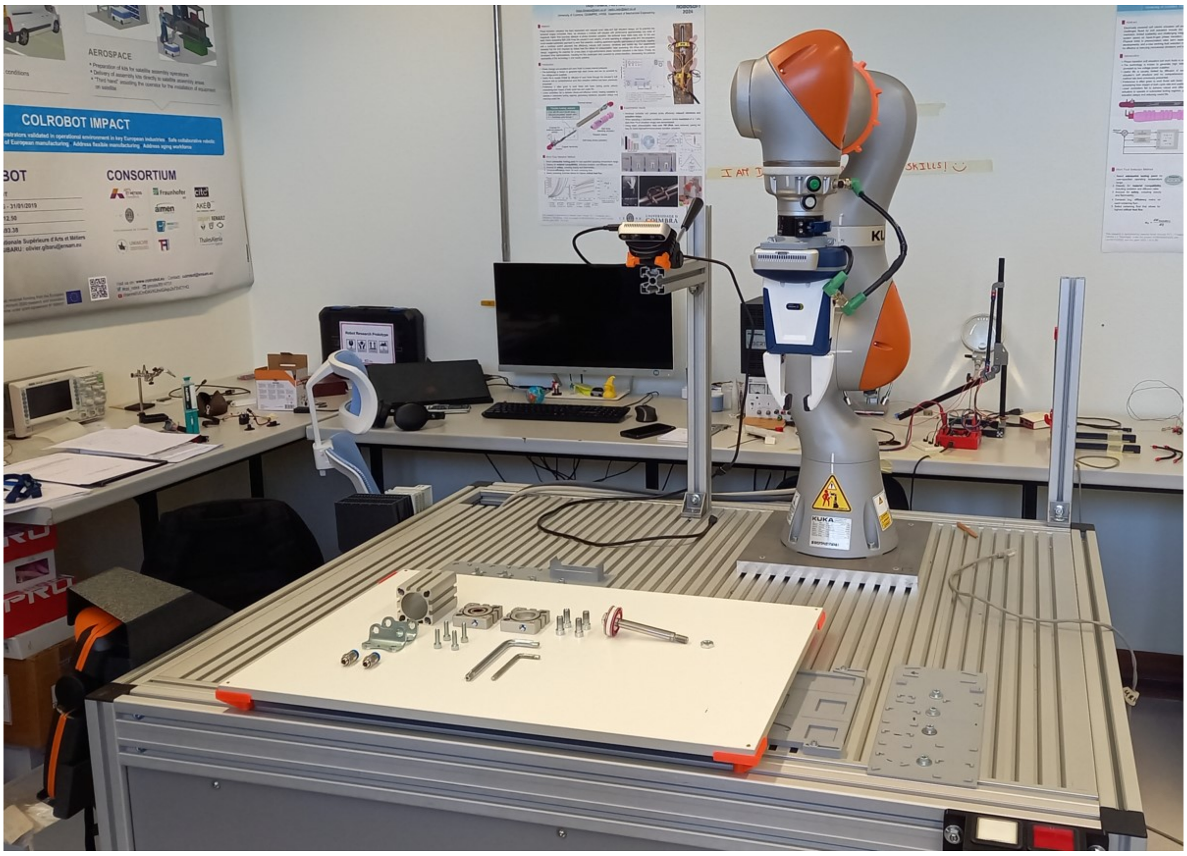


Figure 3.3: Physical setup

A PC connects the robot and the camera and runs the developed robot program. In particular, a laptop has been used, which showed a good performance, but could certainly be improved by using another, more powerful PC. The main characteristics of the laptop used are listed below:

- CPU: AMD Ryzen 5 5600H 3.30 GHz;
- GPU: NVIDIA GeForce GTX 1650;
- RAM: 16 GB.

All details related to the software are covered in the following sections.

Finally, the following figure shows the main measurements of the collaborative robotic station on which the related work has been developed.

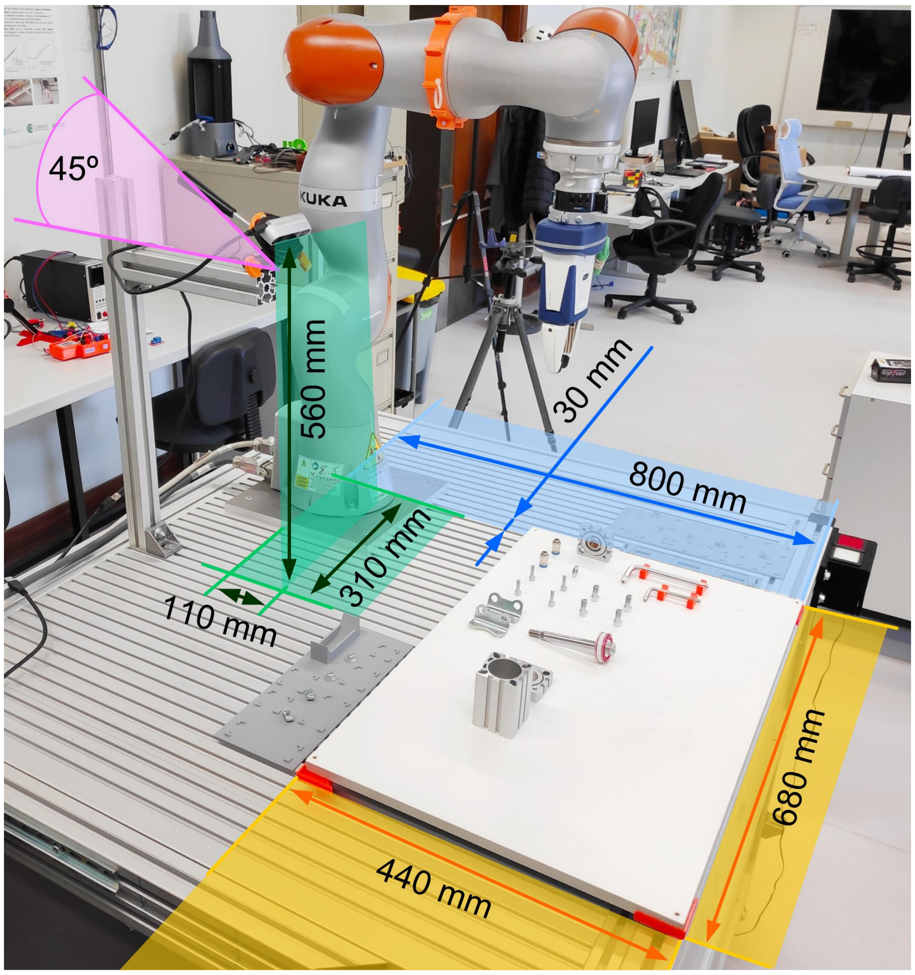


Figure 3.4: Collaborative robotic station with measures

3.3 Robot program

The robot program includes the three already mentioned main focuses of the project: action recognition, action prediction, and action anticipation. This involves: logic algorithms, neural networks, the computer vision control system, and the robot control system. These software run in real-time during the human-robot interaction. Beside this, several programs and scripts have been developed for data acquisition, data augmentation, data checking, data pre-processing, model training, and model testing. These software do not run in real-time in the assembling scenario but they were necessary to obtain the first ones.

Each of these software/scripts is described in detail below and has been uploaded on GitHub. Along with the main robot program, whose GitHub repository is [91], the current work has led to the development of a multi-object tracker and an image labeling tool, both of which are also available on GitHub, at the repositories mentioned in the respective sections.

The robot program and its functions are summarized in the diagram in Fig. 3.5. This diagram of the system helps to understand the different parts of the program and their relationships.

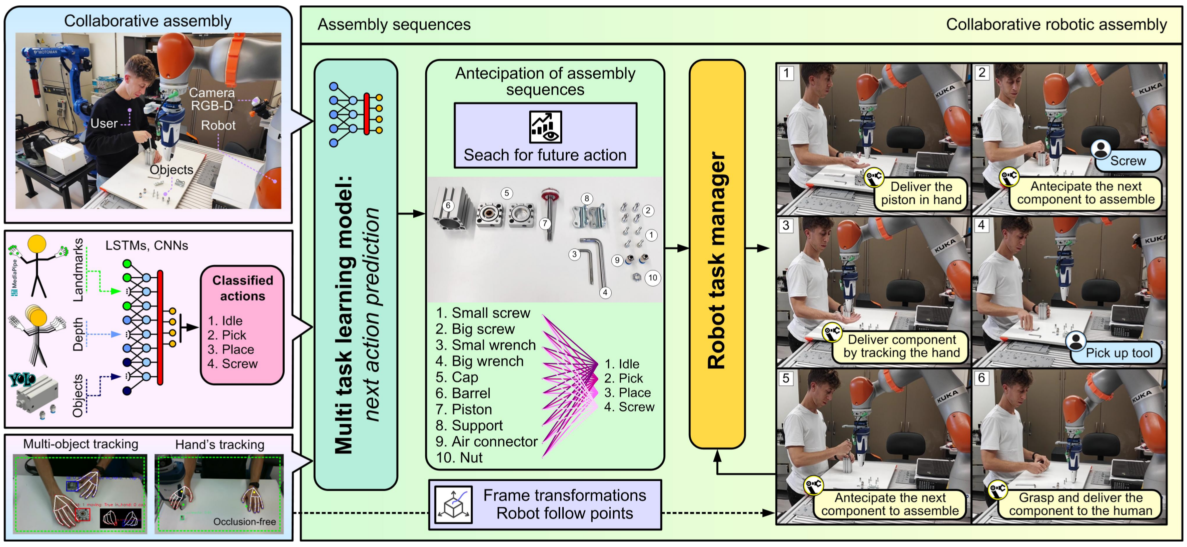


Figure 3.5: Overall diagram of the robot program

The main programming language used is Python, with the exception of a script in MATLAB for the calibration of the camera. The machine learning framework used for programming the neural networks is PyTorch. Also TensorFlow was taken into consideration and some tests have been carried out, but PyTorch resulted more flexible and suitable for the various needs of the project. The used version of PyTorch is 2.2.1, in combination with NVIDIA CUDA Toolkit, version 12.1, for GPU acceleration. Python environments have been developed in Anaconda package and environment manager, to simplify package installation and environment management. Any other technical aspect of the implementations and the specific technical choices will be discussed in the relative sections.

3.3.1 Action recognition

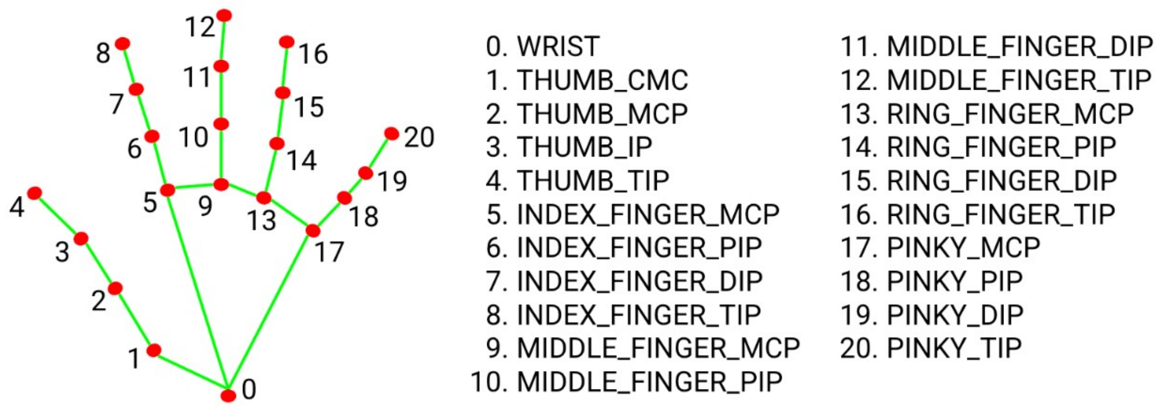
The human actions that the system has been trained to recognize are:

- Idle (not performing any action);
- Pick;
- Place (assemble, more in general);
- Screw with wrench.

These were selected as the most important actions in the assembly cycle of the considered pneumatic cylinder. The idle action is important to provide the model with a viable option in case no action is being performed, otherwise, in that case, high and fluctuating percentages would be allocated among the other actions, increasing the risk of incorrect detection.

For image analysis, the most widely used choice is Convolutional Neural Networks (CNNs), as previously mentioned in Sec. 2.4.2. The problem with CNNs is that they require lots of data to be trained. Acquiring such a big amount of videos, from which extract images, would have been too burdensome for the present work. For this reason, it has been chosen to use MediaPipe [64], in particular the "Hands" tracking solution [65]. In this way, the CNNs drawback of requiring lots of data to be trained is overcome, as CNNs are only used in MediaPipe, which is already trained on huge datasets and which will execute the image analysis, extracting hands landmarks. Landmarks represents human body relevant points in the space. MediaPipe Hands solution extract 21 landmarks for each hand [92], specifying:

- x, y: x and y coordinates normalized by the image width and height respectively (from 0.0 to 1.0, exceeding these values only if a landmark is estimated outside the image frame by seeing the rest of the visible hand);
- z: landmark depth relative to the wrist. The magnitude of z uses roughly the same scale as x.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 3.6: MediaPipe hand landmarks with indices and names [92]

Results also provide additional information such as hand recognition confidence and whether the detected hand is right or left. The confidence parameter has been used to decide whether the detection can be considered reliable, in particular with a minimum threshold of 0.5, while the hand parameter is necessary to distinguish the two hands.

The body pose detection has been excluded. This is because, in a manufacturing assembly scenario, particularly considering a static work environment that includes just a table, only the arms landmarks could be relevant to the recognition of the performed action. For MediaPipe solutions, each arm consists in three landmarks: shoulder, elbow, and wrist. Wrist landmark is already provided by the hand detection solution, leading to a total of only 4 additional landmarks provided by pose detection, 2 for each arm. To get the pose detection, the MediaPipe `Pose` solution or the `Holistic` solution can be used. The first one provides 33 pose landmarks, containing the 4 interested for the elbows and shoulders, therefore having to discard a large part of the data supplied. This solution, however, has the drawback of being completely detached from the Hands solution, requiring another continuously running neural network during the acquisition of information in real-time, leading to a decrease in FPS in the order of 5, considered inappropriate compared to the few additional landmarks provided. The second solution, Holistic, allows for the perception of simultaneous human pose, face landmarks, and hand tracking. Implementing this solution, the quantity of effectively used landmarks for the action recognition purpose of the related work would be minimal, since in addition to the 33 pose landmarks, there would be other 468 face landmarks that the model would continuously search for, but which would never be captured by the camera as it points directly on the working table, unless by mistake, and would in any case be discarded. In addition, the hands tracking results less accurate with this last solution, compared to the Hands solution, which is specialized in hands detection.

Inputs selection and data pre-processing

For the hand landmarks, the x, y and z coordinates provided by the MediaPipe Hands solution, listed and detailed above, have been considered. For each landmark, the x and y coordinates relative to the wrist of the respective hand have also been calculated in the data pre-processing, in order to complete the triad of relative coordinates together with the z already calculated by MediaPipe. This has been done to help the neural network to focus on the relative position and relative movements of fingers of the same hand, which are very important in action recognition, even more than the overall movement of the hand. In some research, especially those dealing with body motion, particular angles between different body parts are also calculated and fed into the network [68]. Human fingers have multiple flexion points and it is difficult to define meaningful angles between finger parts that can help to identify the action being performed. So, in this work, where only hands motion has been considered, it has been chosen to provide the network only with the relative coordinates between the landmarks of the same hand.

Moreover, for completing the triad of absolute coordinates, the depth information from the camera, which represents the absolute z, should be added to absolute image coordinates provided by MediaPipe. The problem with the depth sensor of the RealSense camera is that it has an accuracy not in line with that required for the application and, in general, is not reliable for the single point (pixel). Indeed, the depth image contains many sparse zero values which are associated with a points in the frame not being captured by the depth sensor. To overcome this limit, a system based on a kernel around every frame point has been designed and tested. Despite this algorithm works well, the lack of accuracy in depth measurements has made it preferable not to use this value, as they were seen to be more of a source of noise for the neural network than an enhancement. More details about this are reported in Sec. 3.4. Depth values have been acquired for testing purposes anyway, but they have been then excluded from the selected data to execute the action recognition. See Sec. 4.1 for more details.

In summary, the five numerical data that characterize each landmark for the action recognition neural network are: the two absolute image coordinates and the three coordinates relative to the wrist, all normalized. These pre-processing operations are located in the `action_recognition_model_train.ipynb` program, in particular in the `get_selected_frame_data` function.

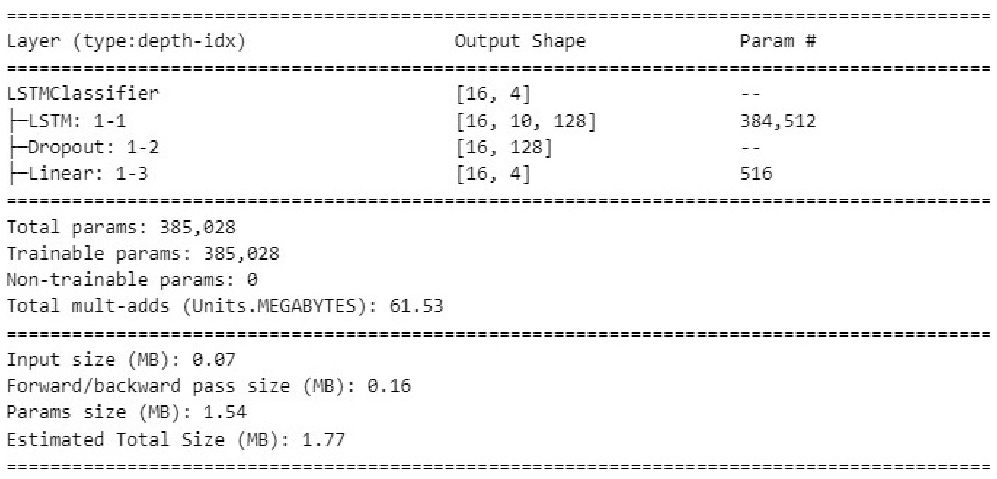
With this data provided by MediaPipe, the initial problem, which involved sequential images analysis, is transformed into more generic numeric sequential data analysis, making the train process more affordable in terms of required samples. Despite this big advantage, this process leads to the downside of data discarding: every information related to the original video, so to the frames, which is not correlated to the human movements captured, is essentially discarded as it is not being acquired. This means the complete loss of the context in which the actions is being executed, that can contain essential information for the action recognition. In Sec. 3.4, speaking about data acquisition, it is shown a script that represents the data actually acquired by the system in a blank frame. Note that this is only a representation of acquired data which can be easily visualized by a human and it is not fed into the neural network as an image - the neural network only processes landmarks data as values. It is a very interesting way to compare system recognition skills to human ones, based on the same available data. There is no doubt that performing the image analysis directly in the action recognition neural network, so without any loss of context, can lead to better recognition. Regardless of this, the amount of data needed for training to reach the same generalization achieved through MediaPipe's hand landmarks is incomparable with the data needed in the latter case.

A helper class, named `HandHelper`, has been created to easily register hands landmarks, divided into right and left hands, and to obtain some notable additional hand points that are important for the application. Specifically, these points are the midpoint between the tips of the thumb and index finger, which is used to retrieve the hand-held object, and the center of the palm, which is used for the robot to deliver objects to the human hand.

Models

Even introducing MediaPipe's hands tracking, the nature of the problem remains sequential and a model with memory mechanisms has to be introduced. From those described in Sec. 2.4.2, four different models have been structured and trained with the same available data, which are listed below. The reported number of neurons and layers are the best found for each model. For every of these models, the output size of the last layer is equal to the number of actions to classify, which is 4. This information will be omitted in the following list.

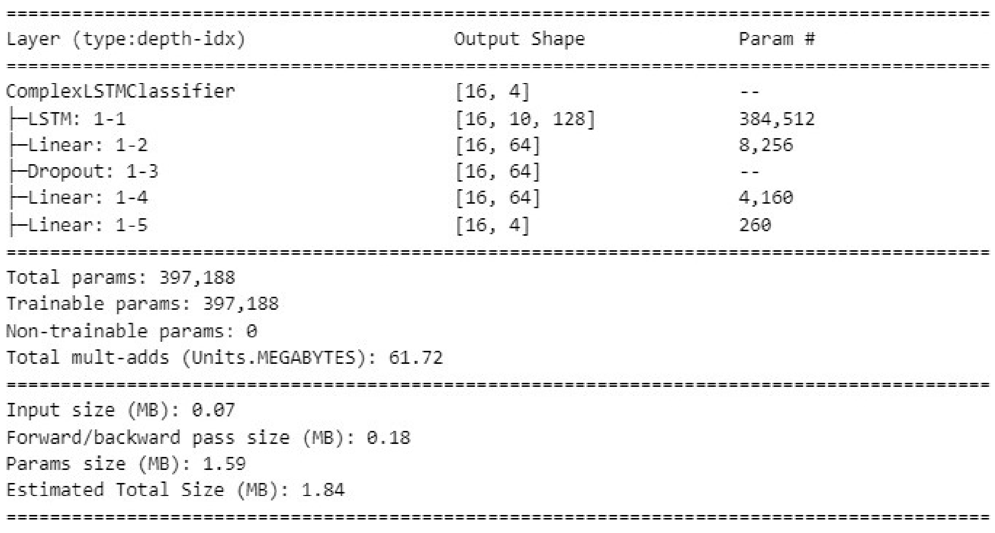
- Simple LSTM model: three layers of LSTM with 128 neurons each with an applied dropout ratio of 0.2, followed by a dropout layer with ratio 0.2 and one fully connected (dense) layer, with input size equal to the hidden state size of LSTM network, that is 128. The name "simple" indicates the presence of just one fully connected layer after the LSTM network, which is essential to translate the information coming out of the last layer of this into the final classification. In contrast, the other used LSTM model was referred to as "complex" to indicate the presence of multiple fully connected layers after the LSTM network;



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
LSTMClassifier                           [16, 4]                   --
├─LSTM: 1-1                              [16, 10, 128]             384,512
├─Dropout: 1-2                           [16, 128]                 --
├─Linear: 1-3                            [16, 4]                   516
==========================================================================================
Total params: 385,028
Trainable params: 385,028
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 61.53
==========================================================================================
Input size (MB): 0.07
Forward/backward pass size (MB): 0.16
Params size (MB): 1.54
Estimated Total Size (MB): 1.77
==========================================================================================
```

Figure 3.7: Simple LSTM classifier model summary

- Complex LSTM model: same initial structure of simple LSTM model: three layers of LSTM with 128 neurons each with an applied dropout radio of 0.2, followed by a dropout layer with ratio 0.2. After these, three fully connected layers, of sizes respectively: (128, 64), (64, 64), (64, 4). After the first and the second fully connected layer a rectified linear unit (ReLu) activation function has been applied to introduce non-linearity [93]. It cannot be applied to the last layer as it is the one that produce the classification output and it would interfere with the softmax activation function used in cross-entropy loss and to get the model's predicted probability distribution. As showed in the below model summary, number of parameters is not heavily modified by the introduction of two more dense layers, especially because the LSTM network requires many more parameters.



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ComplexLSTMClassifier                    [16, 4]                   --
├─LSTM: 1-1                              [16, 10, 128]             384,512
├─Linear: 1-2                            [16, 64]                  8,256
├─Dropout: 1-3                           [16, 64]                  --
├─Linear: 1-4                            [16, 64]                  4,160
├─Linear: 1-5                            [16, 4]                   260
==========================================================================================
Total params: 397,188
Trainable params: 397,188
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 61.72
==========================================================================================
Input size (MB): 0.07
Forward/backward pass size (MB): 0.18
Params size (MB): 1.59
Estimated Total Size (MB): 1.84
==========================================================================================
```

Figure 3.8: Complex LSTM classifier model summary

- Bi-directional LSTM (Bi-LSTM) model: identical structure of simple LSTM model but with the LSTM network composed of three layers of Bi-directional LSTM, with same number of neurons and same dropout ratio applied. The model summary below shows the huge increase in model parameters brought by the usage of Bi-LSTMs, as hidden states and cell states requires twice as many parameters, in addition to the general greater number of parameters for each layer.

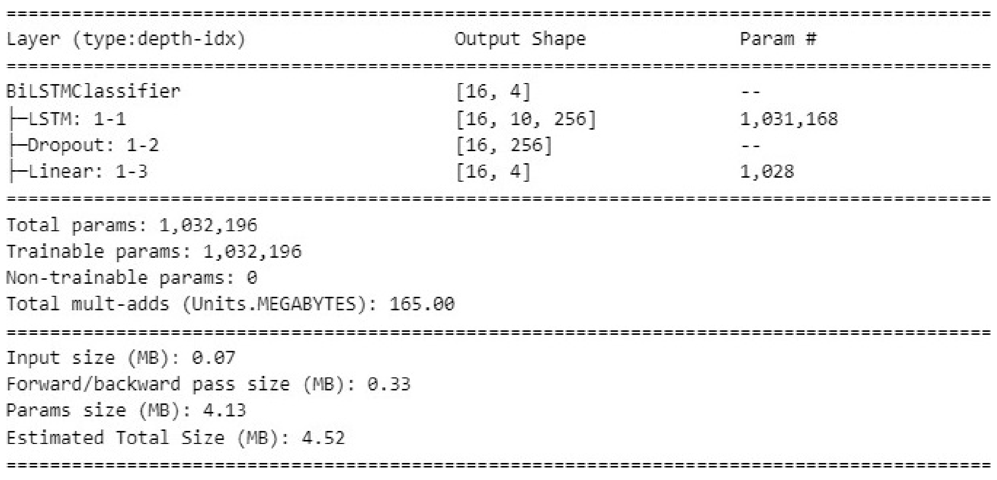
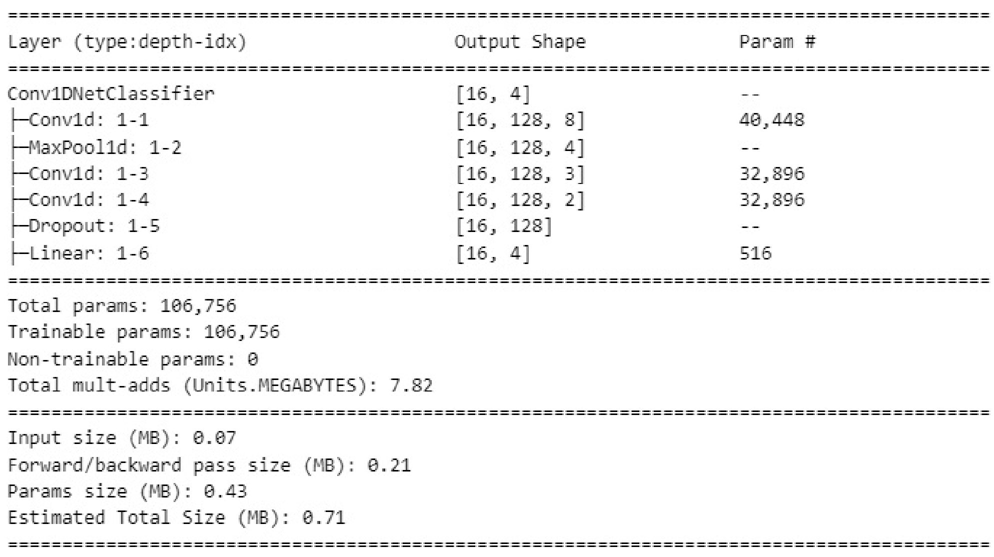

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
BiLSTMClassifier                         [16, 4]                   --
├─LSTM: 1-1                              [16, 10, 256]             1,031,168
├─Dropout: 1-2                           [16, 256]                 --
├─Linear: 1-3                            [16, 4]                   1,028
==========================================================================================
Total params: 1,032,196
Trainable params: 1,032,196
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 165.00
==========================================================================================
Input size (MB): 0.07
Forward/backward pass size (MB): 0.33
Params size (MB): 4.13
Estimated Total Size (MB): 4.52
==========================================================================================
```

Figure 3.9: Bi-LSTM classifier model summary

- Convolutional 1D (Conv1D) model: three consecutive 1D convolutional layers, with kernel sizes of 3, 2, and 2, respectively. The channel size for these has been set to 128. A MaxPool1D layer, with kernel size of 2, is applied after the first convolutional layer to reduce the feature maps dimensions, and so the model's number of parameters, keeping the most important information. The output of every convolutional layer is processed by a ReLu activation function. Continuing, there is a dropout layer with ratio 0.2, followed by the fusion layer to apply a late fusion approach, in order to leverage the temporal information of the data. This fusion layer performs the average of the features extracted by the convolutional network along the time axis, i.e., the last dimension of the tensor. Note that the spatial and temporal axis, i.e., the second and the third dimensions of the tensor, have been permuted for this network, since this is the most appropriate shape to do convolutional operations. This permutation is done at the start of the forward pass. Finally, a fully connected layer concludes the network, also with input size equal to the number of frame features.



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Conv1DNetClassifier                      [16, 4]                   --
├─Conv1d: 1-1                            [16, 128, 8]              40,448
├─MaxPool1d: 1-2                         [16, 128, 4]              --
├─Conv1d: 1-3                            [16, 128, 3]              32,896
├─Conv1d: 1-4                            [16, 128, 2]              32,896
├─Dropout: 1-5                           [16, 128]                 --
├─Linear: 1-6                            [16, 4]                   516
==========================================================================================
Total params: 106,756
Trainable params: 106,756
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 7.82
==========================================================================================
Input size (MB): 0.07
Forward/backward pass size (MB): 0.21
Params size (MB): 0.43
Estimated Total Size (MB): 0.71
==========================================================================================
```

Figure 3.10: Conv1D classifier model summary

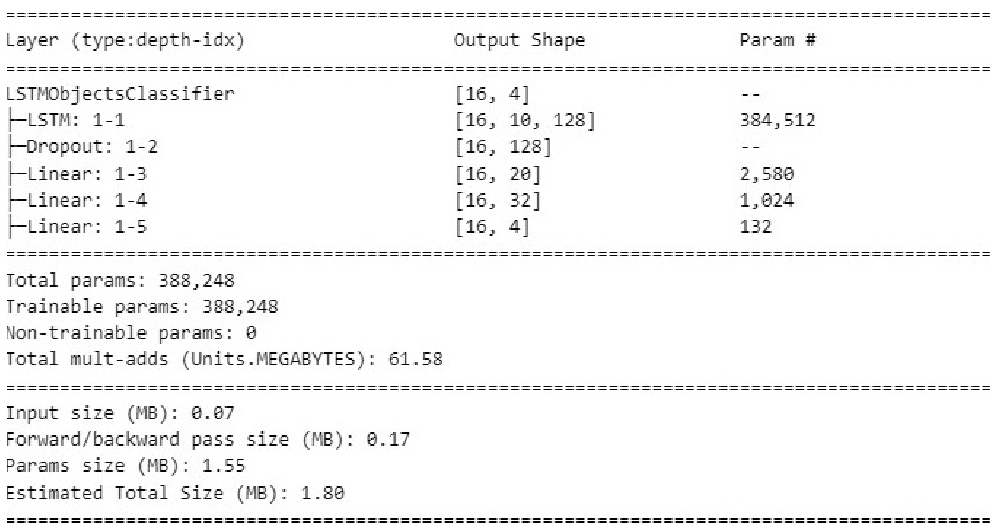
Dropout mechanism has been introduced in every model, using dropout layers and/or specifying dropout ratio in PyTorch LSTM network, which introduces dropout layers with specified ratio between LSTM layers. This introduces noise in data during the training process and helps to use the full potential of network's neurons, improving the generalization capabilities of the model [94].

The models described above have been designed to be used only with data from MediaPipe hands landmarks. In fact, there are no special weighting mechanisms to differentiate data of different nature. Another model has been projected and trained for the action recognition task. This one includes objects data, in particular the objects held in the hands, provided with a combination of object detection performed by YOLO, described in Sec. 3.3.4, and a dedicated multi-object tracker that considers MediaPipe hands landmarks to evaluate which object is being held in hand, described in Sec. 3.3.5. The goal is to better classify actions according to the objects being used, which can be seen as the inverse of learning the affordances of objects from human demonstration [95]. It has been estimated how including object data should allow for a better classification especially for actions that are similar in movements but include different objects in being performed. A case in point is the differentiation between the place and screw_wrench (screw with wrench) action, which can be very similar in the grip of the object and also in the movement, especially when considering only one hand and not having the other as a reference. Including the object hold in the hand, for example a wrench, should remove any doubt about the performed action. Thinking about it, this is what would be implicitly performed by the human brain in an action recognition task. It should be mentioned that, since the objects detection is not 100% accurate, the constructed dataset includes many actions in which the objects in hand were not detected, precisely to provide for this deficiency during the real-time implementation. This means that, some actions in the dataset, and also in real-time, will base their recognition only on hand movement data, even though the model is structured to handle object data as well, and the aforementioned increased confidence in the classification would be lost.

Including objects in the classification of actions, while it might bring benefits in correct recognition, still represents a reduction in flexibility compared to a model that works only with motion data, since the model would be strictly related to the assembled object under consideration. For this reason, the model that will now be described has been extensively tested and its benefits have been evaluated, but the main goal of the work, regarding action recognition, remains to build a model based only on human motion, specifically on the hands landmarks provided by MediaPipe, to be as generic and flexible as possible. In addition, few attempts with such a model including object data for action recognition have been found in literature, so it was deemed appropriate to investigate this topic further.

The data considered for objects are only the label ids. This is because information about detection confidence or bounding box position is useful when the object is actually visible in the frame. The key additional information brought by the object tracker is precisely to indicate what object is supposedly being held, even when that object is not actually visible in the frame. At this point, however, the other information other than the label id of the object in the hand, such as confidence and position, loses its meaning, since the object is not actually detected by YOLO, and the position of the object can only be estimated as coincident with that of the hand, which is data already provided to the model via hand tracking.

The input data contains both hands landmarks data and one-hot encoded label ids of the objects held in the hands. This data is actually provided to the neural network as a concatenated 1D vector, but it is only as a matter of convenience in manipulation and they are not actually processed in this form. Indeed, the amount of hands landmarks data is much higher than the one-hot encoded label ids, and if they would be fed into the network as a unique vector without some weighting mechanism, suggesting the model to increase the attention to the latter, they would be irrelevant. Both the weighting of input data and loss have been tested, but they did not give optimal results. It has been therefore decided to change the model structure. The action recognition model that includes objects data has the same base structure of the Simple LSTM classifier described above. The 1D input vector is divided into landmarks and objects data at the start of the forward pass. The landmark data is fed into the LSTM network, with the same dropout ratio of 0.2. The fully connected layer that closes the LSTM network has an input size of 128, the same of LSTM layers, but, this time, the output size is set to 20. This is because here the objects data will be introduced, so, in this way, the data cardinality is similar for landmarks and objects. The objects data still contains temporal information, that has already been elaborated for landmarks data passing through the LSTM network. To merge temporal information about label ids, an average along the temporal axis, i.e., the second dimension of the tensor, has been performed. Averaging one-hot encoded vectors of object label ids over a period of time allows obtaining a normalized value of the presence of a given object in the frames of the sequence, which in this particular case means held in the hand. For example, is a specific object is detected as held in the hand for 4 frames out of the total 10 frames of the sequence, the corresponding value inside the average one-hot encoded vector would be 0.4. At this point, landmarks and objects data with merged temporal information appear as two 1D vectors of similar cardinality, which can be concatenated and fed into the remaining part of the network to obtain a meaningful prediction. The last layers of the network are two fully connected layers, with size of 32, named “extra” to highlight their addition to the “Simple LSTM” model. The ReLu activation function has been applied in this last part as well to introduce non-linearity.



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
LSTMObjectsClassifier                    [16, 4]                   --
├─LSTM: 1-1                              [16, 10, 128]             384,512
├─Dropout: 1-2                           [16, 128]                 --
├─Linear: 1-3                            [16, 20]                  2,580
├─Linear: 1-4                            [16, 32]                  1,024
├─Linear: 1-5                            [16, 4]                   132
==========================================================================================
Total params: 388,248
Trainable params: 388,248
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 61.58
==========================================================================================
Input size (MB): 0.07
Forward/backward pass size (MB): 0.17
Params size (MB): 1.55
Estimated Total Size (MB): 1.80
==========================================================================================
```

Figure 3.11: LSTM with objects classifier model summary

The `get_selected_frame_data` function of the `action_recognition_model_train.ipynb` program, used to extract the selected frame landmarks data, is also structured to extract frame objects data, which can be set to `None` for use with models that do not include objects data.

Dataset

Datasets of human actions related to the manufacturing context are currently very few. In particular, it is easier to find datasets linked to footage of large areas of manufacturing factories, for example acquired by surveillance cameras, which can be used to train models capable of recognizing potential dangerous situations, but there are no specific datasets for industrial assembly. The dataset found most in line with the needs is Assembly101 [96], which contains videos of people assembling and disassembling 101 "take-apart" toy vehicles. The data of interest are the action verbs on fine-grained actions, which have the same meaning of the actions to be recognized in the related work. On this dataset and on this specific parameter, the action recognition Top-1 accuracy is always less than 70%, obtained using a state-of-the-art video recognition model and two top-performing graph convolutional networks on poses. This can be considered an excellent result if we take into account the extension and variety of the dataset and the actions it includes. Despite this, since in the related work the prediction and anticipation steps are built on top of the action recognition process, even a 70% of accuracy on this one could result not suitable. However, several tests were carried out on this dataset, running the data acquisition script to extract hands landmarks on 50 complete videos, searching for just the four selected actions listed above, that are included in the labeled actions. This showed some limitations of MediaPipe in continuously tracking, i.e., in all frames, the hand landmarks for the main fixed views, which are quite distant from the human operator's hands. In addition, the large difference in extension of the actions makes it difficult to format them equally to be processed by the neural network. The loss and accuracy results obtained with such sampled data from this dataset were poor, leading to the conclusion to adopt a dataset more targeted to the specific assembly problem under analysis. As mentioned, since no other dataset with the searched characteristics was found, a custom one has been constructed.

When creating a dataset for action tasks, such as action recognition, it is necessary to distinguish between clipped actions and multiple actions in long videos, like Assembly101 dataset. In the former, the actions and video clips are on the order of a few seconds, while in the latter, it is several minutes [97]. Extracting actions directly from a video containing a sequence of actions can help dealing with transitions between actions, which are a known problem in implementing action recognition models in real-time [98]. Despite this, as experienced in Assembly101 dataset, actions extracted from a sequence can be vague, overlapping, and differ significantly in length, which makes it difficult to format them equally. In fact, it has been observed that temporal boundaries of actions are not precisely defined in practice, whether they are obtained automatically using weak-supervision or by hand [62]. For these reasons, it has been chosen to build a dataset of clipped actions.

Sec. 3.4 has been dedicated to the description of the data acquisition methods used to build the action recognition dataset. This dataset consists of 250 sequences of frame data for each of the four actions to be classified, for a total dataset length of 1000 sequence samples. Frames data contains both hand landmarks and object information. Each sequence contains 10 frame data, corresponding to an acquisition of 1 second at 10 FPS. All sampled actions have been performed with the right hand, so that single-handed models could be trained. Left hand landmarks and held objects have been sampled as well, to get information about the auxiliary hand. Training action recognition models that account for actions performed with both hands would require mirroring the dataset. This has not been done because the double-handed models studied have been limited to recognizing only actions performed with the right hand as the main hand and the left hand as the auxiliary hand, to determine whether including auxiliary hand data could improve performances respect to single-handed models.

It is worth noting that the number of samples used to build the dataset is generally low to train an action recognition model that is flexible to multiple contexts and is largely robust to multiple assembly environments. On the other hand, the frame data sequences used to build the dataset for action recognition have all been acquired with the camera used to develop the related computer vision system, in the fixed working position. As a result, far less data is required to train the model, although this inevitably reduces the flexibility of the developed model, which is unlikely to achieve the same performance when moved to another workstation. This trade-off has been widely accepted, deeming the related work an excellent example of the performance achievable in a specific human-robot collaborative assembly scenario, which can be extended to other more general scenarios and, if desired, with more actions, simply by expanding the dataset of actions.

Training

A custom train-validation-test splitting function, named `train_val_test_split` is applied to the data in the dataset to get train, validation, and test sets. This function was inspired by the widely used `train_test_split` function of the well-known machine learning Python library `scikit-learn`, with the difference that it also allows the validation set to be obtained directly as a percentage of the initial dataset, without the need to call the function twice and do the percentage calculations to obtain the desired number of samples from the already modified set.

The listed models have been trained and tested on many different combinations of data. Results will be discussed in Sec. 4.1. For all these models, the same loss function, optimizer, and learning rate stepping scheduler were used, as they are best suited for the action recognition problem addressed. The loss function used is the cross-entropy loss, which is the basic choice for classification problems with more than two classes. As an optimizer, the Adam optimizer has been chosen because, due to its efficiency and robustness, it can be considered an excellent default choice for this type of problem. Stochastic gradient descent methods, such as Adam, and optimizers in general, benefit from learning rate decreasing over epochs [99]. This is why a multi-step learning rate scheduler has been implemented as well. It decays the learning rate of each parameter group by a certain gamma once the number of epoch reaches one of the milestones. Gamma decay parameter has been set to 0.1, which means that each milestone the learning rate of the optimizer is reduced by an order of 10, and two milestone epochs have been set to 40 and 80. The initial learning rate for the Adam optimizer is set to 0.001. Different batch sizes have been tested, concluding that the best for the analyzed neural networks is 16.

The training loop is set to a maximum of 100 epochs, which is usually not reached because the implemented early stopping mechanism triggers before. This early stopping logic is based on the validation loss, which is the loss function calculated every epoch over the entire validation set. The `min_delta_improvement` parameter sets a minimum improvement threshold for the validation loss, and if the improvement does not reach this value within the number of epochs specified by the `patience` parameter, the training loop terminates, optionally restoring the best model weights. In the related work, the `min_delta_improvement` is set to 0.001, the patience to 15 epochs and the model is restoring the best weights.

Testing and saving

After the training phase, a set of functions have also been developed in the `action_recognition_model_train.ipynb` program in order to test and eventually save the trained models. The performance metrics used to evaluate the model are accuracy, precision, recall, F1-score, and ROC-AUC score. Their specific usefulness is explained in detail in the corresponding Sec. 4.1 of the results chapter. They are evaluated on both the validation set and the test set, to provide an unbiased assessment of model performance. Confusion matrices and ROC curves are also generated for both sets, allowing a deeper understanding of the models' behavior. The last part of the program, is dedicated to a specific test function which aims to recognize the performed action and to compare the predicted probabilities with the expectations of a human supervisor. The actual sequence data being processed by the neural network is displayed as video over a black frame, at the same FPS as the sampling. This allows the human to check for the quality of the processed sequence and to judge its consistency with the action under consideration. In this way, it is possible to understand whether any prediction errors are due to the model or to noisy input data.

The trained (and tested) models can be saved either using the normal `state_dict`, which is a Python dictionary object that maps each layer to its parameter tensor, including also the optimizer state in the model checkpoint, or using the `TorchScript` format. This is an intermediate representation of a PyTorch model that can be run in Python as well as in a high performance environment like C++ [100]. Although the former format is the most flexible for restoring a model and continuing to work on it, it has the disadvantage of requiring the model's class reference to infer its structure. For this reason, the latter format has been chosen because of the simplicity and conciseness with which the various models can be saved and used in the real-time execution program that implements them. However, both saving methods are present in the training program.

Real-time implementation

For the real-time implementation of the action recognition model, a function called `recognize_action` has been created in the robot program. This function performs inference on the model starting from a `data_buffer`, which contains the same number of frames as the clipped actions used to train the model, in this case 10. The buffer is dynamically populated by adding data from the most recent frame and discarding the oldest data. The outputs of the action recognition model obtained in real-time using this method are highly variable, potentially changing drastically from one inference to the next due to the variation of a single added frame. Therefore, a probability smoother is used. Specifically, exponential smoothing is employed, which is commonly used to smooth the output variations of real-time classification models. This smoother has a parameter α that represents the relative weight of the new probabilities provided on the update of the previous step's probabilities, kept in the memory of the smoother class. The higher the α, the greater the weight the new probabilities have on updating the old ones. Let $\mathbf{p}_n$ be the vector of action probabilities classified at the current frame n, whose sum equals one and where p_i represents the probability attributed to the current execution of the action of index i by the observed human operator. Let $\tilde{\mathbf{p}}_{n-1}$ be the vector of smoothed probabilities from the previous frame $(n-1)$ saved in the smoother class memory. The smoothed probabilities of the current frame are given by the following formula:

$$\tilde{\mathbf{p}}_n = \alpha\,\mathbf{p}_n + (1-\alpha)\,\tilde{\mathbf{p}}_{n-1}$$

The value of α has been set to 0.4, meaning a 40% weight of the current probabilities on updating the previous ones. This value is moderately conservative and allows for very good behavior of the action classification probabilities, reducing fluctuations while allowing high confidence thresholds for correct classification. The threshold value set for classification confidence is 0.9. Additionally, colored bars for each action have been added in the top left corner of the camera view screen, filling up based on the classification probability percentage. This provides clear and intuitive information during real-time tests. The text with the corresponding action name lights up in yellow when the 90% threshold is exceeded.

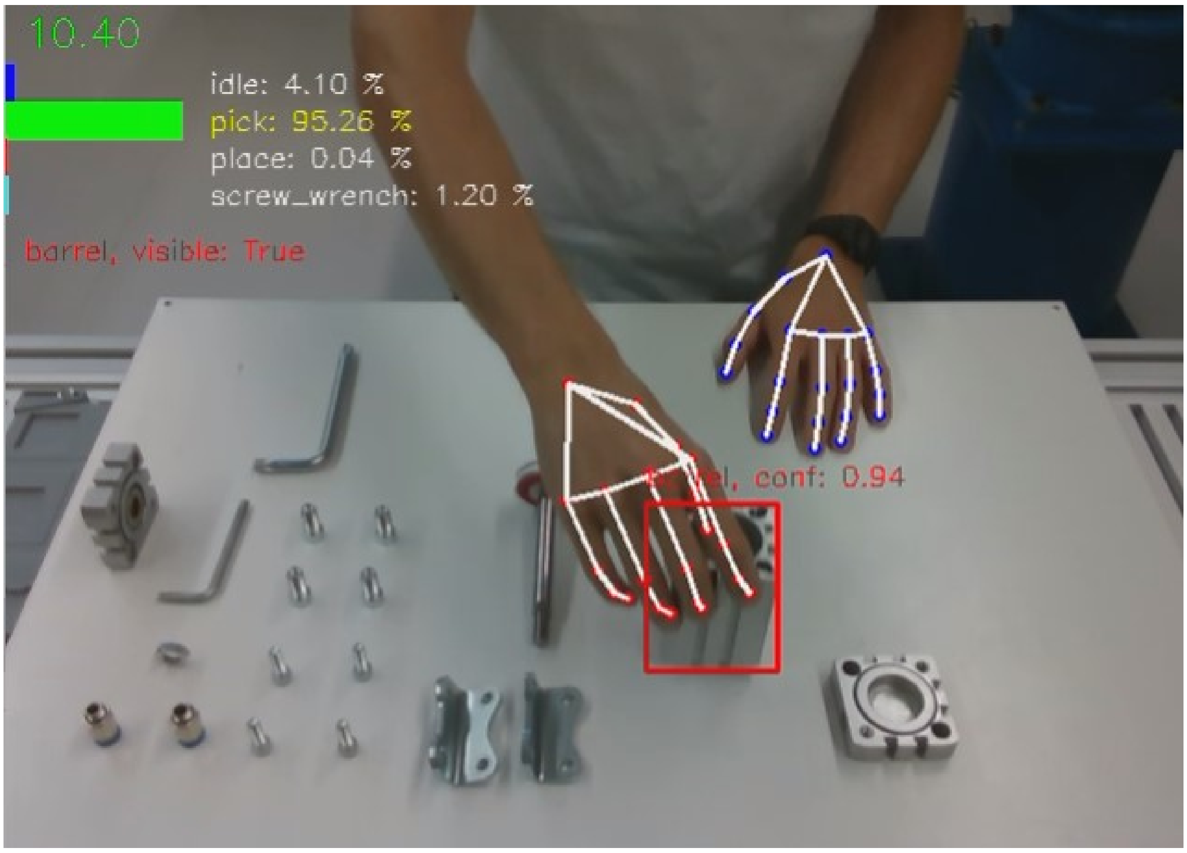


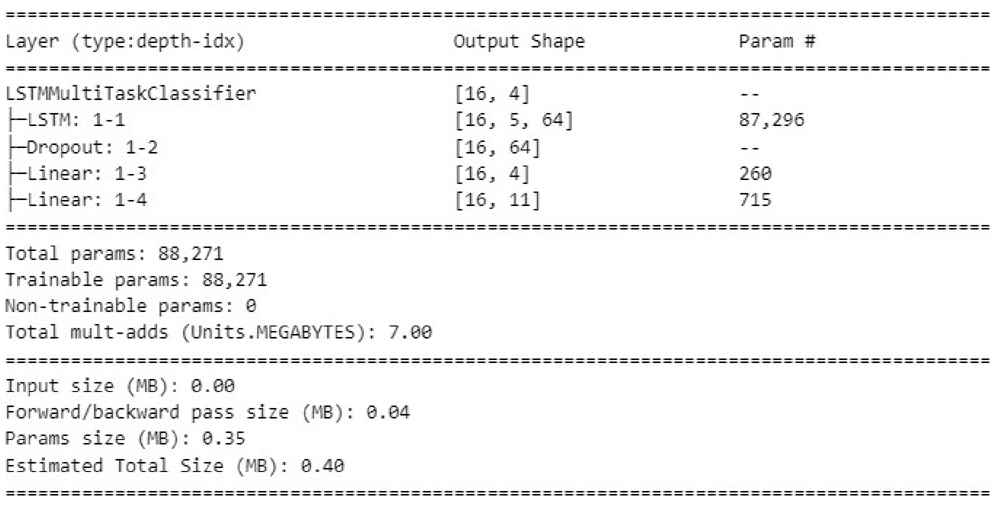
Figure 3.12: Probability bars for action recognition in real-time implementation

3.3.2 Action prediction

Another fundamental focus of this work is the action prediction. This falls into the class of problems known as time-series forecasting or time-series prediction. In the related work, prediction is interested in two different types of data: actions and objects. These data are correlated, although the forecast results should be separated. In this scenario, multi-task learning (MTL) is the key to creating models that can learn multiple tasks with correlations between them. These sophisticated models are able to leverage interdependencies within the input data, including various forecasting objectives within a single framework [101].

Model

The MTL classifier model built to handle this time-series prediction problem which involve two tasks is based on a LSTM network, that represents the common part of the overall network. The input vectors, containing data for both actions and objects, are first fed into this network with three LSTM layers of 64 hidden size. A dropout ratio of 0.2 is applied as well. Then, a dropout layer with ratio 0.2 is applied to the last hidden state, containing the most relevant extracted temporal information. From this, the neural network branches into two sub-networks, each in charge of performing the prediction of its respective type (actions and objects). The sub-networks in this case are very simple, consisting of a single fully connected layer, both with input size equal to 64, in accordance with the hidden size of the LSTM network, and output sizes equal to the cardinality of actions and objects respectively, in this case 4 and 11 (10 objects + 1 for the empty hand). The model produces two distinct outputs, which need to be compared with the corresponding true-ground labels and for which two separate loss functions are required.



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
LSTMMultiTaskClassifier                  [16, 4]                   --
├─LSTM: 1-1                              [16, 5, 64]               87,296
├─Dropout: 1-2                           [16, 64]                  --
├─Linear: 1-3                            [16, 4]                   260
├─Linear: 1-4                            [16, 11]                  715
==========================================================================================
Total params: 88,271
Trainable params: 88,271
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 7.00
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.04
Params size (MB): 0.35
Estimated Total Size (MB): 0.40
==========================================================================================
```

Figure 3.13: MTL classifier model summary

Dataset

To obtain predictions, it is necessary to create a dataset that contains the action-object sequences used during the assembly cycle, and to determine the prediction window size. The prediction window is the set of consecutive data within the reference sequence that the model uses to make predictions. The size of this window is a sensitive parameter since the flexibility and accuracy of the network will depend heavily on it. With an extended window there is greater confidence in the prediction because it is based on more data, but at the same time, having an excessively large window makes it difficult for the model to be flexible to changes in the sequence, as it bases predictions on a considerable amount of consecutive data. A size of 5 has been chosen for the prediction window. Less extended windows have also been tried out but, since some small sets of actions are repeated many times within the same sequence of assembling, e.g., pick and place a screw, it has been seen how reducing it below 5 resulted in model insecurity about where in the sequence the current prediction window actually was, resulting in erroneous predictions.

Only the object held by the right hand has been taken into account. This decision was made to make the system simpler but at the same time to allow focus on a single hand, which together with action recognition models based on single hand movements, leaves room for future implementations of two-independent-handed models, both for action recognition and prediction. This complex system has not been investigated in the present study, as it has been considered outside the already sufficiently extended scope.

The system handles the case of empty hand (or undetected object). An additional label id is added, incrementally, to the end of the list of existing ones precisely to indicate an empty hand or an undetected object. For the pneumatic cylinder under consideration, the length of a complete assembly sequence is, generally, between 40 and 60 action-object pairs, depending on the assembly order followed and, to a greater extent, how many idle actions are actually present between the effective actions. In a perfect scenario in which the action recognition model achieves near-perfect accuracy in a real-time application, it would be reasonable to discard idle actions from the dataset on which the network is trained and from the sequence of actions detected in real-time. The utility of an idle class for action recognition was explained in the dedicated Sec. 3.3.1, but in terms of action prediction it actually represents only random noise in the analyzed action sequences. By neglecting them, the model could focus more on the actual actions being performed. Unfortunately, although the models obtained for action recognition give very satisfying results, they do not achieve such high accuracy during real-time implementation as to allow this. Therefore, it has been preferred to also include idle actions with their respective held objects in the sequences for action prediction, so that even if an action is not correctly recognized, and consequently, with good probability, the idle action is recognized instead, there will be at least the information of the object being held. Thus, by training the action prediction model with the introduction of "idle actions noise", an attempt is made to reduce the impact of failure of the action recognition model on it.

Two different approaches have been used to create the datasets for training the action prediction model, both of which have been evaluated. Although the considered pneumatic cylinder can be assembled following very different sequences, since many components do not require a mandatory assembly order, it has been preferred in both datasets to adhere to a precise assembly sequence, which is the same described in Sec. 3.1, to verify the collaborative system's capabilities in a condition where the sequence must be strictly followed. In more permissive assemblies in terms of possible assembly sequences, a system that performs well under stricter conditions can only perform even better with more freedom available.

- A dataset generated from variations of a hand-built reference sequence, which will be referred to as "hand-built sequence variations dataset". This reference sequence is registered in the `assembling_sequence.csv` file. From this, 20 variations have been generated using the `generate_variation_df` function, contained in the `action_prediction_model_train.ipynb` notebook. This function introduces randomness into the sequence with the following operations:

  - Action-object pair deletion;
  - Idle action insertion;
  - Real action replacement with idle action;
  - Real object in the right hand deleted (replaced with "none" object);
  - Real object in the right hand replaced with a random object.

  Each of these variations is applied based on an independent probability, which has been set to 5% for all of them. These variations are designed to reproduce real conditions that may occur during real-time implementation. For example, the deletion of an action-object pair corresponds to an action that is not detected by the action recognition model, and consequently the object is not recorded either. Replacing an action with an idle action corresponds to the action recognition model not reaching the confidence level required to classify an action, so the idle action is recognized instead, as explained earlier and defined as "idle actions noise". It is pointed out that the base sequence does not contain any idle actions except for the initial one, and therefore all other idle actions in the dataset are randomly generated. This should be kept in mind when interpreting the results of the action prediction model. Object deletion or replacement corresponds to an object detector or object tracker failure. With these sequence variations, the model can be trained to successfully generalize to many situations that may arise during real-time implementation.

- A dataset generated from real assembly sequences observed by the computer vision system, based on the action information provided by the action recognition model and the information of the object held in the right hand provided by the combination of the YOLO object detector and the developed object tracker. It will be referred to as “real sequences dataset”. This scenario involves the human performing the entire assembly sequence independently, without the collaboration of the robot, while being observed by the computer vision system, for a sufficient number of sequences to allow the creation of a proper dataset. This would make it possible to create a collaborative system with no prior knowledge of the assembly sequence, which could be an interesting result for application flexibility, especially in the manufacturing industry. It is worth noting that it is a bit of a stretch to talk about no prior knowledge of the assembly sequence, since it is true that in this case the robot program does not have any information about the sequence itself, but it is still necessary to collect images of the specific objects of the component to be assembled and the tools required, and to label them manually in order to train YOLO. This is certainly a much more burdensome process than manually entering one or more assembly sequences. The expected result was that the model trained on this dataset would perform worse than the model trained on the dataset described above, since the real-time data for its construction is subject to the errors of the action recognition model, and for objects, those of YOLO and the object tracker are also added. In any case, it has been considered interesting to investigate this alternative and test its feasibility. A total of 20 complete real assembly sequences, including action and object data, have been collected to test a model trained in this modality.

Sequence variations’ data is saved into `csv` format files, as a pair of integer values containing the action id and the label id of the object held in the right hand. They are linked together to form one contiguous time-series. The prediction model needs a reasonably large amount of data in order to learn the sequences. Therefore, if few assembly sequences are available, they must be repeated and linked together. This linking process takes place at the beginning of the training script for the action prediction model, which is named `action_prediction_model_train.ipynb`. The Python data analysis library `pandas` is used to convert `csv` files into the efficient and easy-to-handle `DataFrame` data structures, which are then concatenated together.

Training

MTL models must necessarily be trained on data from all tasks at the same time and not sequentially, otherwise the phenomenon known as "catastrophic interference" occurs. This well-known problem refers to the tendency of neural networks to drastically forget information learned about a specific task if data of other nature, i.e., related to another task, is provided [102]. This is why custom functions and a custom PyTorch `Dataset` object were specifically created. First, the `df_to_X_y_data` function is used to collect data and labels from the concatenated `DataFrame` to train the model. It applies one-hot encoding to both action ids and object label ids, concatenating them for input data (X) and leaving them separate for labels (y). Next, a custom train-validation-test splitting function, named `multi_task_train_val_test_split` is applied to get train, validation, and test sets. This function was also inspired by the well-known `train_test_split` function. It also allows to get the validation set directly, and it differs from it in the way the data is processed, as it allows to handle multiple labels for each input data. Finally, the custom `MultiTaskDataset` allows for correctly split the obtained data into batches of the specified length for training and testing purposes.

The loss functions used are both cross-entropy loss, as both tasks involve classification problems with more than two classes. The optimizer and learning rate scheduler chosen are the same as those adopted for the action recognition neural network, Adam and the multi-step learning rate scheduler, respectively, because of their efficiency and reliability for classification problems. The initial learning rate for the Adam optimizer is set to 0.01. The milestones for the multi-step learning rate scheduler have been moved a little sooner than the one previously used for recognition, specifically to 20 and 40, as this network has to be stopped earlier in order to avoid overfitting. The gamma decay parameter has been set again to 0.1. The training loop is set to a maximum of 100 epochs, which is never reached because of the early stopping mechanism, which a patience of 15 epochs. The model's best weight restoring in early stopping is disabled, since the training process was observed to be short and stable after a few epochs, meaning that keeping the weights from the last epoch has a better effect than restoring the best weights.

It is worth noting how, despite multiple loss functions, one for each task to be classified, the optimizer always remains one. In fact, a peculiarity of MTL models is that they combine the loss functions of the various tasks, being able to adjust the weights as well, in order to modify the relative importance that the model attributes to the correct classification of one task over another [103]. It has been chosen to give the prediction of actions double the importance compared to that of the objects held in the hand, as they are more important for the correct reconstruction of the assembly sequence. Therefore, when summing the losses, that relating to actions is multiplied by 1.0 (that is a useless from a mathematical point of view but it is irrelevant in practice) while that relating to objects is multiplied by 0.5. The total loss thus obtained is used for the backward pass and the optimization. This process is repeated for every batch, in every epoch.

Testing and saving

This program, like the one described earlier for action recognition, contains a section dedicated to testing and saving the trained models. Both saving formats with the `state_dict` and `TorchScript` are available. The performance metrics used to evaluate the model are the same as those used for action recognition models: accuracy, precision, recall, F1-score, and ROC-AUC score. Since the model is trained on two tasks, concerning action and object prediction, they are evaluated on both tasks separately, to provide a comprehensive assessment of the model's performance. Confusion matrices and ROC curves are also generated for both tasks.

Particularly interesting is the function `search_for_future_action`, which has been provided here for testing purposes, and will be actually implemented in real-time. This function allows the model's predictions to be extended deep into the future, based on previous predictions, searching for a specific action. More information about this is provided in Sec. 3.3.3, speaking about action anticipation, where the use of this fundamental function will also be explained.

Real-time implementation

Speaking of real-time implementation in the robot program, successfully recognized actions are registered to build the data sequence necessary for action prediction using the `register_action_in_sequence` function, dividing actions, right hand objects and left hand objects in specific lists. As underlined before, objects held in the left hand have not been used in the related work, but they are anyway registered to be implemented in a possible two-handed independent system. The last part of these lists is then used to perform the actual prediction, according to the prediction window size. The full lists can also be saved as sequences for the action prediction model training, with the correct format, through the `save_observation` function.

3.3.3 Action anticipation

In Sec. 2.5, on the state-of-the-art of prediction and anticipation, it has been pointed out how task anticipation, and more specifically action anticipation, is allowed by prediction and leads to a more natural and intuitive human-robot interaction. Once the prediction of next actions is available, anticipating them means having the robot perform one of them, instead of the human, in a way that is useful for the assembly cycle that is taking place in collaboration with the human.

In-depth prediction

The action prediction model is able to perform forecasts about the next possible action. This is done in the real-time implementation by using the function `predict_next_action`. The actions that the robot is able to perform are limited by certain aspects such as the tools available, the end-effector used, and the size and characteristics of the objects to interact with. This means that not all actions in the assembly sequence can be carried out by the robot. So, predicting only one action in the future may be limiting in some situations and more in-depth predictions may be required. This is why a dedicated function, named `search_for_future_action`, has been developed. This function actually uses the `predict_next_action` function to forecast the next most probable action in the sequence, based on the recognized ones. The difference here is that a specific action is provided to this function to indicate the future action to look for. If the predicted next action does not match the searched one, this is appended to the sequence, the prediction window is shifted to maintain the size, and the window is re-input into the action prediction model. This process is repeated iteratively until the corresponding action is found. In this way, this function allows a deep search of future possible actions based on the learned assembly sequences, returning a chain of predictions in which the last one is the one searched. Hence the decision to give greater weight to actions rather than objects, as they are central to defining the assembly sequence. Obviously, the confidence associated with the predictions of actions that the model performs in depth must take into account the fact that they were themselves based on other predictions, i.e., conditional probability. Therefore, to return the overall confidence value, the probabilities of the prediction chain are multiplied by the aforementioned function. This multiplication concerns only the actions, since they define the structure of the sequence, and not the objects, which have no direct correlation with the other objects in the sequence. In addition, idle predicted actions are not considered in the conditional probability calculation for actions because they are irregularly present in both the hand-built sequence variations dataset and the real sequences dataset. These confidence values are then used to decide whether it is safe to anticipate the predicted action. Another boolean parameter named `force_object` can be provided to the function. As the name suggest, if it is set to `True` and if the object found for the specified action is none, the second most probable object will be returned. In this way, it is still possible to obtain a plausible object for the pick action. An acceptance threshold based on minimum confidence will then be defined. Below is reported the pseudo-code of the `search_for_future_action` function just described.

Algorithm 1 search_for_future_action - Perform in-depth prediction by searching for a specific future action, based on sequence data

```
Require: sequence_data, action_to_find, force_object
 1: Initialize one_hot_sequence_data
 2: for each action_pair in sequence_data do
 3:     Append concatenated one-hot encodings of action_pair to
        one_hot_sequence_data
 4: end for
 5: Initialize prediction_pairs, confidences, object_forced as False
 6: while True do
 7:     $prob1, prob2$ ← predict_next_action(one_hot_sequence_data)
 8:     $action\_prediction$ ← argmax($prob1$)
 9:     $object\_predictions$ ← sorted indices of $prob2$
10:     if Action(action_prediction) == action_to_find then
11:         if force_object and object_predictions[0] == len(OBJECT_NAMES)
            then
12:             Append forced object pair to prediction_pairs and confidences
13:             Set object_forced to True
14:         else
15:             Append predicted pair to prediction_pairs and confidences
16:         end if
17:         break
18:     else
19:         Append predicted pair to prediction_pairs and confidences
20:         Append concatenated one-hot encodings of predicted pair to
            one_hot_sequence_data
21:     end if
22: end while
23: Compute action_conditional_probability ignoring IDLE actions
24: return prediction_pairs, tuple((action_conditional_probability, confidences[-
    1][1])), object_forced
```

In the related work, only the action of pick among those recognized by the program was considered as executable by the robot instead of the human. Indeed, in order for the robot to assemble (place) an object, it would first require an additional vision system integral to the robot that would allow a close and appropriate view of the area involved in the assembly, which is generally not difficult to implement, but it would primarily require an intra-object detection model that can understand in which area it should actually be placed. For example, every component with holes for inserting other parts should be sampled in all of them so that insertion can take place properly. To obtain this result, another YOLO model should be specifically trained, with a large amount of data containing all possible critical points for assembling the objects of interest. Acceptable results for intra-object detection may be difficult to achieve even with the state-of-the-art YOLO object detector due to high object occlusion [104]. Additionally, if the collaborative assembly have to take place under even slightly dynamic conditions, such as in mid-air with the assembly held by the human, it would require a complicated algorithm to achieve the flexibility and understanding that two humans collaborating in such a situation might exhibit. The same considerations apply to the screwing action, which should in any case be performed with an appropriate collaborative screwing tool.

Considering only the picking action means that, for the tests performed, the robot is subservient to the human and has the only task of picking up and delivering objects. This mode of collaborative usage of the robot is certainly not the most flexible and comprehensive that can be realized. Nevertheless, it remains an excellent example to show the usefulness of anticipating human actions in a human-robot collaborative scenario. Moreover, the decision-making power of which object to pick up is completely in the hands of the robot's program, with no need for guidance from the human.

The action confidence threshold has been set to 0.8, to ensure an adequate level of safety in performing the picking action. The object confidence threshold has been set to 0.6, as it is generally lower, given the larger number of classes and the less weight given in training the model. The conditional probability for the action prediction confidence and the last object prediction confidence are returned by the `search_for_future_action` function and compared to these thresholds. These thresholds have been chosen in such a way that the robot performs only actions it is sufficiently confident of, otherwise it is preferable for it to remain idle rather than perform a potentially incorrect action.

Camera calibration

To physically perform any action, including picking, based on the camera view, the camera must be calibrated. To perform the calibration, the MATLAB tool `Camera Calibrator`, present in the `Computer Vision Toolbox` [105] has been used. This tool allows to estimate the geometric parameters of a camera. The official calibration grid attached to the tool documentation has been printed and fixed on a cardboard to give it rigidity. This grid must be captured in full by the camera to be calibrated at various positions within the viewing area of it.

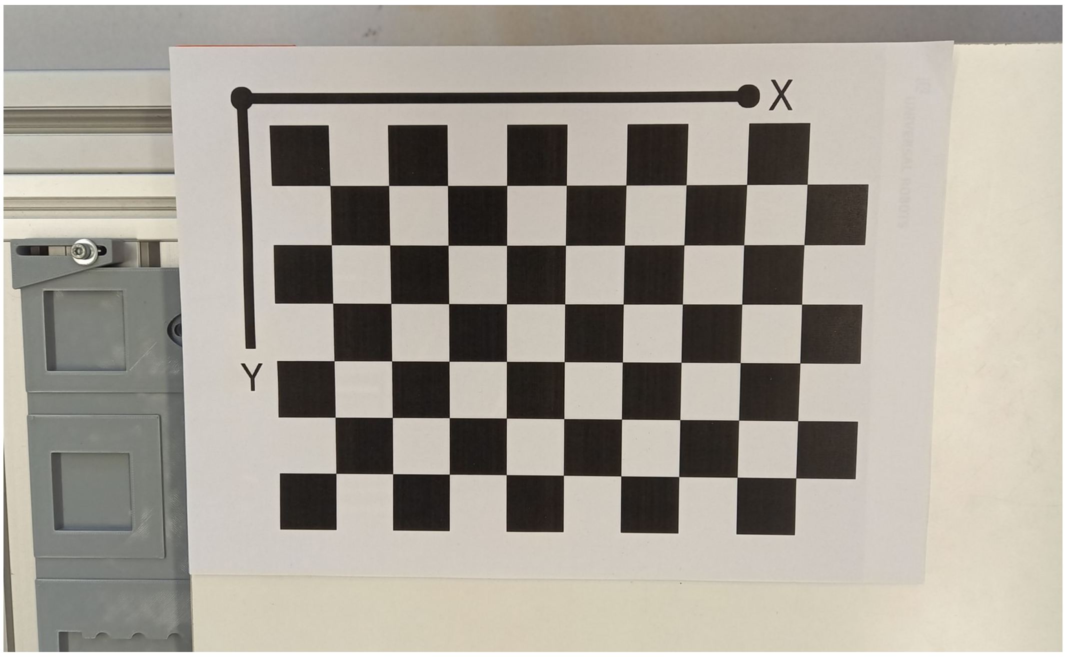


Figure 3.14: Calibration grid

The Camera Calibrator tool has been set to acquire 20 images, with an interval of 5 seconds to allow enough time to move the calibration grid in a different position. A known image, which is usually the first for convenience, should contain the calibration grid in a notable position so that it can be reproduced easily and can be used to calculate the transformation matrix. Specifically, the grid was placed with two sides coinciding with those of the working table, as visible in Fig. 3.15.

Once all the images have been acquired, the real-world dimensions of the square side of the calibration grid have to be provided to the program. From this, the geometric parameters of the camera are estimated and can be exported to the MATLAB workspace. It also shows the overall mean reprojection error in pixels and the one for each captured image. This information is useful to eventually discard calibration images with higher errors. If the calibration grid is particularly distorted in some images, the tool will automatically discard them.

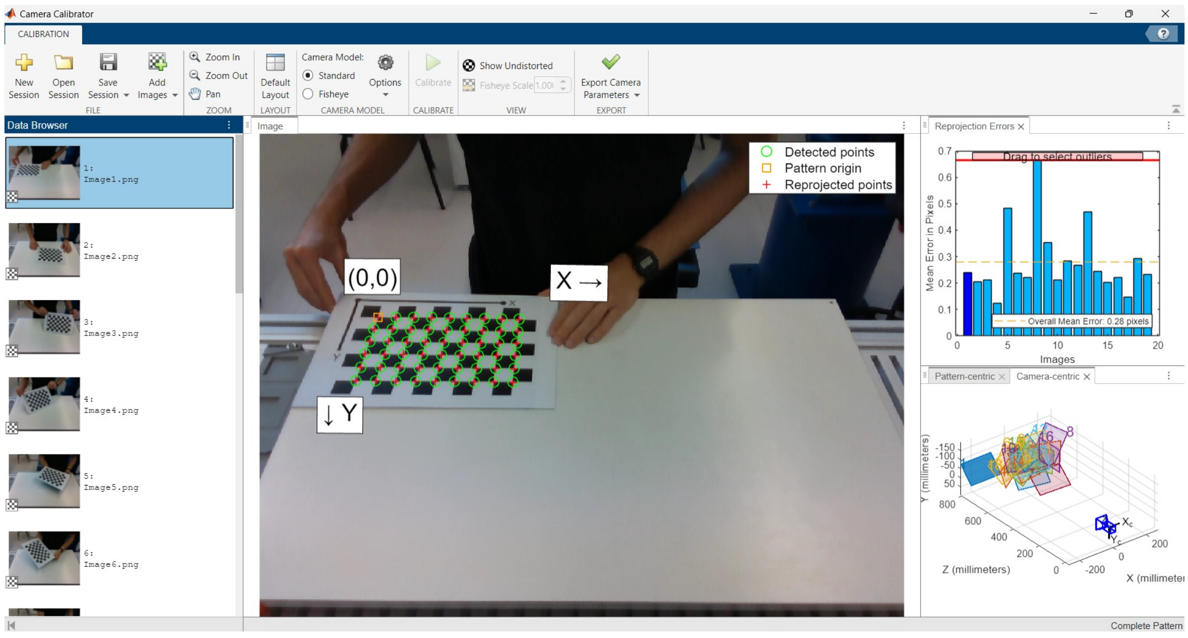


Figure 3.15: Camera Calibrator tool with acquired images

In MATLAB, a short script has been developed to obtain the transformation matrix from the camera reference system to the robot reference system ${}^{R}_{C}\mathbf{T}$. This can be found in the same GitHub repository as the robot program [91]. The camera parameters returned by the calibration tool provide camera-grid rotation matrices ${}^{G}_{C}\mathbf{R}$ and grid position vectors in the camera reference system ${}^{C}\mathbf{p}_G$. In particular, parameters related to the image containing the calibration grid in the notable position have to be selected, in this case the first image, showed in Fig. 3.15. Then, the reference system of the grid position vector has to be changed to the grid reference system, to match that of the rotation matrix.

$$ {}^{G}\mathbf{p}_C = -{}^{G}_{C}\mathbf{R}\,{}^{C}\mathbf{p}_G $$

In this way, it can be combined with the obtained rotation matrix to get the camera-grid transformation matrix:

$$ {}^{G}_{C}\mathbf{T} = \left[\begin{array}{ccc:c} & {}^{G}_{C}\mathbf{R} & & {}^{G}\mathbf{p}_C \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] $$

From this, to get to the transformation matrix from the camera to the robot reference system ${}^{R}_{C}\mathbf{T}$, the transformation matrix from the grid to the robot reference system ${}^{R}_{G}\mathbf{T}$ is needed, so that it can be composed with the ${}^{G}_{C}\mathbf{T}$ matrix just obtained. The corresponding rotation matrix ${}^{G}_{C}\mathbf{R}$ can be found simply by making explicit the rotations that exist between the two reference systems. Since the robot is fixed on the same horizontal working plane in which the grid is located, this matrix is composed of only elements 0, 1 and -1. The grid position vector in the robot reference system ${}^{R}\mathbf{p}_G$ is found by moving the robot end-effector to the grid origin, either through the teach pendant or with the dedicated Python functions that will be discussed later. The position was then obtained and reported in the MATLAB script. At this point there are the elements to compose the transformation matrix ${}^{R}_{G}\mathbf{T}$, and, finally, to compose the transformation matrix searched:

$$ {}^{R}_{C}\mathbf{T} = {}^{R}_{G}\mathbf{T}\,{}^{G}_{C}\mathbf{T} $$

The values of this matrix were reported in Python, in the class used for communication with the robot, which will be described below.

Robot control

The KUKA LBR iiwa robot in use can be programmed with the KUKA Sunrise.OS toolboxes, which support Java as a programming language. A Java server, implementing a MATLAB script, runs on the robot controller. A dedicated Python library, named `iiwaPy3` [106], has been used together with this server to enable communication with a Python program. A wrapper class, named `RobotController`, has been created around this library to facilitate the connection, to easily allow coordinate transformation using to the previously calculated transformation matrix ${}^{R}_{C}\mathbf{T}$, and to add parameters and methods useful for the application under consideration. Because of the way the server that runs in the robot controller works, the tool center point (TCP) must be defined at every connection. For this reason, the coordinates of the TCP with respect to the robot flange have been measured and reported in the connection method of this class.

Figure 3.16: The Java server running on the robot controller

Real-time action anticipation

When the robot program has sufficient confidence in the predicted picking action, it will be executed, anticipating the human. The `pick_and_deliver_object` function has been designed to accomplish this task. Here, the objects visible by the camera are obtained from the object tracker. These are scanned for an object with the same label id as the one being searched for and suitable for picking, i.e., one that is not moving and not already held by the human. At this point, the coordinates of the center of the object's bounding box in the camera frame are calculated, and the depth of the same point is obtained by using the depth sensor. More information about obtaining depth values can be found in Sec. 3.4, related to data acquisition. Hence, the triad of coordinates in the camera reference system is available. The `deproject_pixel_to_point` method of the `RealSenseCamera` class is applied to this triad, to obtain real-world values using the camera's intrinsic parameters, i.e., ${}^{C}\mathbf{p}_{obj}$. Next, the camera-robot transformation matrix ${}^{R}_{C}\mathbf{T}$ obtained earlier is applied through the `cam_to_robot_transform` function.

In this way, the coordinates of the center of the object to be picked up are obtained in the robot's base reference system.

$$ {}^{R}\mathbf{p}_{obj} = {}^{R}_{C}\mathbf{T}\,{}^{C}\mathbf{p}_{obj} $$

From this, an approach position to the object is also defined, with the same x and y real-world coordinates and a fixed z of $100\,mm$, which allows approaching from above without interfering with other objects in the working plane. The last step before performing the picking action is the control of the effective reachability of the picking and approach positions by the robot, without incurring singularity conditions or getting excessively close to them. The `check_workspace` method of the `RobotController` class is used for this purpose.

If one of the tracked objects meets all the requirements just described, the robot decides to anticipate the human action, in this case to pick up an object from the working table. At this point, after picking up the object, the robot has to deliver it to the human. It is good that this stage, where the actual interaction with the human takes place, is as natural and intuitive as possible. For this reason, it has been developed a system that delivers directly into the human's hand, the right hand in this case. The process of obtaining delivery coordinates is similar to that described above for the picking task. The hand centers are provided directly by the `HandHelper` class; they are marked with yellow dots in Fig. 3.17. A loop is established in which the coordinates of the center of the hand of interest (the right) are continuously obtained. This is to allow the human to dynamically change the hand position during the delivering operation. The delivery position is defined as the center of the hand, with a z coordinate increased by $50\,mm$ to allow the object to be passed easily. A tolerance distance for the delivery position is also defined. Within this loop, the robot end-effector is continuously commanded to the updated delivery position. If the distance between the reached position and the current delivery position is less than the tolerance threshold, the object is released and thus delivered to the human.

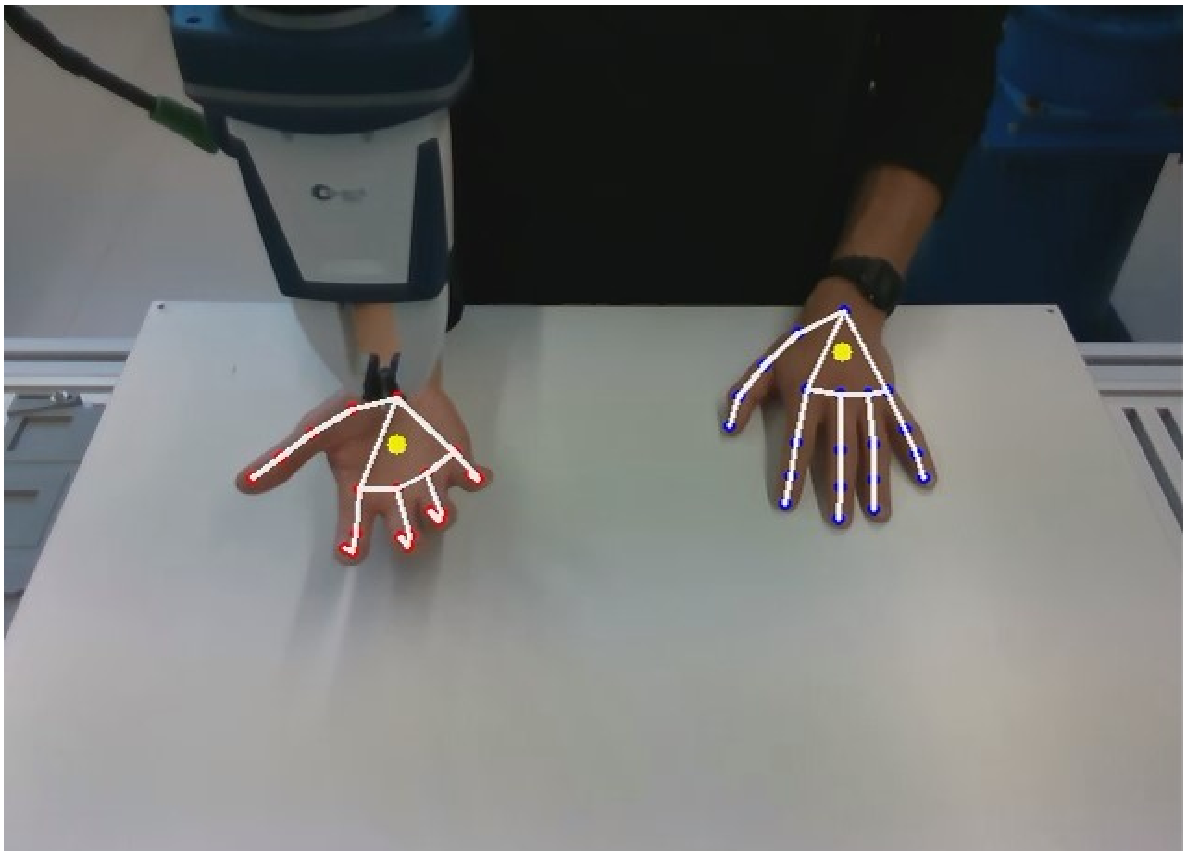


Figure 3.17: Robot delivering the picked object in the hand (hand centers marked with yellow dots)

Since the picking action performed by the robot anticipates and replaces a human action within the assembly sequence, but the actual picking motion is not performed by the human, so it is not recognizable by the action recognition model, it is added programmatically to the sequence, along with the picked and delivered object. In addition, the object detector struggles to correctly recognize the object being delivered by the robot to the human operator. However, since the information about the object picked and delivered is well known within the robot's program, this object is forced into the object tracker as the object in the human's (right) hand through the function `force_object_in_hand`.

After the registration of the action-object pair in the sequence, the prediction of the next searched action is performed, in this case always the pick action. If the prediction meets the set confidence thresholds and the predicted object is available within the workspace, the robot proceeds with the anticipation of this action immediately after delivering the previous object, otherwise it returns to the home position. In this way, a great degree of fluency can be achieved in human-robot collaboration.

The `iiwaPy3` library used to control the robot executes each movement command synchronously, meaning that the program is paused until the movement is completed. This behavior is incompatible with the use of the developed object tracker and affects negatively the tracking of hand movements performed by MediaPipe. For this reason, the `pick_and_deliver_object` function has been implemented within a thread, in order to make it a background process. Furthermore, the function, hence the thread, has been structured to work with an execution queue, so that multiple actions to anticipate can be queued and executed sequentially. The action recognition and the prediction of future ones is paused when the robot is delivering an object, as the vision of the camera is often obstructed by the robot and this makes correct recognition difficult. With a more complex hardware setup that uses multiple cameras, the execution queue could be very useful to increase the naturalness of interaction with the robot, or even interrupt or modify anticipated actions while they are being performed if further human actions are detected in the meantime.

3.3.4 Object detection

As mentioned in Sec. 2.6, the state-of-the-art object detector YOLO has been implemented, in particular YOLOv9 [82], which is the latest version available at the time of this writing. It allows for optimal real-time object detection and has the strong advantage that can be trained on custom dataset, from scratch or fine-tuning a model pre-trained on big datasets. In this way, object detection can be achieved on application-specific objects. In addition, fine-tuning allows to obtain an excellent object detector with a relatively small dataset of custom objects. This section describes the steps taken to create the application-specific object dataset, starting with the labeling tool created to label the images, and the YOLO fine-tuning process.

Image labeling

Training YOLO on a dataset of custom objects requires a cumbersome process in which numerous images of objects have to be captured and labelled by hand. Labelling an image involves manually drawing a bounding box around the objects present and indicating their class index. In addition, the labels of each image must be saved in a text file following the formatting indicated in the documentation [107]. Several labeling tools can be found online, but it has been preferred to develop an own labeling tool to implement a specific library, for the purpose that will now be discussed. The library in question is the Segment Anything Model (SAM), by Meta AI [108]. This amazing model, trained on a huge dataset, produces high quality object masks from different input types. Before moving on, it is meant to describe the idea behind the use of this model and why it has no longer been implemented. The original intention was to use it so that the robot program could autonomously segment objects affected by actions recognized by the action recognition system. In this way, creating the custom dataset for YOLO training would require much less effort since the objects would already be segmented and would only need to be manually labeled with the correct class. This would add huge flexibility to the collaborative application, as it would reduce the training time in case new objects need to be used in the assembly process. The two main problems encountered, which led to the idea being discarded, are:

- The model developed for action recognition does not recognize stages of actions but classifies them as a whole. In fact, assuming that there is no model available for object detection, since the data to train it is still being collected, without a subdivision into stages of the action it is impossible to determine the plausible location of the object involved in the action. For example, suppose a pick action. This can be divided into three stages: approaching the hand to the object (pre-contact), grabbing the object (contact) and moving the hand away with the picked object (post-contact).

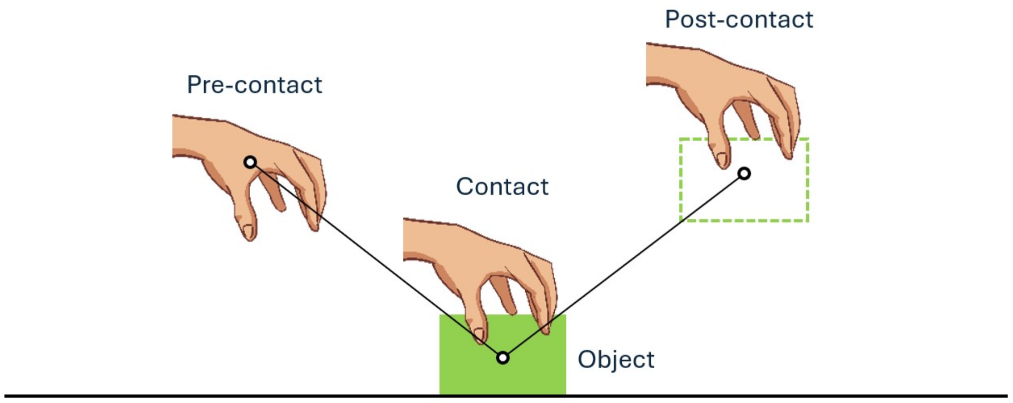


Figure 3.18: Pick action divided into stages

To achieve this subdivision, the action recognition model must be trained with a dataset of actions partitioned and labeled accordingly, increasing the overall complexity of the model. With this, it is possible to define the stage where the actual interaction with the object takes place, i.e., the contact stage, and consequently determine its position based on that of the hand involved. After that, it is necessary to have some memory buffer of the past frames, to trace back to a frame far enough in the past that the hand does not obstruct the view of the object, presumably at the beginning of the pre-contact stage. Segmentation via SAM must be applied to this frame, which cannot be done in real-time because of the heaviness of the model, completely understandable for what it can do, but which is not a problem since the frames of interest can be saved and processed offline later. Despite the technical difficulty of the process, this would have been willingly explored if it were not for the second problem encountered, which is more limiting than this one;

- The main problem, which is difficult to bypass, is that YOLO, to be properly trained, needs all the objects to be identified within the images to be labeled, not just some of them. If this were not the case, many of the objects for which it is being trained would be correctly identified but not matched in the provided labels, so they would be considered as incorrect identifications. This does not allow the model to be trained. Therefore, even if the object involved in the action is identified correctly, there would be no way to automatically detect all the others in the workspace. This would involve training the model with images that contain only one object to be identified, greatly increasing the number of images needed and reducing the flexibility advantage that would be gained by this automatic segmentation method. An alternative would be to crop the bounding box of the object affected by the action and train YOLO with only images of cropped objects. Although no research has been found that delves into the effects of training YOLO in this way, it is well known that is not recommended and generally leads to several problems, e.g., loss of contextual information, incorrect anchor boxes training, scale and position not being reproducible in full images, etc.

Although the initial idea was discarded, it has been chosen to go ahead with the development of a labeling tool powered by SAM, in order to investigate the actual performance and facilitate the labeling process. The technical difficulty described in the point one is not present here, as the input to segment the object is not obtained programmatically but is required to manually draw a bounding box around each object. SAM's segmentation capabilities and zero-shot performance, on even very particular objects and non-uniform backgrounds, are outstanding, as shown in Fig. 3.19.

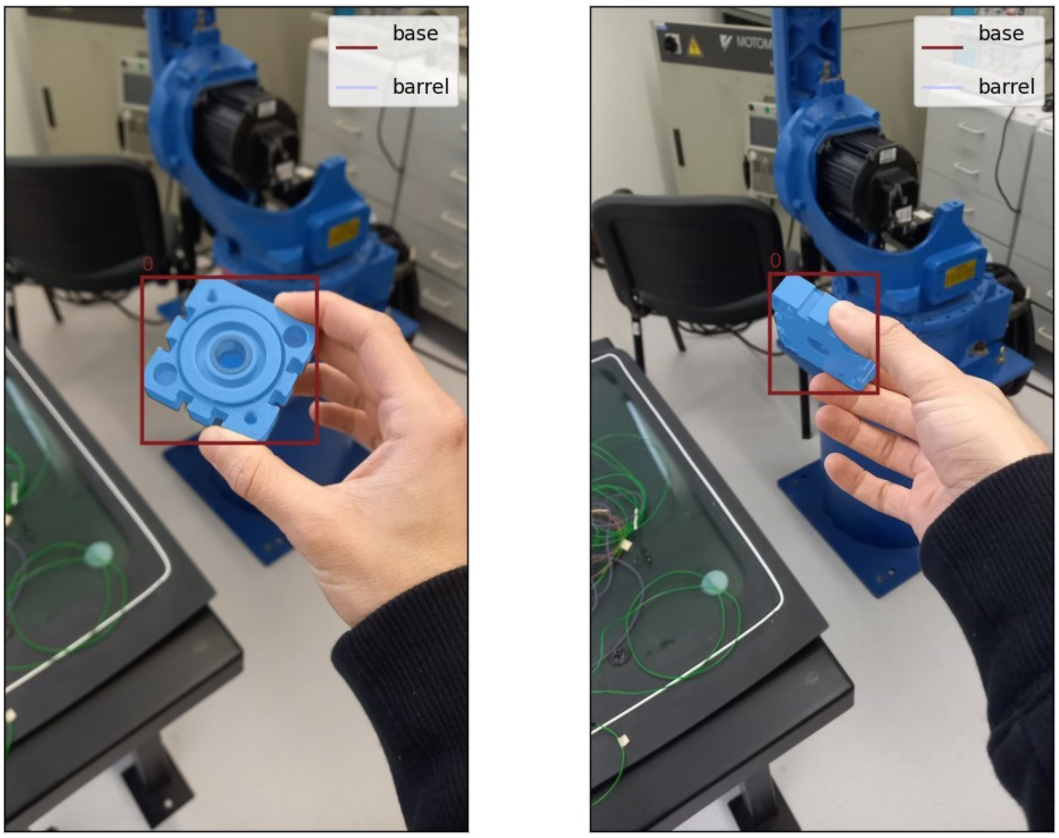
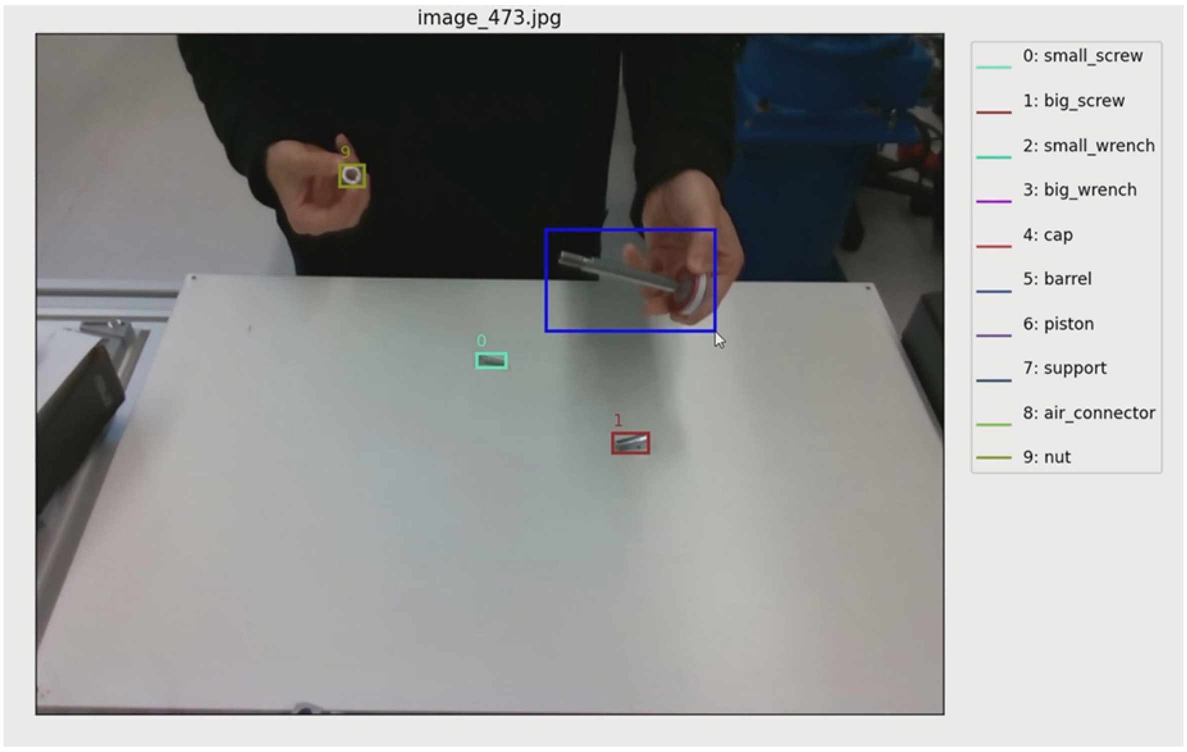


Figure 3.19: Example of SAM's segmentation on an object with complex edges

The developed tool allows to load images from a directory, draw bounding boxes around objects of interest, assign labels, and save the labels in YOLO format. It also comes with an image mirrorer to run after labeling, to perform data augmentation. It horizontally mirrors the images, including the corresponding labels. All images were used in their original and mirrored states, doubling the data available for training the model. Finally, another script has been developed to execute the train-validation split, required to properly train YOLO. It operates within the same folders used for images and labels and the validation size can be set as a percentage of the available images. This useful tool, along with the other utility scripts, has been released as an open-source project on GitHub [109], so that everyone can test and use it. Refer to the dedicated documentation for more information.

An example of the use of the labeling tool is shown in Fig. 3.20. It can be seen that the roughly manually drawn bounding box is turned into an accurate bounding box that perfectly surrounds the object.



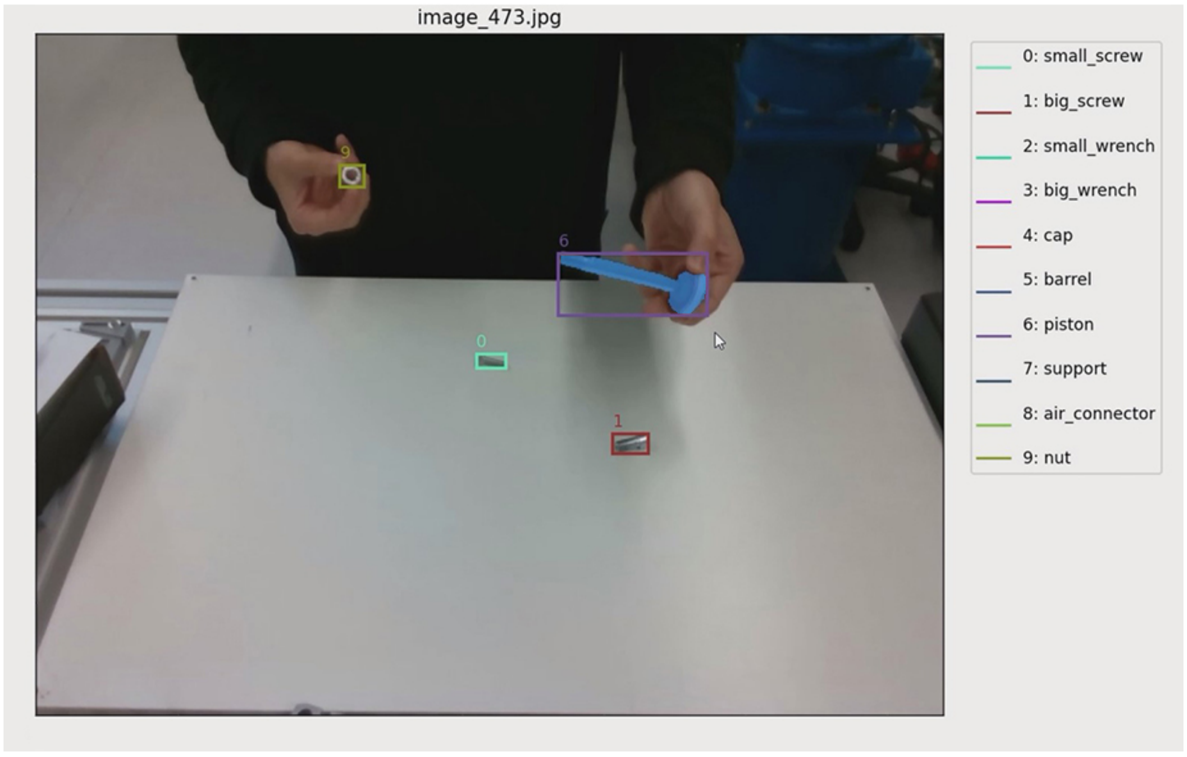


Figure 3.20: Screenshots of the use of `SAM-YOLO Image Labeling Tool`

SAM extracts masks from the objects, which could be used to train an object segmentation model. The tool is designed to create bounding boxes out of the object's mask. The masks of selected objects are displayed, but ignored at the moment of saving, as only the bounding boxes are needed for the object detection purposes for which YOLO has been used. The extension of the tool to also save object masks is fast and may be done in the future.

Dataset

A total of 10 labels have been identified among the application-specific objects. Their classification has already been provided in the problem definition Sec. 1.3. A list of them is reported here with the specific ids and names used in YOLO training and in the rest of the robot program:

0. small_screw;
1. big_screw;
2. small_wrench;
3. big_wrench;
4. cap;
5. barrel;
6. piston;
7. support;
8. air_connector;
9. nut.

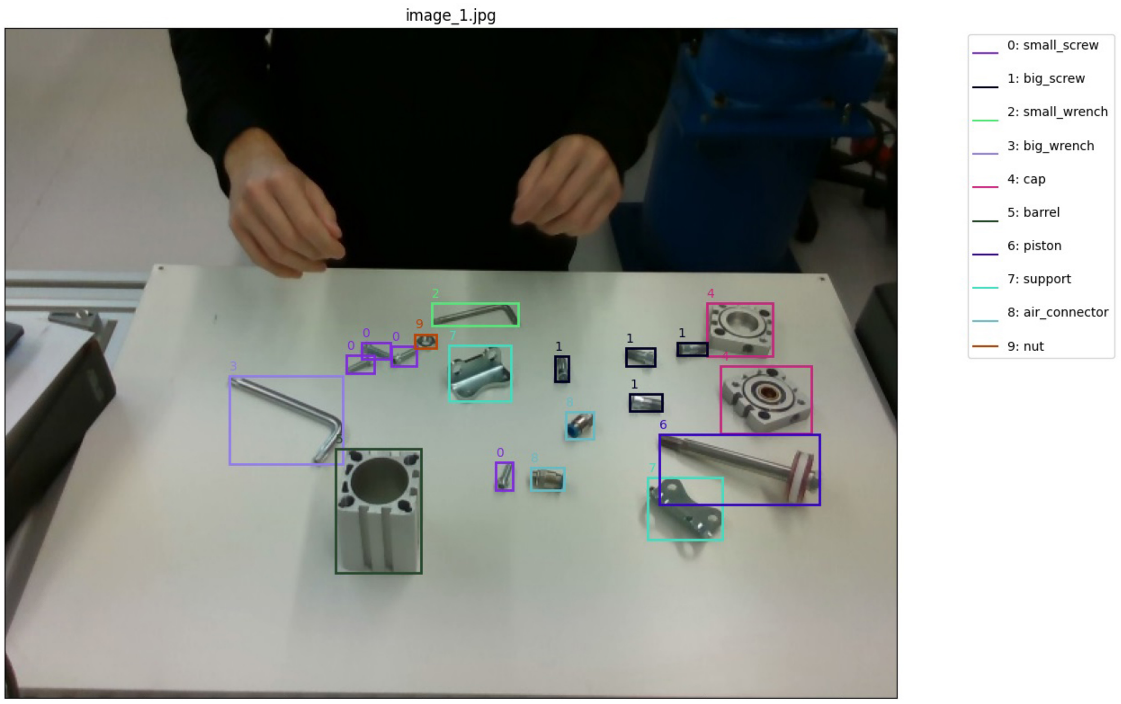


Figure 3.21: Image of the dataset including all the objects in the assembly, labeled using the `SAM-YOLO Image Labeling Tool`

The two caps, upper and lower, are similar but structurally different, as the upper one contains a hole for the piston rod, while the lower one is completely closed. The reason they have not been differentiated but considered as a single class "cap" is because the assembly most likely starts with one of the two caps handled by the human in the first or second useful action, when the action prediction model still does not have enough data to predict the next action, leaving only the other cap available to the robot. Therefore, it has been decided to consider a single class for caps to slightly simplify the dataset construction. Conversely, distinguishing between the two screws, small and large, has been strictly necessary to differentiate these two similar objects that have distinctly different roles in the assembly of the pneumatic cylinder. Care has been taken to extensively sample the two screws to allow better training on these two very similar objects. Below is a figure showing how similar the two screws are in shape and size.

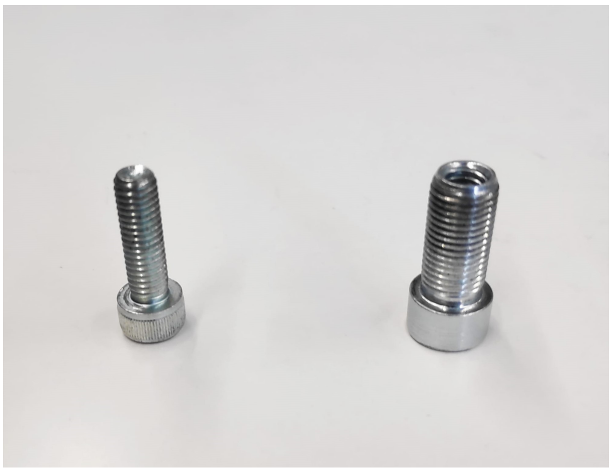


Figure 3.22: "Small" and "big" screws comparison

Once the images of the objects in the workspace have been collected, with some of them being discarded because some objects were barely visible, they have been labeled with the `SAM-YOLO Image Labeling Tool`. In total, 444 images have been labeled, with an average of 2.96 objects per image. These have also been mirrored with the mirroring tool, to perform data augmentation and improve training, for a total of 888 labeled images.

It is worth noting that the images used to build the training dataset for fine-tuning have all been acquired with the camera used to develop the related computer vision system, in the fixed working position. This reduces the variability of the context in which the objects are located, reducing the flexibility of the trained object detection model to the specific application, but at the same time requiring much less data for training.

The most recent guidelines for constructing an optimal database for training YOLO are those for YOLOv5 [110], which can reasonably be considered valid for YOLOv9 as well, since they are kept up to date. These recommend that each class should appear in at least 1500 different images and that there should be at least 10000 labeled instances for each class in total. These values are far from those adopted for the constructed dataset. However, considering what has been said about the intention to create an application-specific object detection model, which therefore requires fewer samples because they are acquired in the same context as the inference one, and the fact that it will be fine-tuned a pre-trained model, the samples produced have been deemed sufficient. Fig. 3.23 shows the number of instances for each class and bounding boxes spatial distribution in the application-specific object dataset, in particular the train dataset. It can be seen that more samples have been voluntarily acquired for the two analyzed screws, specifically to better train the model to distinguish these two classes, which are the most similar. It can also be noticed that the spatial distribution of the bounding boxes is mirrored horizontally, due to the data augmentation performed by mirroring images and labels.

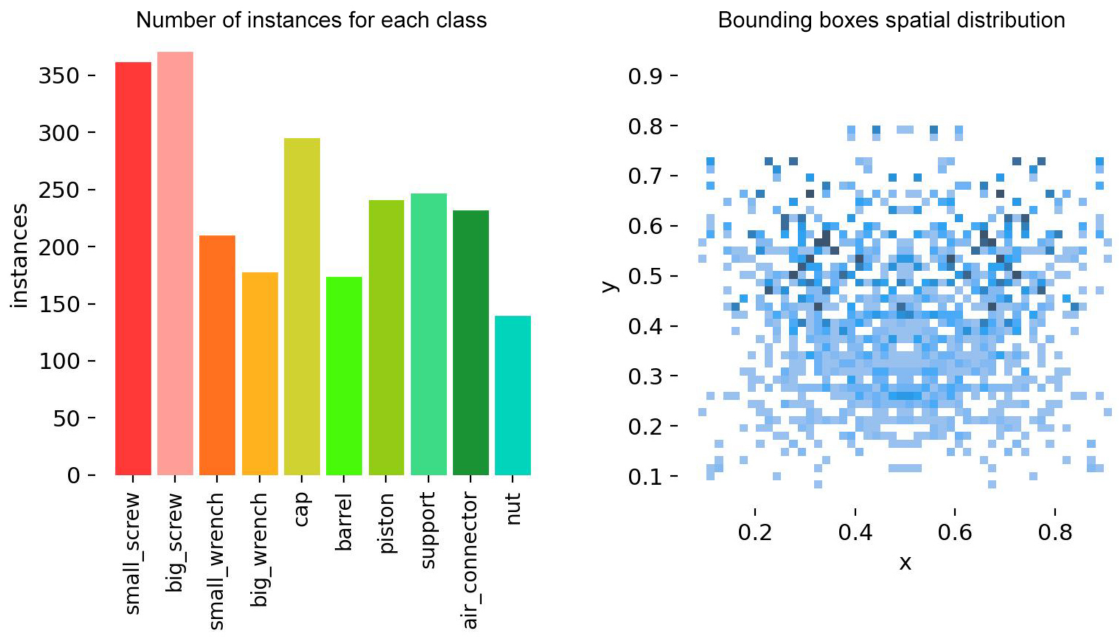


Figure 3.23: Number of instances for each class and bounding boxes spatial distribution in the application-specific object train dataset

Training (fine-tuning)

After labeling the images, they have been split into train and validation sets using the `yolo_train_val_split.py` script included in the labeling tool, with 90% for the train set and 10% for the validation set. At this point, a simple script has been developed to enable the YOLO training via the SDK provided by `Ultralytics`. This is available in the `yolo_train.ipynb` notebook. The `YOLOv9c` model, pre-trained on MS COCO (Microsoft Common Objects in Context) dataset [111], has been chosen as the starting point for fine-tuning with the application-specific object dataset that has been created. The training script has been executed on a Google Colab GPU environment, since much computational power is required to train YOLO. The environment used featured a NVIDIA T4 Tensor Core GPU of 16 GB of memory, with CUDA 12.2 installed to enable GPU-accelerated operations. The images, along with their respective labels, divided into the train and validation sets, have been uploaded to Google Drive, which was mounted within Google Colab's Linux environment. These are the fine-tuning configuration parameters used:

- Pre-trained model: YOLOv9c;

- Batch size: 16;
- Epochs: 100;
- Target image size: 640 (longest side is resized to 640 while maintaining the original aspect ratio, so 640x480 pixels);
- Optimizer: Stochastic Gradient Descent (SGD) with momentum;
- Initial learning rate: 0.01;
- Momentum: 0.937.

The results will be discussed in detail in the relevant Sec. 4.4 of the results chapter. The weights of the fine-tuned model have then been exported to be used locally within the robot program.

3.3.5 Object tracking

As anticipated, a dedicated custom multi-object tracker has been developed, in order to satisfy the features required for the application. To understand the need for a multi-object tracker, remember that action prediction is closely related to action recognition. The latter occurs first, and when a new action is recognized, it is registered in the sequence, to enable the prediction of future actions. Along with actions, manipulated objects are also registered in the sequence. As previously mentioned, the action recognition model that has been developed does not recognize stages of actions but classifies them as a whole. In fact, it has been trained with a dataset of clipped actions, as described in detail in Sec. 3.3.1, which represent complete actions. Refer now to Fig. 3.18, that shows three stages into which the pick action can be divided. Because of the way the action recognition model has been trained, the actual recognition of the pick action occurs not before the contact stage, but could also occur at the post-contact stage. In contact or post-contact stages, the object involved in the action is held in the hand, so its view by the camera, and thus object detection via YOLO, is partially or completely obstructed, depending on the size of the object. This means that, if the system relied only on real-time object detection, i.e., relative to the frame in which the action is recognized with sufficient confidence, the object detected as being held in the hand would in most cases be null due to total obstruction, or otherwise potentially incorrect due to partial obstruction. This is unacceptable for the purpose of sequence creation for correct prediction of future actions.

Consider now how the same situation would be having a human being who has to recognize the action and classify the involved object, observing the unfolding of the entire action. Again, the view of the object would be partially or totally obstructed during the contact or post-contact stages, when the recognition of the action can be made with sufficient confidence by the human. The difference is that the human is easily able to use short-term memory mechanisms that enable him to associate the object seen just before the contact with the hand of the person performing the action, asserting with absolute certainty that now that object is in the hand, even though it is not visible at the moment. This short-term memory mechanism, which is very natural for a human being, is far from simple for a computer vision system and falls into the category of object tracking.

The developed multi-object tracker is inspired by `BoT-SORT` [86] and `ByteTrack` [87], which are directly usable with YOLO with the `Ultralytics` library. The structure is simplified compared to these. The distinguishing feature is the implementation of hand tracking with MediaPipe and a dedicated hand-held object tracking logic. This tracker has been documented and released as a separate open-source project on GitHub [112], so that it can be used for any project where tracking of objects interacting with hands is needed, as it is considered incredibly useful and effective.

The desired results have been achieved with the developed tracker, showing very good tracking of objects held in the hand, which was the main focus, with very good re-identification after obstruction. A detailed analysis of the results will be presented in the dedicated Sec. 4.5.

Object tracker description

The main class of the tracker is the `ObjectTracker` class. The main method is `register_seen_objects`, which is meant to be used within the frame acquisition loop of the camera. At each loop iteration, the objects are detected by YOLO from the RGB acquired frame, in this case the application-specific objects for which it has been properly trained. These are then used to create instances of `YoloObject`, which is a class used by the tracker to group information about label id, detection confidence and bounding box corner coordinates for each detected object. Next, these instances are registered in the tracker. The midpoints between the tips of the thumb and index finger of both hands are also provided, as they have been chosen to define the most plausible point of contact of an object with the hand during the manipulation to perform an action. They are obtained from MediaPipe hand landmarks through the `HandHelper` class. After that, the `increment_frame_index` method is used within the frame acquisition loop to increment the frame index to keep track of the frame count and handle expiration and false seen logic.

The main features of the developed tracker are reported here:

- Tracks visible objects, associating each one with a unique id which is maintained across frames, as a common multi-object tracker;
- Associates tracked objects with detected hand positions using MediaPipe, specifically the tips midpoints, to determine which object is being held in the hand;
- Manages the state of objects being moved;
- Keeps tracking objects presumably in hand even if they are not actually visible to YOLO;
- Handles the re-identification of objects when they reappear after being hidden by hand;
- Maintains a list of active and expired objects, with a patience logic to wait for re-appearance.

One aim of the work was to make the application of the developed models and tools as flexible as possible, so that human-robot collaboration is as natural as possible for humans. For this, among other things, efforts were made to make the object tracker suitable for working with objects randomly disposed on the working table, thus also very close to each other. Obviously, overlaying objects excessively could go beyond the limits of object detection by YOLO, on which the tracker depends in the first place. Therefore, staying within the limits of proper object detection, an attempt has been made to develop a feature for the tracker that would make it possible to distinguish which object is actually being picked up, even if multiple objects are in close proximity to the hand and, more importantly, are simultaneously not visible to the camera.

To accomplish this task, a human being would rely on the recognition of the actual grasping action, i.e., when clasping the fingers of the hand around an object, along with the short-term memory mechanism described earlier that allows to remember which object was in the current hand position, even though it is not visible at the moment. As mentioned previously, the recognition model is only able to classify the action as a whole and cannot distinguish the actual grasping action. Therefore, the mechanism to distinguish multiple covered objects cannot be based on the recognition of the specific grasping action, but can rely only on the current position of the hand, in particular the tips midpoint. This totally non-trivial feature has been implemented in the object tracker by adding a mechanism of object persistence in the hand, expressed in frames elapsed by the object in the proximity zone that makes it considered as being held in the hand. In this way, even if the hand temporarily passes through several objects to reach the desired one, they will not be locked by the tracker as hand-held because the persistence is not extended enough. When the actual object to be picked is reached, the picking action performed should keep the object within the hand-held area long enough to lock it for the tracker. This value of required in-hand persistence frames is adjustable within the tracker, also depending on the acquisition FPS. In the real-time implementation, it has been set to 3, to account for the dynamic nature of the actions performed. In Fig. 3.24, it is shown an example of how the tracker was able to effectively identify the grasped screw among other nearby ones. Most importantly, all the screws placed between the human operator and the grasped screw were crossed by the hand before reaching the grasping position, but were effectively recognized as transient objects and thus ignored. To get the figure and demonstrate the functionality, the bounding box of the object was made visible as long as it was considered to be in hand by the tracker even though the object was not actually visible.

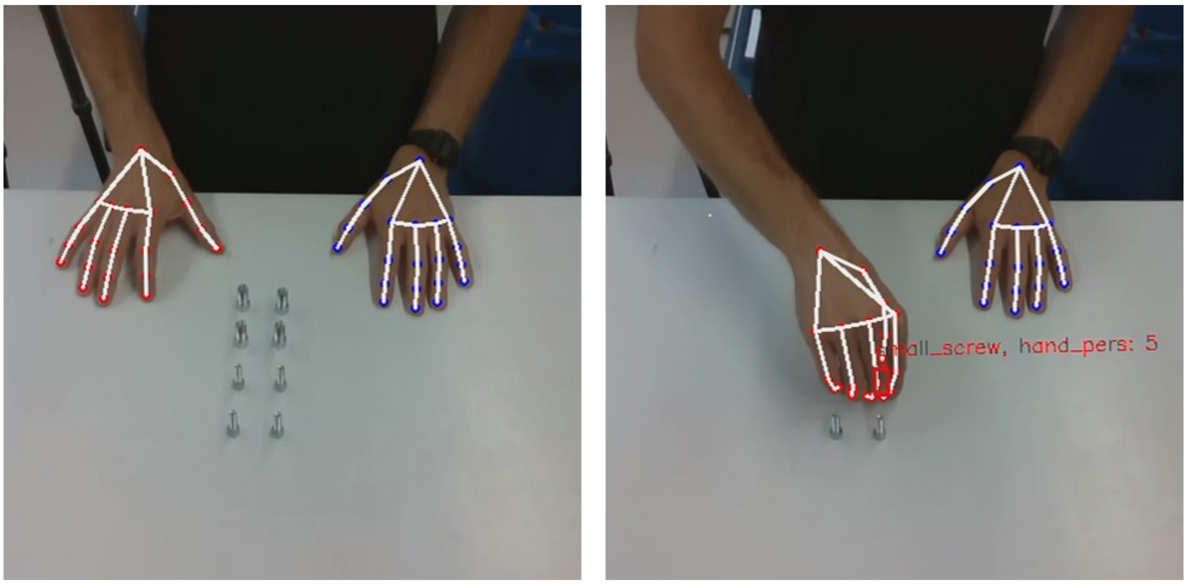


Figure 3.24: Object tracker hand persistence example

Object tracker algorithm

The algorithm behind the developed object tracker is here described. Since it is quite complex, it is considered more effective to describe the logic of the algorithm's operation than to provide pseudo-code, which would be more difficult to understand.

The main functions of the tracker have been listed above. As mentioned, the main method is `register_seen_objects`, which is used within the camera acquisition loop to register the objects detected by YOLO, updating the state of the objects being tracked. The following are the main steps of this method, in the exact order in which they are executed. The order is critical to the final tracking result obtained.

- Visible objects logic: the list of objects detected by YOLO in the current frame, thus actually visible, is iterated. For each of these objects, the object closest to it is searched among all objects with the same label id already in the tracking system, taking the center point of the object's bounding box as a reference. If at least one object with the same label id is found and, more importantly, if its distance is less than a set threshold value, the object detected by YOLO is associated with the tracked object, meaning that the current visible object is considered to be that exact tracked object in the new frame. Each tracked object is identified with a unique tracker id, which is an incremental integer. Then, an additional, more restrictive threshold is tested on the distance to define whether the object is in motion or not. In fact, consider that the distance is being calculated between the position of the object in the tracker's memory, i.e., that of the previous frame, and the current position. This means that the measurement of the distance between these two positions corresponds to the displacement value during the elapsed time between the two frames, which, if above a certain threshold, makes the object considered to be moving. All objects detected by YOLO that have been associated with a tracked object are removed from the list of objects to be tracked. The remaining objects will be used later for the re-identification logic of previously hidden objects or for the creation of a new tracked object.

  The index of the current frame is also registered as the last frame in which the tracked object was seen, and the counter of frames in which the object was persisted is incremented by one. The first one is used to define which objects are considered to be expired because they are not visible for too many frames, the latter is used to verify that the visible object is not a false positive and therefore should not be tracked. Only the values of these parameters are registered in this method, while the logic related to these two operations is then performed during frame increment through the `increment_frame_index` method.

  The last operation performed for visible objects is the hand association. For each visible object, it is checked the proximity to the hands, using the midpoints of the thumb and index finger of each hand as a reference. If the object already tracked as held has the same tracker id as an object within a certain distance, defined by a threshold, the in-hand persistence value is incremented by one to enable the mechanism for locking objects in the hand, described above. If, instead, the hand is empty or the object associated with it has not reached the threshold of persistence frames required to consider the object stable in the hand, the object in the hand is replaced with the newly detected nearest one.

- Back to track from hand hide logic or create a new tracked object: the list of objects detected by YOLO, thus currently visible, in which all those associated with a tracked object have been removed, is now iterated. For each of these objects, it is checked whether a hand currently contains a non-visible object with the same label id and if this hand is near to the detected object, comparing the distance with a threshold value. If this match is found, it means that the object under consideration, which has been detected by YOLO but is not associated with any tracked object in the proximity, is most likely the same object that was held in the hand, thus no longer visible in previous frames, and it is re-associated with the respective tracked object. If no match with the held objects is found for the detected object, a new tracked object is created, with a new unique id given by the incremental integer, and added to the list of tracked objects.

- Hidden objects logic (possible hidden from hand): the list of tracked objects is iterated to search for objects in the list but not associated with any visible object. A tracked object may be in this condition for one of the following three reasons:

  - Object out of camera frame;
  - Object actually visible in the camera frame but not detected by YOLO in the current frame, i.e., an object detector's error;
  - Object hidden from camera view, potentially by a hand that is interacting with it.

  First, the state of these objects is set to not visible and in motion. After that, the first two cases are handled in the same way, that is, no further action is performed on these tracked objects. If the object is out of the camera frame, it is left to the function that defines expired objects to stop tracking it if it does not reappear in the frame within the set patience limit. Instead, in the case there is a non-detection by YOLO in the current frame, which is supposed to be performed correctly in subsequent frames, the object will be tracked back by the visible objects logic when it is detected again, maintaining the previous association. To determine whether the object falls into the last case, i.e., potentially held in a hand that therefore hides it, which is the most important case for the current tracker, the distance between its last available position in the tracking system and that of the current hand reference points is measured. These distances are compared to a defined threshold, which is slightly more permissive than the one used to define whether a visible object is in the hand, since it is referred to a position that is not current, but past. If the distance is below this threshold, the object is classified as being held in the respective hand and not visible, continuing to increase if necessary the in hand persistence value so that it can be locked.

- Hand release logic: in this last part of the function, it is searched for objects that are tracked as held in the hand and are currently visible, but are located away from the relative hand, beyond a set threshold. In this case, the object is considered as released from the hand, so its in-hand persistence is reset to 0 and its state is updated as not held in the hand.

As mentioned, the `register_seen_objects` function just described is always executed in tandem with the `increment_frame_index` function, which, in addition to incrementing the frame index as the name suggests, handles the logic of eliminating tracking for false seen objects (false positives by the object detector) or for objects that are no longer seen for an excessive number of frames, i.e., expired objects. In the first case, the patience value in frames for false seen has been set to 5. If an object among those tracked has a persistence value in frames less than this and is not visible, i.e., it has not been visible for that number of consecutive frames, this is considered a false detection by YOLO and thus eliminated from the tracking system. In the second case, there is an object that has already met the frames persistence threshold for the false detection check, so it is a stable tracked object, but has not been detected in the scene for several frames. The patience value in frames for considering a tracked object as expired has been set to 20. This value is multiplied by a coefficient greater than 1, which in particular has been set to 2, if the object under consideration is tracked as hand-held, in order to allow for proper re-association of the object once it reappears in the scene, even if the time for which it is held is extended. Unlike false seen objects, which are permanently removed from the tracking system, expired objects are removed from the actively tracked object list and are moved to the expired object list. In the present program, the list of expired objects is used only within the function `force_object_in_hand`. This allows an object to be programmatically set as held in the hand. Specifically, it is used within the program once the robot delivers an object into the human's hand. Since the object has been manipulated by the robot for the time it takes to perform the picking and delivering action, it is likely that the object has not been detected for several frames in the meantime and has therefore expired. So, the object is retrieved from this list to maintain proper tracking continuity and moved to the list of actively tracked objects, set as held in the right hand by the human. In addition to this useful use, many other uses can be based on this list of objects whose tracking has expired, e.g., advanced re-identification logic.

Operation example

Fig. 3.25 shows a frame of the stream captured by the camera, also implementing the MediaPipe's hand detection. In the figure, the human operator has just grasped an object with the right hand, i.e., the contact stage of the pick action. It can be clearly seen how the hand completely covers the object to the camera's view, so YOLO cannot in any way identify the object from this frame, just as a human being could not. Usually, the recognition of the picking action occurs after the contact, thus in a frame between this and subsequent ones, in which the object is moved by the hand and continues to be hidden from the camera's view. Without an object tracker, the association of an object with the recognized action is impossible.

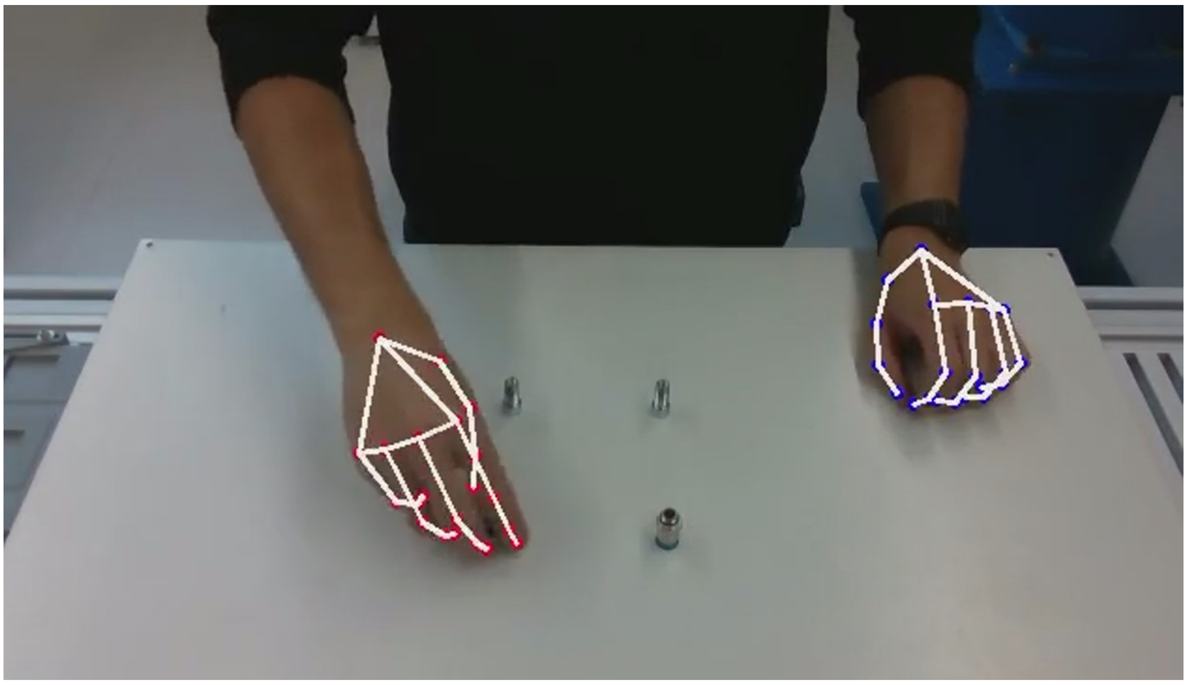
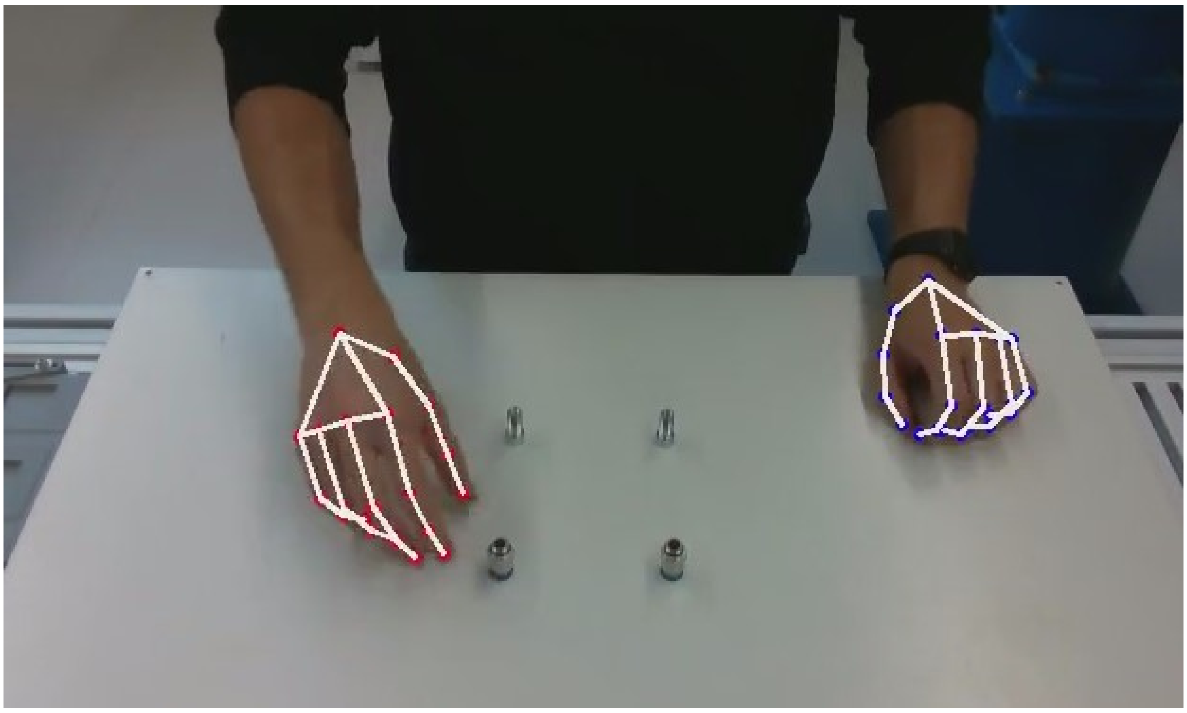
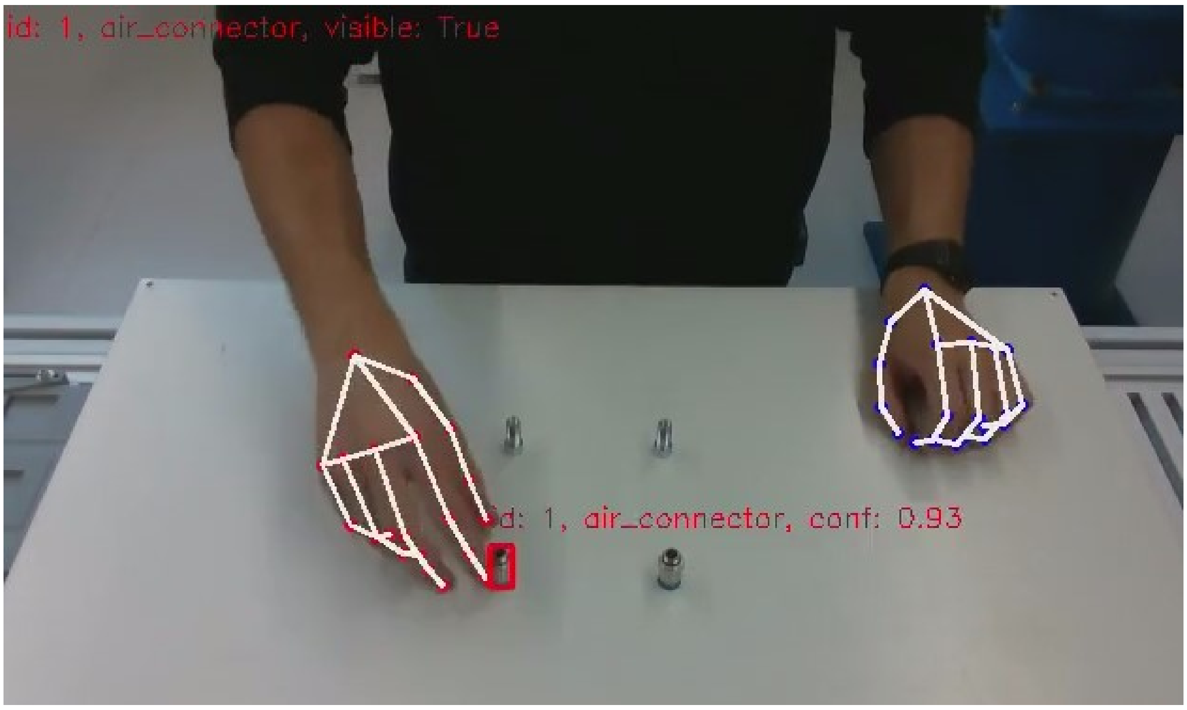


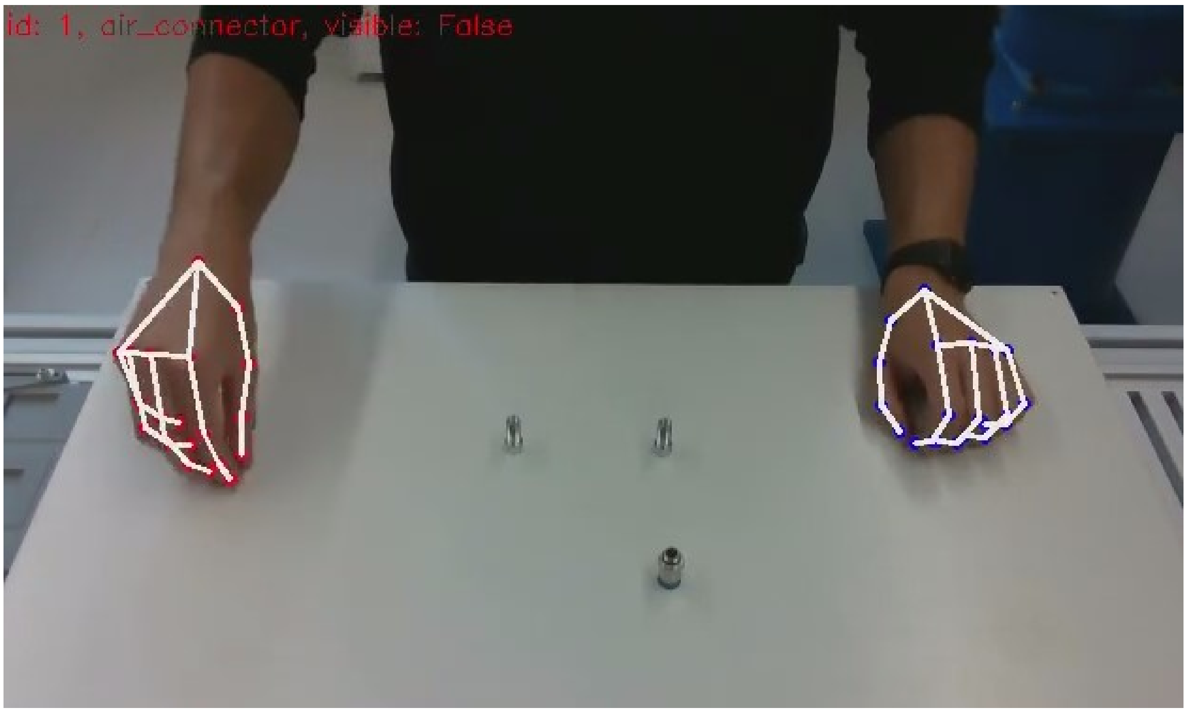
Figure 3.25: Contact stage of pick action without the object tracker

Refer now to Fig. 3.26. It shows three non-consecutive frames extracted from the real-time implementation of the developed object tracker to the same scenario just described. They represent the three stages of pre-contact, contact, and post-contact for a pick action. More precisely, the second frame, related to the contact, actually shows a frame immediately preceding the contact, in which the object is still visible to the camera and the tracker associates the picking hand with the object based on proximity. When the tips midpoint of a hand comes into close proximity to a tracked object, the bounding box of that object is shown with the color related to the hand, which is red for the right and blue for the left, and the association process begins. On the screen showing the images captured by the camera, the objects that the tracker considers to be held in the hand are shown in the upper left corner with their respective colors. In the upper left corner of the screen, the objects that the tracker considers to be held in the hand are displayed with their respective colors. The interesting information is not so much in the second frame, where the object is still visible and the hand is close so it is easily relatable to the action, but in the third frame, which shows a generic moment of post-contact. Again, in this frame, as the one shown previously in Fig. 3.25, the view of the object held in the hand is completely obstructed, but this time, thanks to the tracker, the information of the object held in the hand is retained due to the memory mechanism it implements. In the GitHub repository of the developed object tracker [112], it is possible to find the entire video from which these frames were extracted to fully view the operation, including the excellent re-identification of the object after release from the hand.


(a) Pre-contact


id: 1, air_connector, visible: True
id: 1, air_connector, conf: 0.93

(b) Contact (immediately before the contact)



(c) Post-contact

Figure 3.26: Pre-contact, contact, and post-contact stages for a pick action implementing the object tracker

3.4 Data acquisition

This section describes the methods used to acquire the data used for training the action recognition model. A distinct section has been dedicated because the process is very detailed and the action recognition model is central in the related work, being the basis for the proper functioning of subsequent action prediction and anticipation. Information about other data acquired, including those for training YOLO for object detection, have all been described above in the respective sections.

The two types of data that have been collected for the action recognition task are related to hands motion and objects. They have been acquired together, using the script `data_acquisition.ipynb`. The fundamental parameters of the script, which impact the behavior of the models, are the two at the beginning: the sampling frames per second (FPS), which has been set to 10, and the sampling seconds, which has been set to 1. This means that the script, for each sequence, is acquiring data for 1 second, for a total of 10 frames. Sequence data is then saved into specific folder with incremental numeric id, each with two sub-folders called "landmarks" and "objects", dedicated to landmarks and objects data respectively. Frame data, both for landmarks and objects, are saved locally as numpy array, in the easy-to-manage file format `npy`, with an incremental numeric id, in this case from 0 to 9 (10 frames per sequence in total).

3.4.1 Landmarks data acquisition

As mentioned in Sec. 3.3.1, human actions have been related to MediaPipe's hands landmarks to capture hands movement, so hands landmarks data, listed in the same section, has been processed and acquired. For each hand, 21 landmarks are provided by MediaPipe, providing 3 coordinates for each of them, as described earlier in Sec. 3.3.1. To these 3 coordinates, the depth value provided by the RealSense camera is added. It is extracted from the depth image, aligned with the RGB image, at the corresponding pixel of the landmark. To get depth based on a single pixel is not a best practice, as the depth sensor of the RealSense camera is not really reliable for the single point. In fact, the depth image contains many sparse zero values which corresponds to points in the frame not being captured by the depth sensor, especially where there are objects. The following figure highlights in green all points corresponding to a null depth (zero values).

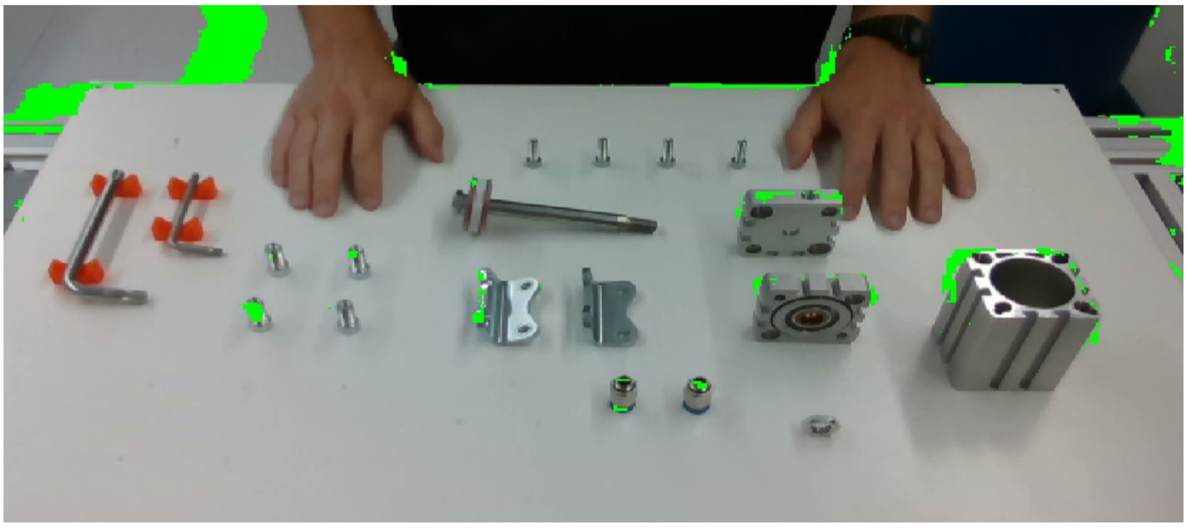


Figure 3.27: Highlighted zero depth values (in green) in the acquired image

A specific algorithm, based on a squared kernel considered around every frame pixel, has been developed to overcome this limit. This is found in the `get_depth` method of the `RealSenseCamera` class, but has to be explicitly activated by setting the boolean `apply_kernel` to `True`, otherwise the single pixel depth is provided. The `kernel_size` parameter specify the side length of the squared kernel. Then, the depth values of the kernel are extracted from the depth image, discarding the zero values. Finally, the average value of these non-zero values is returned as the depth of the central point of interest. This has been done with the aim that zero values would not affect the detected depth value.

This algorithm has proven to be very useful in reducing zero-values for depth. Despite this, the general lack of accuracy in depth measurements has made it preferable not to use the depth value in the action recognition neural network. After various tests, it was clear how it constituted a source of noise rather than useful data.

The 4 values for each landmark are then concatenated into a 1D vector, following the hand landmarks order used by MediaPipe, for a total of 84 values per hand. These vectors are in turn concatenated, first the right hand and then the left, to form a single vector that contains the information of both hands, with a total of 168 values. If one hand could not be detected in an acquired frame, MediaPipe Hands solution does not provide it in the process results. This has been properly handled to add zeros as values relative to these landmarks, so as to still have vectors of the same size. When acquiring a sequence, the 1D vector containing the 168 values of frame landmarks data is saved temporary in memory, in a list, and then saved locally in the landmarks folder of the sequence as a numpy array at the end of the acquisition. This delay has been made in order not to slow down the acquisition process with local savings, keeping the sampling frame rate more stable as possible. The model training program contains appropriate functions to extract back single landmarks information from this concatenated 1D vector and to perform the proper data manipulations.

3.4.2 Objects data acquisition

Object data has been acquired as well. To make it relevant for the action recognition, only objects held in the hands have been considered. This information is provided by the dedicated object tracker that has been developed to get hand-held objects and track them properly. More details on this complex tracker is provided in the dedicated Sec. 3.3.5. For each object held in the hand, a 1D vector is stored. It contains 6 elements:

- The label id of the object, specified during the YOLO training. Look at Sec. 3.3.4 for more information;
- The id of the hand that is holding the object: 0 for the right hand, 1 for the left hand;
- The x and y coordinates of the top-left corner (x_1, y_1) and bottom-right corner (x_2, y_2) of the bounding box, relative to the camera frame.

Even for this data, the model training program has methods to perform the appropriate data manipulations.

3.4.3 Data acquisition setup and program

The Intel RealSense D435i camera comes with a Python package, named `pyrealsense2`, that provides the C++ to Python binding required to access the SDK. Around this, a further wrapper class, named `RealSenseCamera`, has been specifically developed, to easily expose the most suitable methods for the application under consideration. In the `connect` method of the aforementioned class, the camera is configured for both RGB and depth channels, with a resolution of 640x480 pixels, at 30 FPS. The alignment of the two channels is also performed, with reference to the RGB one. This will allow to correctly obtain the depth value related to specific coordinates in the camera frame, with the method `get_depth`. Intrinsic parameters are saved into `intrinsics` parameter and they are necessary to deproject a pixel to a point, with coordinates in millimeter, using `deproject_pixel_to_point` function.

To make data acquisition easier, a set of physical buttons and a pedal have been used. These are connected to Arduino, that runs a basic program to handle the buttons and pedal press and communicates to the data acquisition program in the PC through a wired serial communication, using Universal Asynchronous Receiver-Transmitter (UART) protocol.

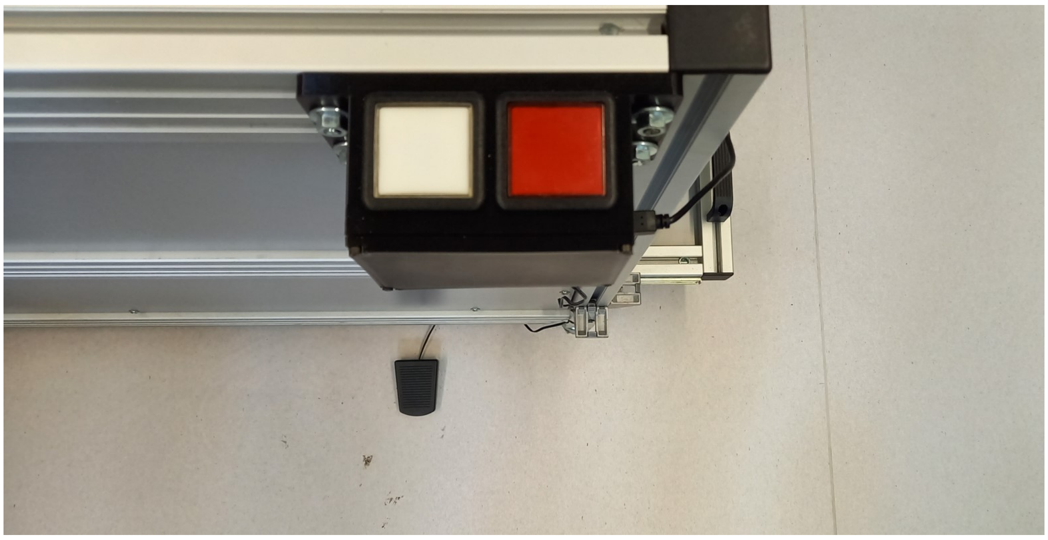


Figure 3.28: Buttons and pedal setup using Arduino

The serial communication in Python is simplified by the use of `pySerial` library. The acquisition program is set to be just a receiver. The listening loop has been run in a separate threat, that execute the function `arduino_connection_thread`, as it requires to continuously read the incoming line of the communication in a dedicated loop. In this way, the main loop that acquires frames from the camera can run independently, just checking for the boolean `acquiring`, of the specifically created `Recorder` class, to start each acquisition. The functions associated with the buttons and the pedal are:

- White button: after one second, clears the landmark and object acquisition lists to ensure a correct new acquisition and starts recording. When the recording starts, the program emit a quick sound to warn the operator that is performing the actions to be sampled. The recording automatically terminates after the set period, in this case 1 second, which corresponds to 10 acquired frames, and the local saving of the elements in the lists occurs automatically. In the camera loop, there is a check for for the time elapsed between the acquisition of one frame and the next. It skips frames if these are sampled faster than specified, and ends the current acquisition with an error if the frames are being processed too slowly to comply the set sampling FPS. After each successfully terminated acquisition, the acquired data are reproduced over a black frame, This is for the human to check if the sampled sequence is good and every frame has been captured correctly. The numeric id of the sampled sequences is automatically increased;

- Red button: returns back to the previous numeric id. Running again a registration, with the white button, will overwrite the corresponding acquisition. This is useful when, after consulting the video of the data acquired over the black frame, some errors are noticed, linked to system acquisition errors or human errors in executing the action;

- Pedal: terminates the acquisition program. When restarting, the last numeric id is automatically restored and the sampling process can continue from where it was left off.

As mentioned, when an acquisition successfully terminate, a representation of the acquired data over a blank frame is played as a video, using the same sampling FPS, through the `play_video_from_sequence_data` function.

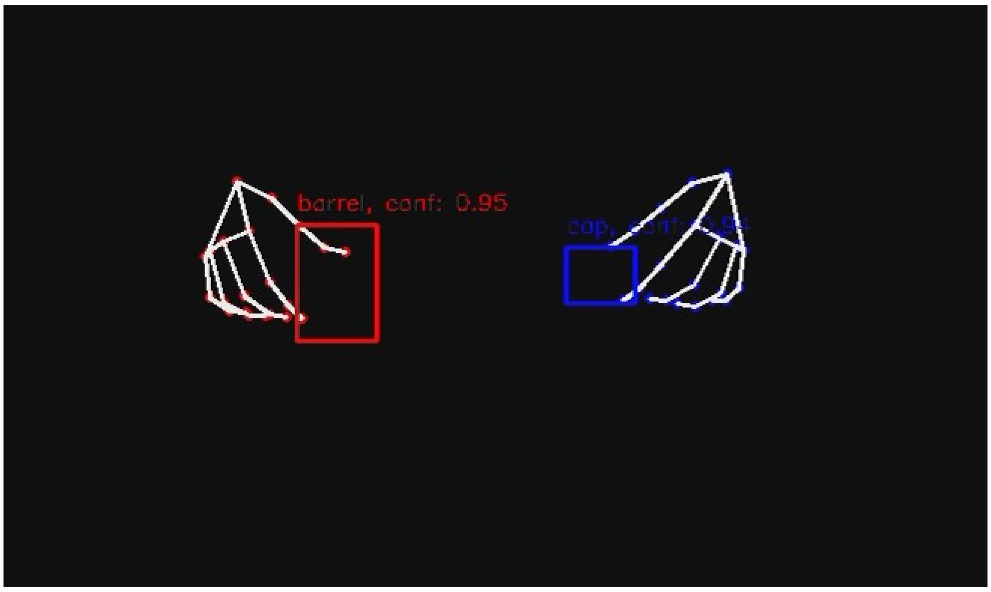


Figure 3.29: Acquired data visualization on blank frame

It is worth noting that this visual representation is in no way fed into the neural network as an image. Indeed, the neural network only processes landmarks data as values. It is just a representation of the registered data which can be easily visualized by the human to check for the correctness. This is also an interesting way to compare system recognition skills to human ones, based on the same available data. For example, in Fig. 3.30 it is showed an acquired frame data representation on blank frame compared with the actual frame seen by the camera. In the sequence from where this frame has been extracted, the human is performing a screwing action, in particular with a wrench. By looking at the RGB image, it is relatively easy for a human to correctly classify the action being executed, even if it is just one frame and not the full video sequence. In fact, the wrench which can be clearly seen in the hand suggests the screwing with wrench action. Recall that only landmarks and objects data have been acquired to train the action recognition model, extracting them from the acquisition context. The reasons for this decision have been discussed extensively in Sec. 3.3.1. The representation over a blank frame shows how the difficult in recognizing the action rapidly increase if only the acquired frame data is available. All the context related to the image is actually discarded. In addition, the data is subject to the limits of the models used for process hands tracking and objects detection, MediaPipe and YOLO. For example, hand landmarks can be distorted if a large object is held in the hand, not allowing for a clean view of it, or the hand could even stop to be tracked for some frames. Furthermore, YOLO struggles to detect objects when hands interact with them, covering some portions, despite efforts to include many conditions in which objects were held in the hand in the training dataset. The developed object tracker, described in detail in Sec. 3.3.5, comes to help trying to ease this problem.

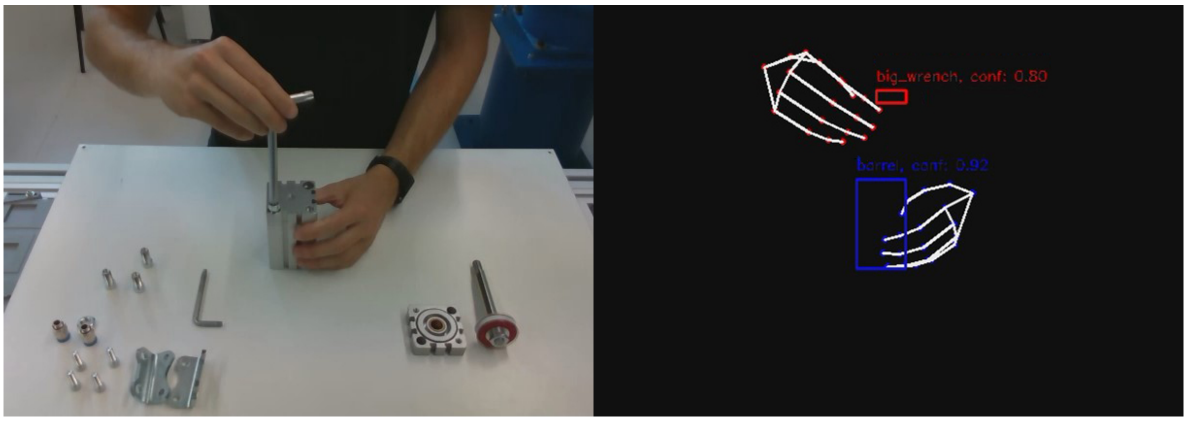


Figure 3.30: Comparison between camera view (left) and frame data actually acquired (right)

Chapter 4

Results

In this chapter, evaluations on all investigated models for action recognition and action prediction are presented. In addition, performance metrics obtained from YOLO fine-tuning on the constructed application-specific object dataset will be shown. Finally, the overall results obtained from the real-time implementation of the system will be presented, including the evaluation of the developed object tracker. The evaluation of the real-time implementation remains more difficult to quantify numerically, but an attempt has been made to report values that are as true and repeatable as possible.

4.1 Action recognition

Action recognition models have been evaluated using several performance metrics, computed on both the validation and test sets to provide an unbiased assessment of each model's performance.

The performance metrics considered are the following:

- Accuracy (Top-1): the proportion of correct predictions out of all predictions made. Only the conventional Top-1 accuracy is considered;
- Macro-averaged Precision: the ratio of correctly predicted positive observations to the total predicted positives. It indicates how many of the positive predictions made by the model are actually correct, i.e., the quality of positive predictions;
- Macro-averaged Recall (Sensitivity): the ratio of correctly predicted positive observations to all instances of the actual class. It measures the model's ability to identify all relevant instances in the dataset;
- Macro-averaged F1-score: the harmonic mean of precision and recall, providing a single metric that balances both. This metric is very common in the results presented in the studies;

- Macro-averaged ROC-AUC (Area Under the Receiver Operating Characteristic Curve): an aggregate measure of performance across all possible classification thresholds.

The macro-average applied to precision, recall, F1-score, and ROC-AUC calculates each class's performance metric and then performs the arithmetic mean across all classes. This gives equal weight to each class, regardless of the number of instances.

Because the dataset built to train the action recognition model is perfectly balanced, the accuracy metric alone could be a reliable indicator of model performance, as the risk of accuracy being inflated by a majority class is minimized. Nevertheless, precision, recall and F1-score have also been evaluated to provide a deeper understanding of the model's performance.

The list of investigated models, including detailed description, is provided in Sec. 3.3.1. A list of their names is reported here:

- Simple LSTM;
- Complex LSTM;
- Bi-directional LSTM (Bi-LSTM);
- Convolutional 1D (Conv1D);
- LSTM with object (LSTM-Objects).

The LSTM model that includes object (LSTM-Objects) data as input has been also included to allow direct performance comparison with other models that do not use it. Further consideration on this model will be carried out later.

The parameters used to train all action recognition models, including the optimizer, learning rate scheduler, and early stopping mechanism, were discussed in Sec. 3.3.1 and are summarized here:

- Batch size: 16;
- Epochs: 100;
- Optimizer: Adam;
- Initial learning rate: 0.001;
- Learning rate scheduler: multi-step learning rate scheduler;

- Learning rate scheduler gamma: 0.1;
- Learning rate scheduler milestone epochs: 40, 80;
- Early stopping enabled based on validation loss with best weight restore;
- Patience: 15;
- Minimum delta improvement in validation loss: 0.001.

It is anticipated that the results obtained for all models are generally excellent. It is recalled, however, that they have to be considered as limited to the specific workstation of the related work, described in Sec. 3.2, since the actions dataset has been constructed based on frame data sequences acquired by the camera in the specific setup of the current application. For this reason, these results should not be considered repeatable with the same high performance in a different workstation, although this could certainly be achieved by first expanding the actions dataset and re-training the models with it.

4.1.1 Single-handed models

Models that take into account single-handed data, both for movement and objects when specified, will be shown first. In particular, the right hand data will be taken into consideration, that is the hand with which all actions in the dataset have been performed. Landmark depth data is not considered here and will be introduced in the next section. These single-handed models showed excellent results despite the lack of the auxiliary hand information. Next, models taking both hands' data into account will be shown, in order to compare the results.

The performance of each single-handed model on the validation and test sets is summarized in Table 4.1 and Table 4.2. These metrics provide a comprehensive overview of how well each model generalizes to unseen data. As previously described, the size of the validation set and test set have been taken equal to 10 percent of the total number of samples, which in the case of the dataset used to train the action recognition model corresponds to 100 samples each. The training, validation, and test sets used to evaluate performance metrics and to generate all the graphs shown below are the same for all models.

Metric	**Sim LSTM**	**Com LSTM**	**Bi-LSTM**	**Conv1D**	**LSTM-Objs**
Accuracy	0.9400	0.9200	0.9400	0.9000	0.9700
Precision	0.9482	0.9159	0.9387	0.9001	0.9724
Recall	0.9355	0.9143	0.9395	0.9024	0.9698
F1-Score	0.9396	0.9139	0.9390	0.9000	0.9709
ROC-AUC	0.9954	0.9898	0.9959	0.9864	0.9989

Table 4.1: Performance metrics comparison on validation set (single-handed models - right hand)

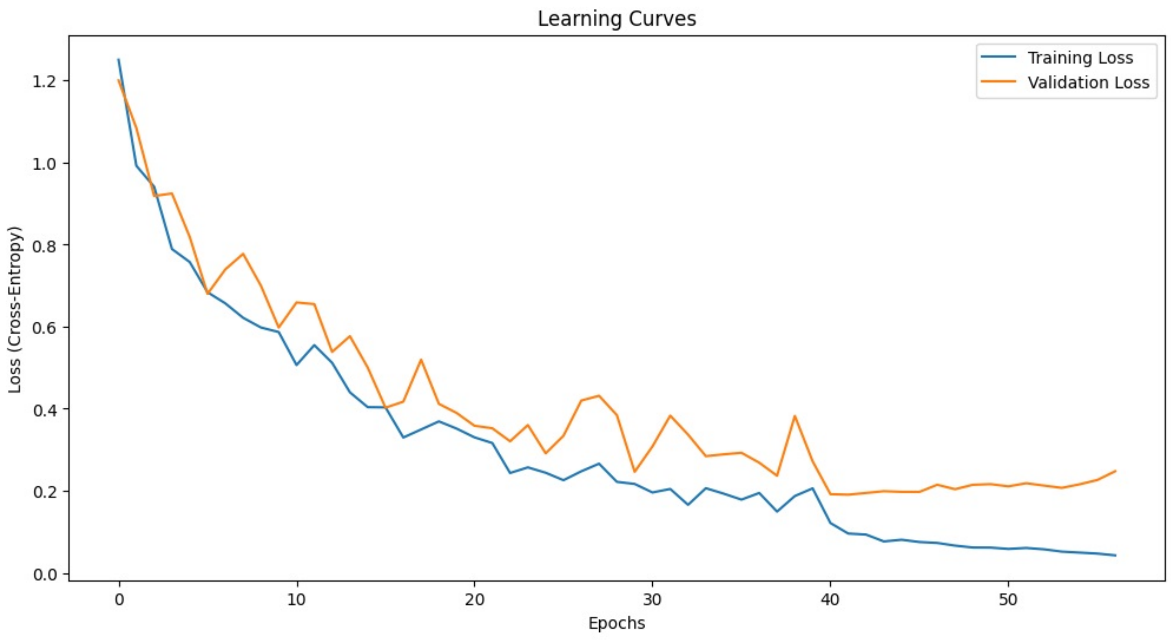
Metric	**Sim LSTM**	**Com LSTM**	**Bi-LSTM**	**Conv1D**	**LSTM-Objs**
Accuracy	0.9500	0.9200	0.9500	0.9100	0.9700
Precision	0.9550	0.9291	0.9516	0.9174	0.9711
Recall	0.9513	0.9226	0.9492	0.9124	0.9717
F1-Score	0.9519	0.9240	0.9499	0.9143	0.9707
ROC-AUC	0.9971	0.9863	0.9947	0.9854	0.9980

Table 4.2: Performance metrics comparison on test set (single-handed models - right hand)

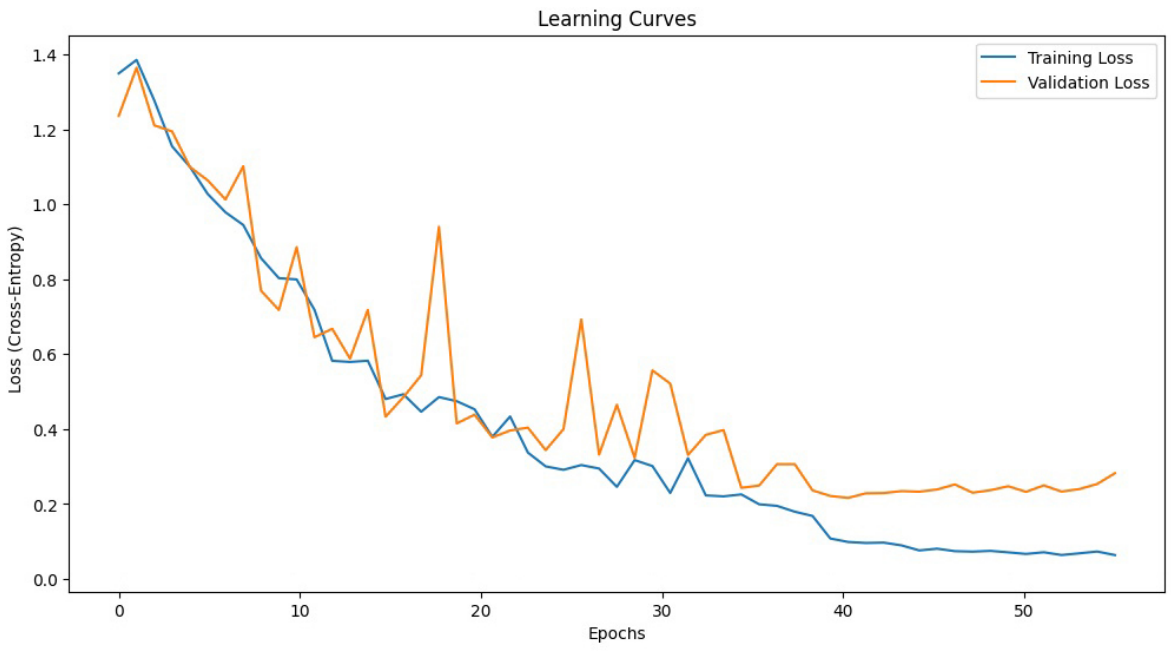
As can be seen, the accuracy results are excellent for all models, being always equal to or greater than 90%, especially for those that implement an LSTM network. LSTM networks confirm their great ability to capture temporal dependencies, behaving robustly in classifying human actions. The CNN model, while slightly less effective, still shows very good performance, which is particularly appreciable considering the much smaller number of parameters compared to models with LSTM networks, resulting in a more rapid real-time implementation. Another consideration that can be made is that the LSTM model that includes object data (LSTM-Objects) performs better than those without these data, as expected, but the performance gap is minimal. This will be explored further later by analyzing the confusion matrices.

Precision, recall, F1-score metrics are consistent with accuracy for all the models, demonstrating well-rounded performance and errors evenly distributed between false positives and false negatives. This is in line with the expected optimal behavior due to the perfectly balanced dataset. The ROC-AUC metric confirms the ability of each model to discriminate well between different classes. Finally, having obtained similar performance metrics related to validation and test sets indicates that the models achieve a consistent level of generalization on unseen data, suggesting that they have learned the underlying patterns in the training data without overfitting or underfitting.

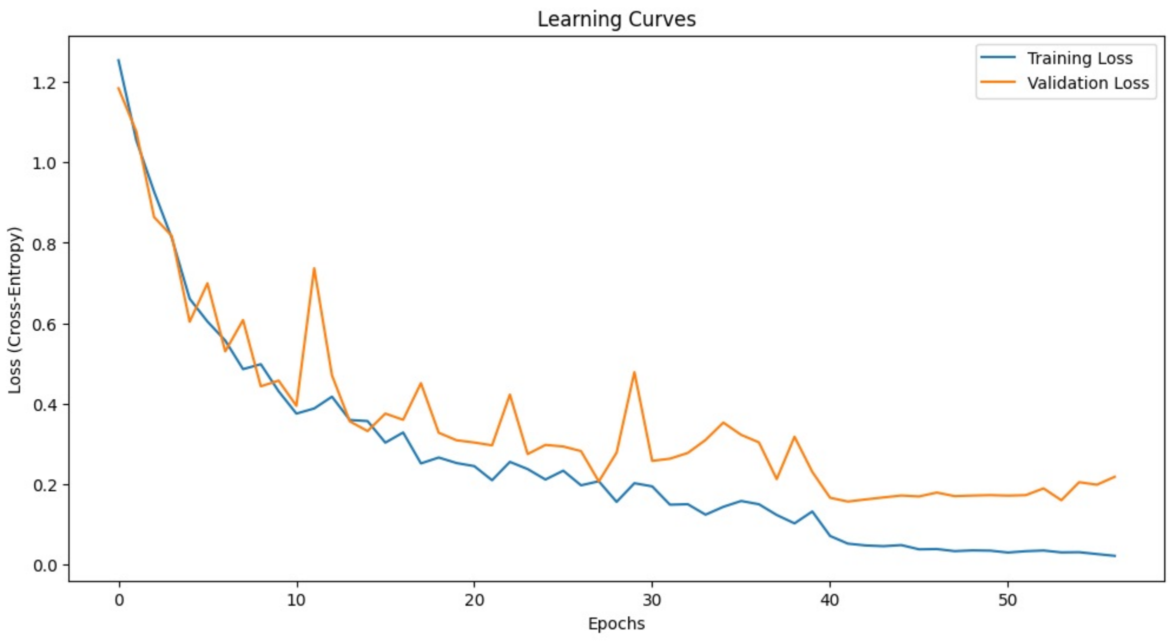
The learning curves for training and validation loss (cross-entropy loss) over the epochs provide insight into each model's learning process. They help in diagnosing issues such as overfitting or underfitting. Fig. 4.1 shows the learning curves for all the analyzed models.



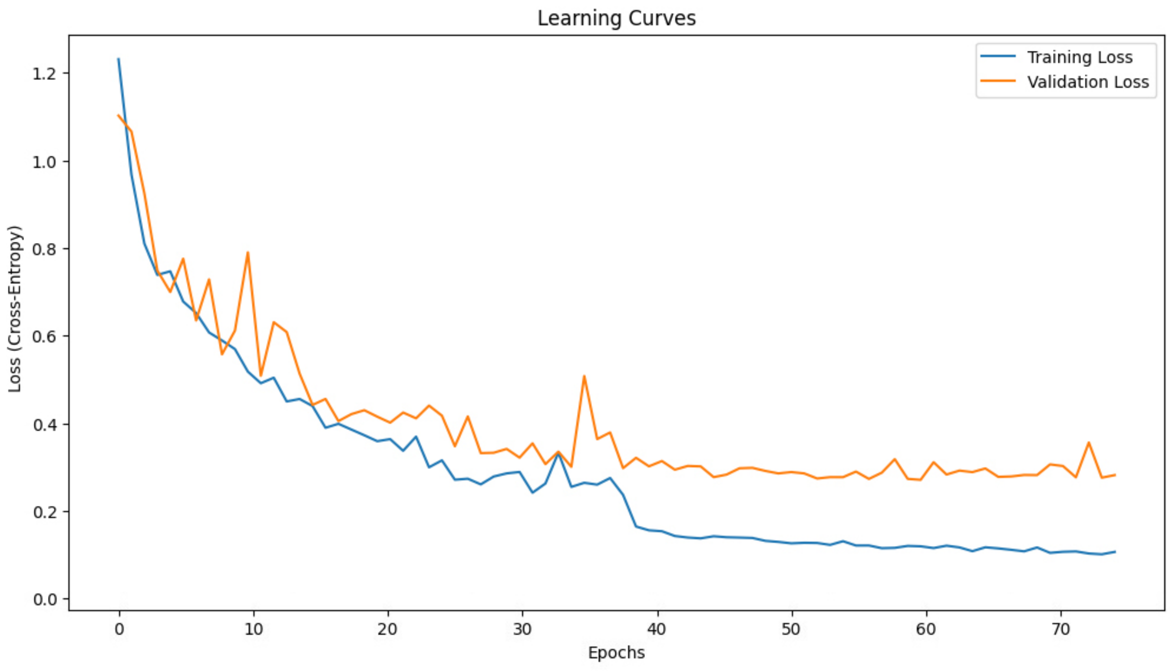
(a) Simple LSTM model



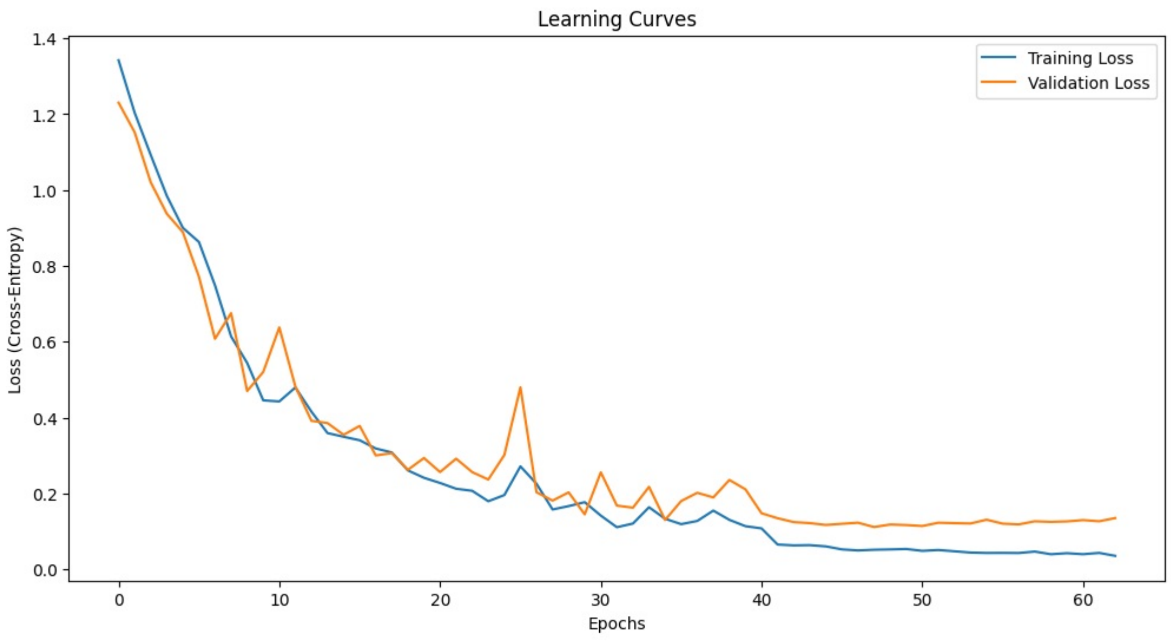
(b) Complex LSTM model



(c) Bi-LSTM model



(d) Conv1D model



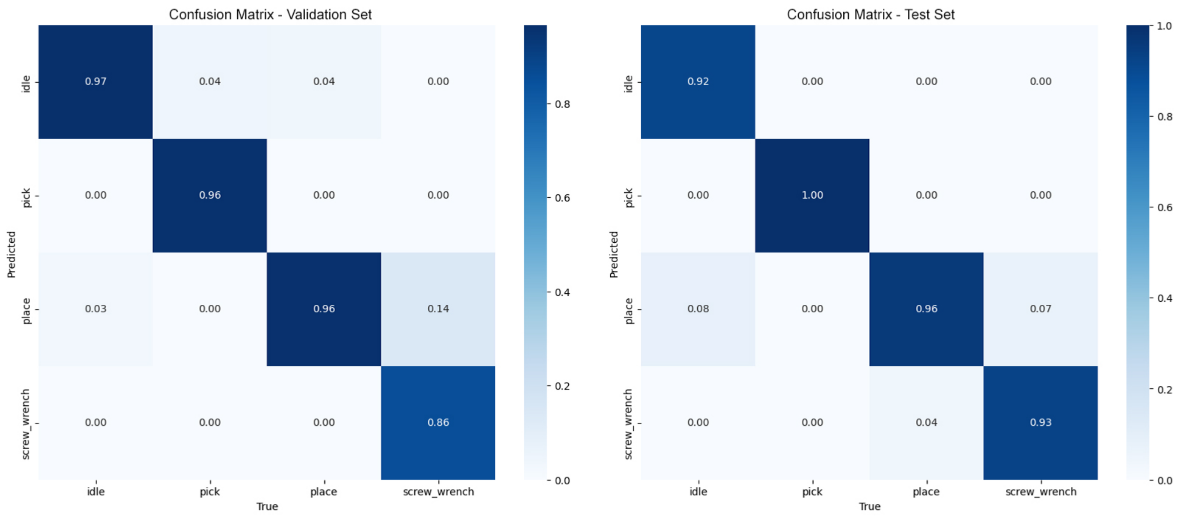
(e) LSTM-Objects model

Figure 4.1: Learning curves for the analyzed models: Simple LSTM, Complex LSTM, Bi-LSTM, Conv1D, LSTM-Objects (single-handed models - right hand)

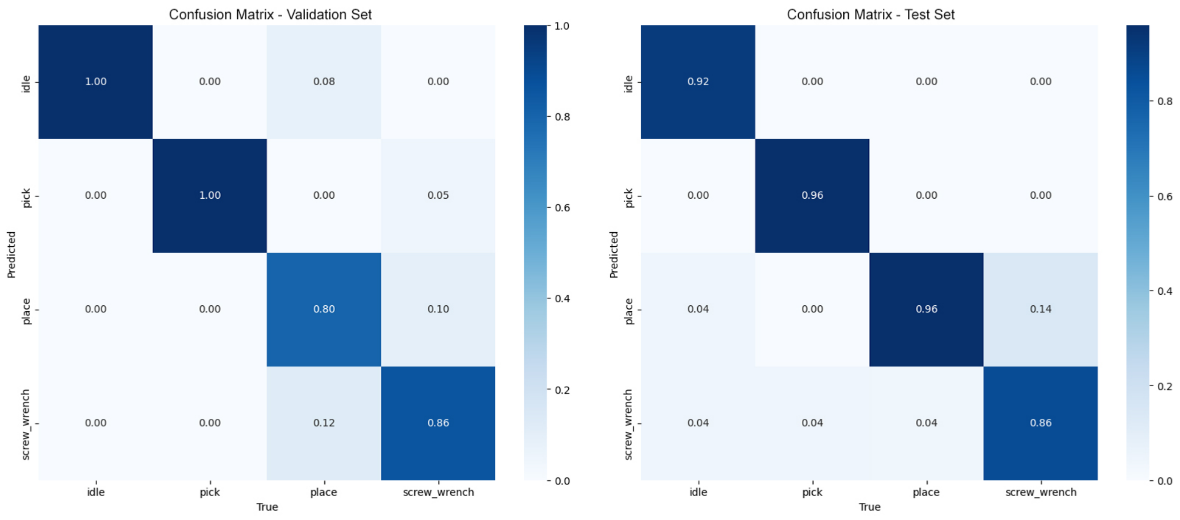
The learning curves show a good behavior for all of the models, with both loss functions gradually dropping for both the train and validation sets. It can be clearly seen in all the learning curves the effect of reducing the learning rate at the milestone in epoch 40, performed by the multi-step learning rate scheduler. This milestone has been chosen because the learning curves tend to stabilize at constant values around epoch 40. By reducing the learning rate by an order of magnitude, further slight improvements are sought. Then, the early stopping mechanism stops the training after 15 epochs of patience with no improvement in validation loss, restoring the model weights of the best epoch, which is for all models around epoch 40. The second milestone at epoch 80 is never actually reached. In fact, this has only been introduced as an aid to force convergence in early stopping, since the learning rate after this milestone is $1e-5$, which is very ineffective on model learning.

The model that performs slightly worse in terms of loss is the convolutional 1D network (Conv1D), confirming what has been observed from performance metrics. It is worth noting that the Complex LSTM network performs slightly worse than the Simple LSTM network, which shows that the structure of the former is overly complex to learn data patterns, resulting in a less effective train. This again confirms what has been obtained for the performance metrics.

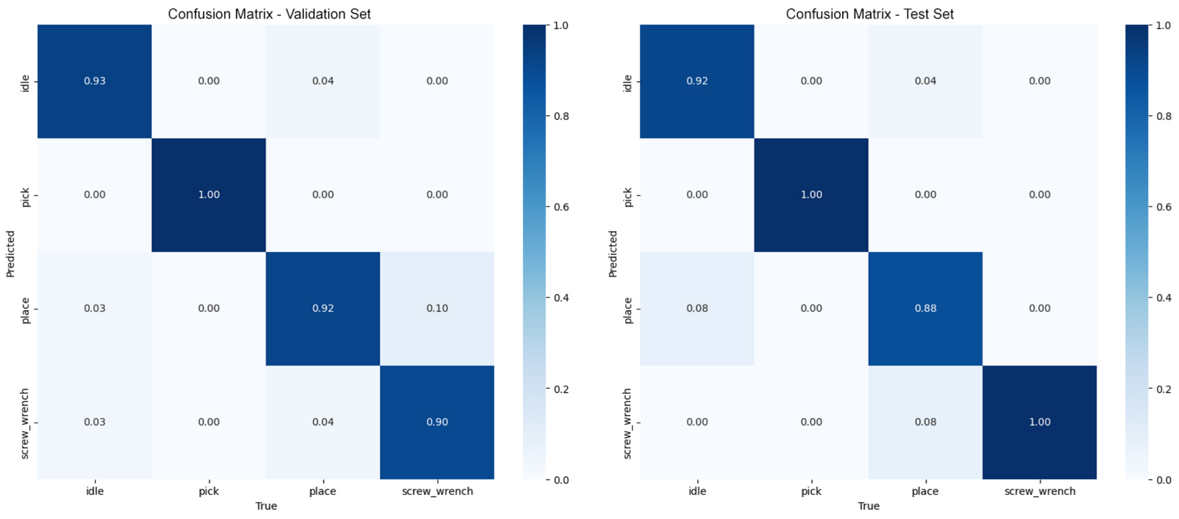
The confusion matrices provide a detailed breakdown of the classification performance across different classes. Fig. 4.2 represents the confusion matrices for the analyzed models, both on the validation set (left) and on the test set (right). These are reported in normalized form to allow for a more meaningful reading. In addition, the data visualization library `seaborn` has been used to display the confusion matrices in the same style as that produced by YOLO training, which is particularly appreciable.



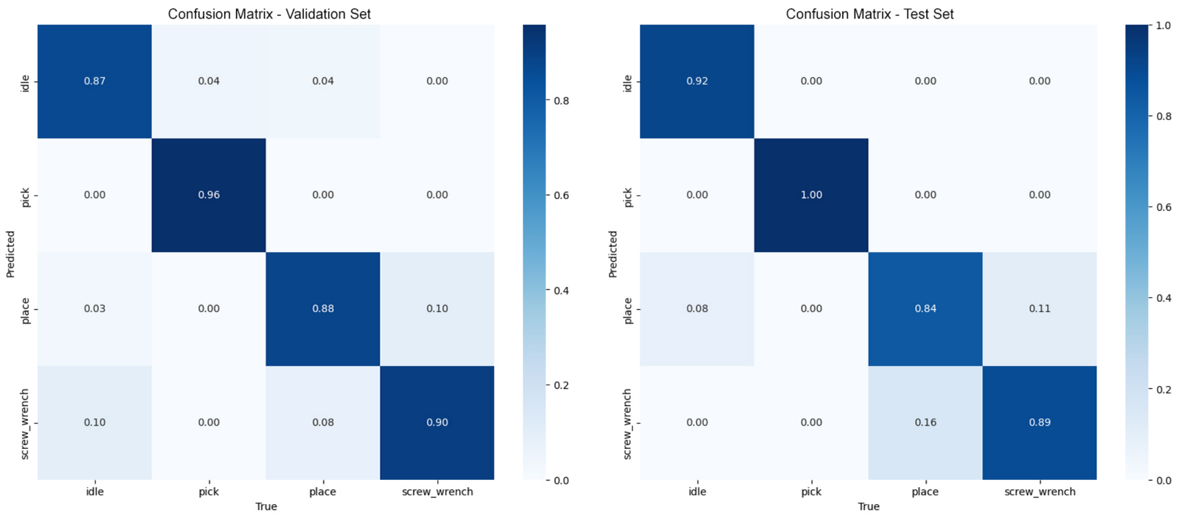
(a) Simple LSTM model



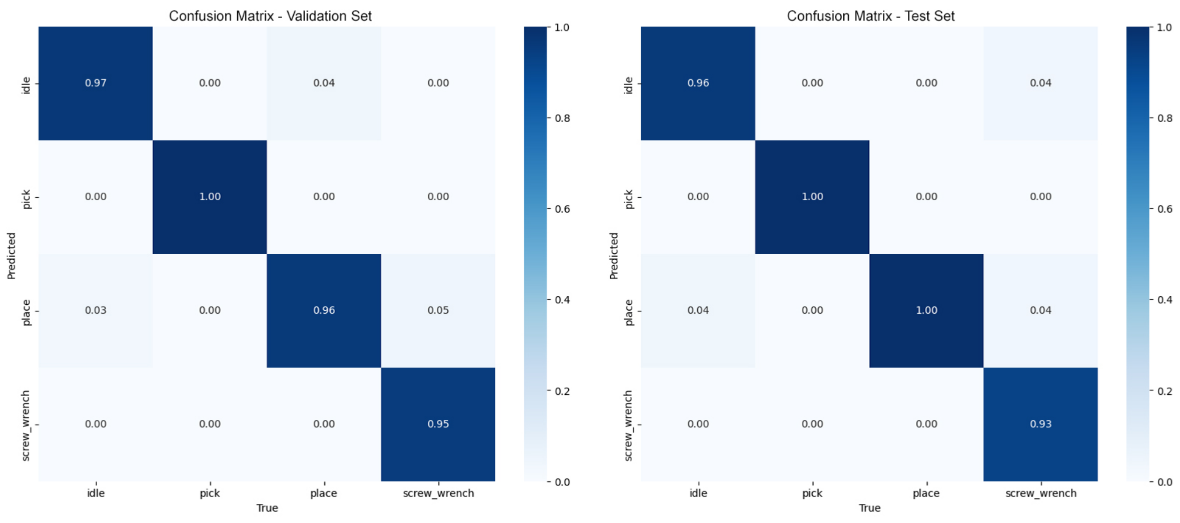
(b) Complex LSTM model



(c) Bi-LSTM model



(d) Conv1D model



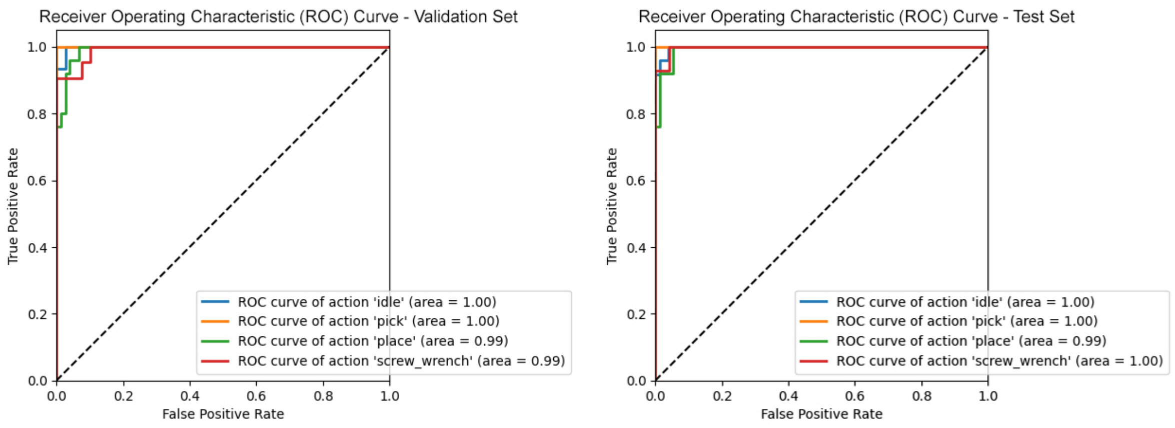
(e) LSTM-Objects model

Figure 4.2: Confusion matrices for the analyzed models on the validation set (left) and on the test set (right): Simple LSTM, Complex LSTM, Bi-LSTM, Conv1D, LSTM-Objects (single-handed models - right hand)

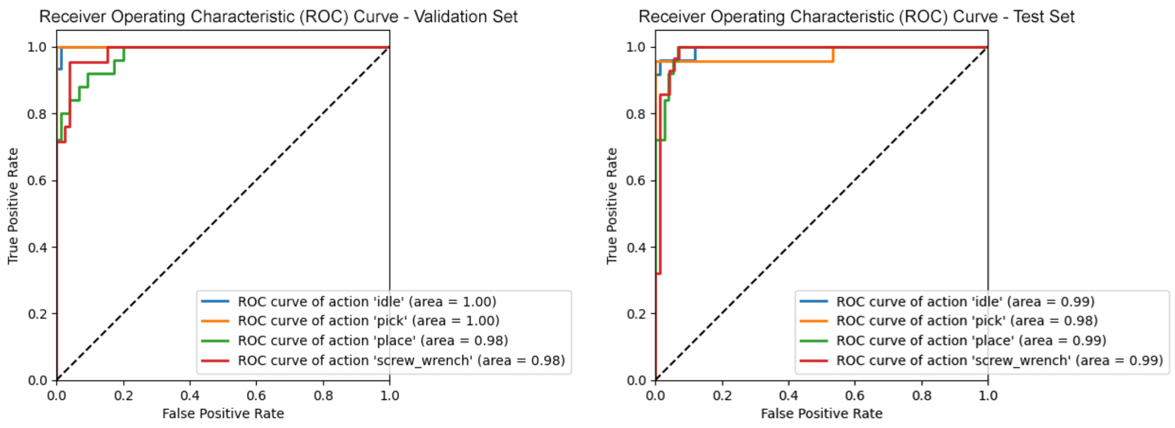
From the confusion matrices, it can be seen that the overall performance is excellent, and the similarity of the results obtained with validation and test sets confirms the consistency of the models' generalization capabilities. The pick action is the one best recognized by all models. The other three actions are confused more often. In particular, the idle and screw_wrench actions are mostly confused with the place action, but almost never with each other. This is understandable, since these may be similar to some specific place actions, but are quite different from each other. In particular, idle actions are likely to be confused with weakly dynamic place actions, which are obviously present in the training dataset, e.g., for object insertions that require little hand movement, while screw_wrench actions are confused with place actions in which the hand is particularly closed and moves by slightly rotating around itself.

The LSTM-Objects model gives the best results, although the difference with the Simple LSTM and Bi-LSTM models is small. As expected, this model manages to distinguish better between place and screw_wrench actions, since the two wrenches are only used for the latter. Perfect recognition is not achievable because, as previously mentioned, the actions dataset also includes actions without objects in the hand, in order to train the model to recognize actions even if the object in the hand is not detected in the real-time application. In these cases, the benefit of increased accuracy in the classification due to the object data is lost, so the recognition is based only on the hand motion data.

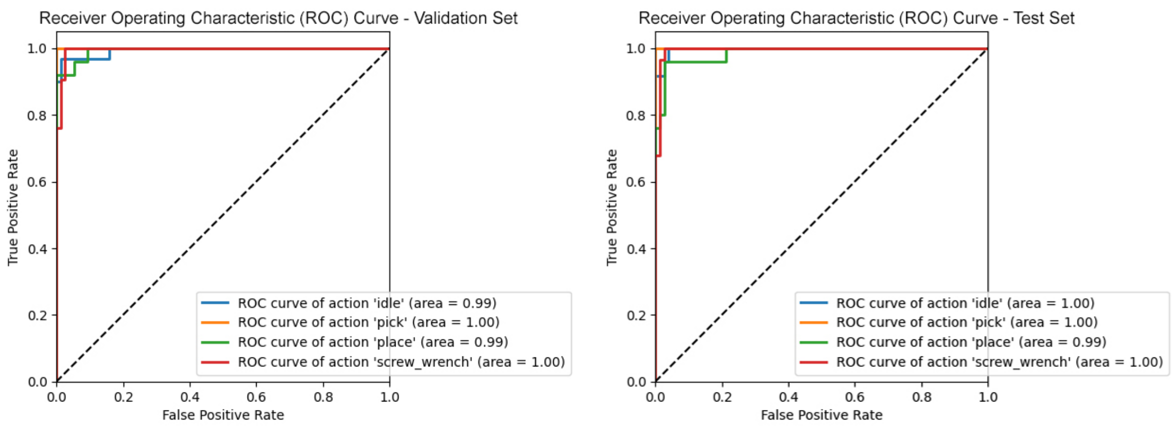
The ROC curves (Receiver Operating Characteristic curves) for each model are illustrated in Fig. 4.3, both for the validation set (left) and the test set (right). These curves show the trade-off between true positive rate and false positive rate at various threshold settings for each class. Since ROC curves are traditionally used for binary classification, the One-vs-Rest (OvR) approach has been used to extend them to the multi-class context. This method creates a separate ROC curve for each class by treating the current class as the positive class and all other classes as the negative class. The area under these curves (ROC-AUC) provides a measure of each model's ability to distinguish between the classes.



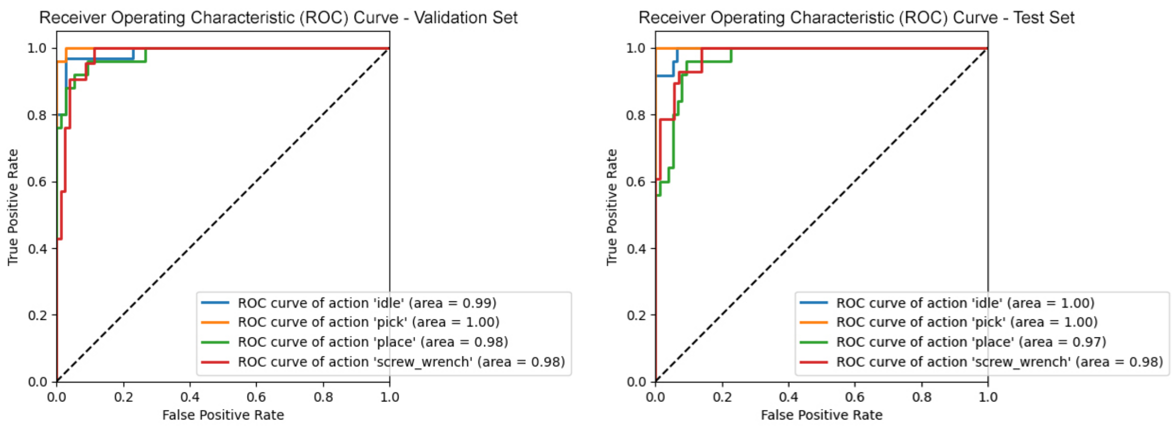
(a) Simple LSTM model



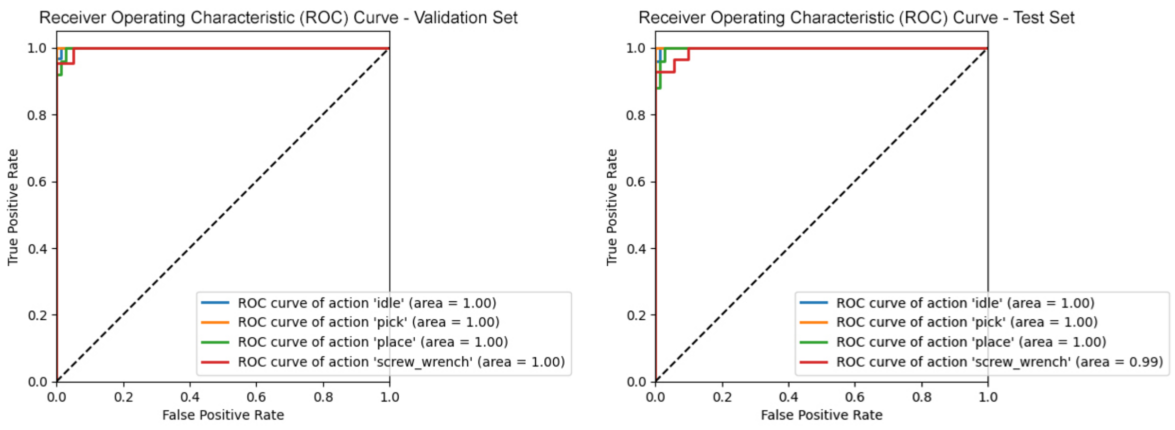
(b) Complex LSTM model



(c) Bi-LSTM model



(d) Conv1D model



(e) LSTM-Objects model

Figure 4.3: ROC curves for the analyzed models on the validation set (left) and on the test set (right): Simple LSTM, Complex LSTM, Bi-LSTM, Conv1D, LSTM-Objects (single-handed models - right hand)

ROC curves confirm the excellent performance of the models already discussed. The ROC-AUC scores are close to 1 for all actions, suggesting that the models are effective at distinguishing between the classes. A detailed analysis of the individual ROC curves and ROC-AUC scores shows the same, albeit slight, difficulties in the recognition of the idle, place, and screw_wrench actions that have been found in the confusion matrices analysis.

4.1.2 Single-handed models including depth value

Concerning the single-handed models that also take into account the depth measurement value of hand landmarks, the performance metrics will be shown for all the models without object data while only the confusion matrices and ROC curves of the two best performing models will be reported, which are the Simple LSTM and Bi-LSTM. The intent is to show how the introduction of the depth value provided by a not very accurate depth sensor represents a source of noise rather than meaningful data, slightly worsening the performance of the models. The depth value has been normalized by dividing it by the width of the frame, to give it a magnitude comparable to that of the x and y coordinates relative to the frame.

Performance metrics for the single-handed models including depth value, on both validation and test sets, are reported below:

Metric	**Simple LSTM**	**Complex LSTM**	**Bi-LSTM**	**Conv1D**
Accuracy	0.9200	0.9100	0.9400	0.9000
Precision	0.9201	0.9289	0.9388	0.8951
Recall	0.9198	0.9019	0.9414	0.8993
F1-Score	0.9177	0.9061	0.9387	0.8963
ROC-AUC	0.9922	0.9836	0.9897	0.9852

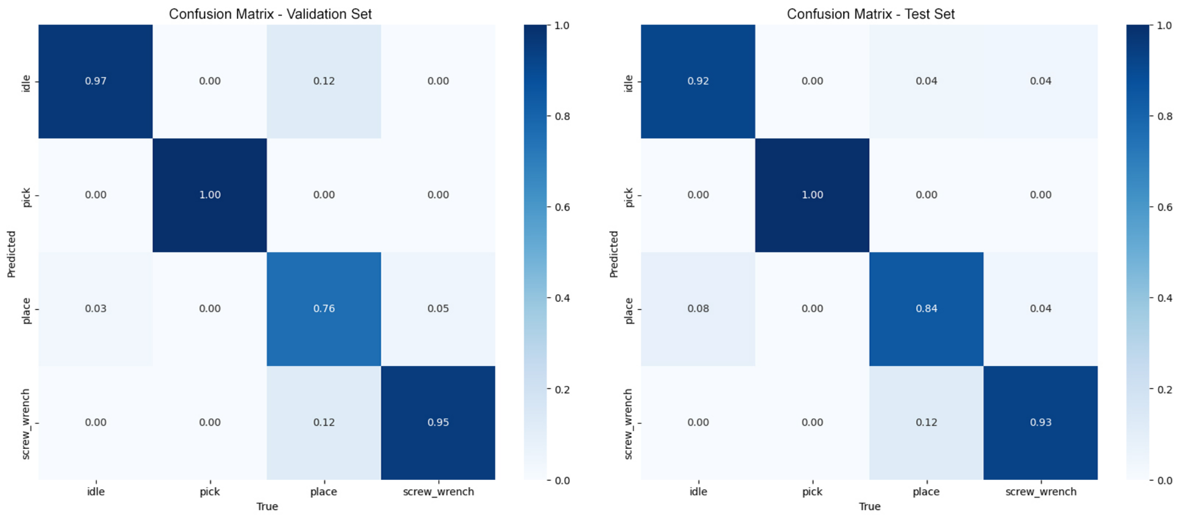
Table 4.3: Performance metrics comparison on validation set (single-handed models - right hand - including depth value)

Metric	**Simple LSTM**	**Complex LSTM**	**Bi-LSTM**	**Conv1D**
Accuracy	0.9200	0.9200	0.9200	0.9100
Precision	0.9221	0.9212	0.9331	0.9224
Recall	0.9213	0.9245	0.9245	0.9126
F1-Score	0.9215	0.9239	0.9246	0.9149
ROC-AUC	0.9934	0.9886	0.9941	0.9805

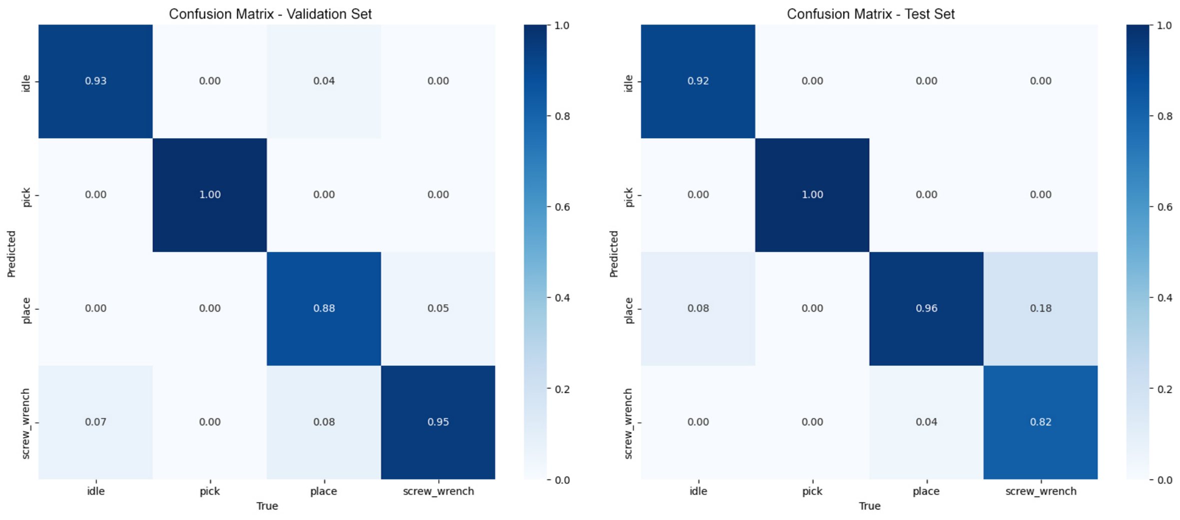
Table 4.4: Performance metrics comparison on test set (single-handed models - right hand - including depth value)

It can be seen that the performances, while still very good, are generally lower than those obtained with the corresponding single-handed models without the depth value. This means that the imprecise depth value introduces noise into the model.

The confusion matrices and ROC curves of the two best performing models, including depth data, that follow, confirm this general deterioration in performance.

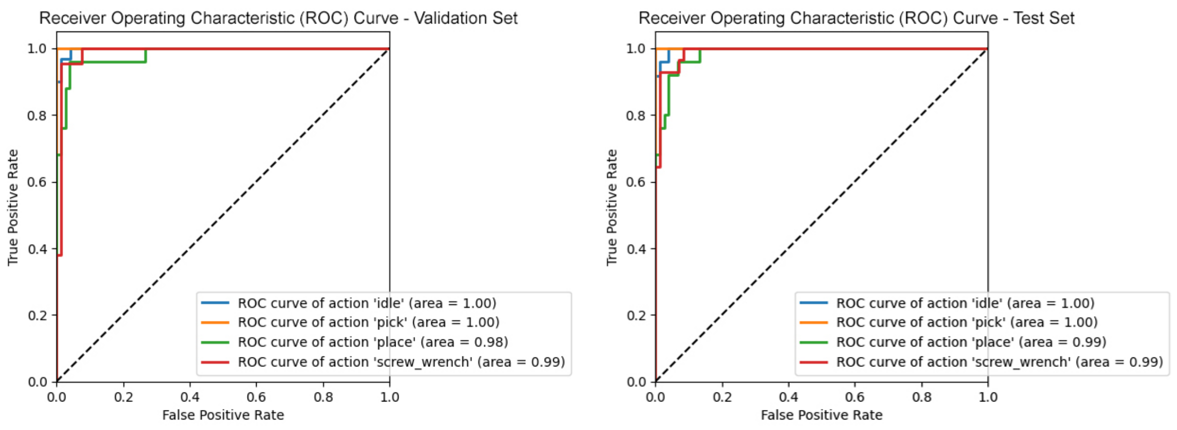


(a) Simple LSTM model

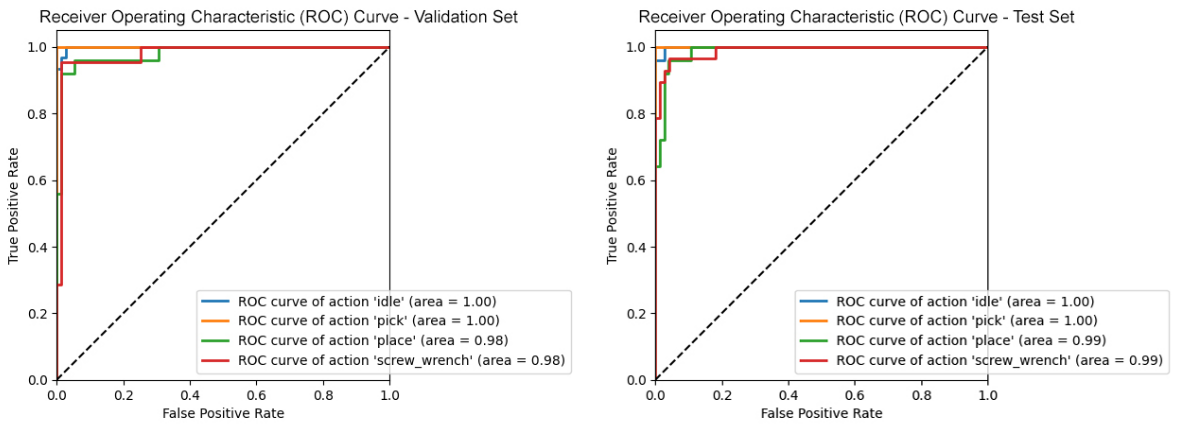


(b) Bi-LSTM model

Figure 4.4: Confusion matrices for the two best performing models on the validation set (left) and on the test set (right): Simple LSTM and Bi-LSTM (single-handed models - right hand - including depth value)



(a) Simple LSTM model



(b) Bi-LSTM model

Figure 4.5: ROC curves for the two best performing models on the validation set (left) and on the test set (right): Simple LSTM and Bi-LSTM (single-handed models - right hand - including depth value)

Comparing the confusion matrices with those in Fig. 4.2 for the same two models, it can be seen that those obtained including depth data have greater and more scattered extra-diagonal values, a symptom of noise in the input data. The same can be said when comparing the ROC curves with those in Fig. 4.3.

As a result of what has been obtained, it has been decided to exclude depth values from the action recognition models tested in real-time and from the subsequent performance investigations on the double-handed models, which will be described in the following section.

4.1.3 Double-handed models

The results obtained for models that include data of both hands are now presented, again showing the performance metrics for all the models without object data and only the confusion matrices and ROC curves of the two best performing models, the Simple LSTM and the Bi-LSTM, to keep the discussion concise. Depth values have been excluded for the considerations made above.

The training dataset contains actions performed only with the right hand as the main hand and the left hand as the auxiliary hand. This has not been mirrored to create more general double-handed models that could recognize actions performed with both hands, as the intention was to investigate whether including auxiliary hand data could improve the already excellent performance of single-handed models. For example, in a screwing action, the auxiliary hand is used to hold the component on which the screw is being tightened in a specific way. Knowing this additional information could improve the performance of the model.

Performance metrics for the double-handed models, on both validation and test sets, are reported below:

Metric	**Simple LSTM**	**Complex LSTM**	**Bi-LSTM**	**Conv1D**
Accuracy	0.9400	0.9200	0.9400	0.9000
Precision	0.9370	0.9176	0.9368	0.8954
Recall	0.9379	0.9140	0.9324	0.8960
F1-Score	0.9372	0.9151	0.9339	0.8935
ROC-AUC	0.9922	0.9932	0.9939	0.9778

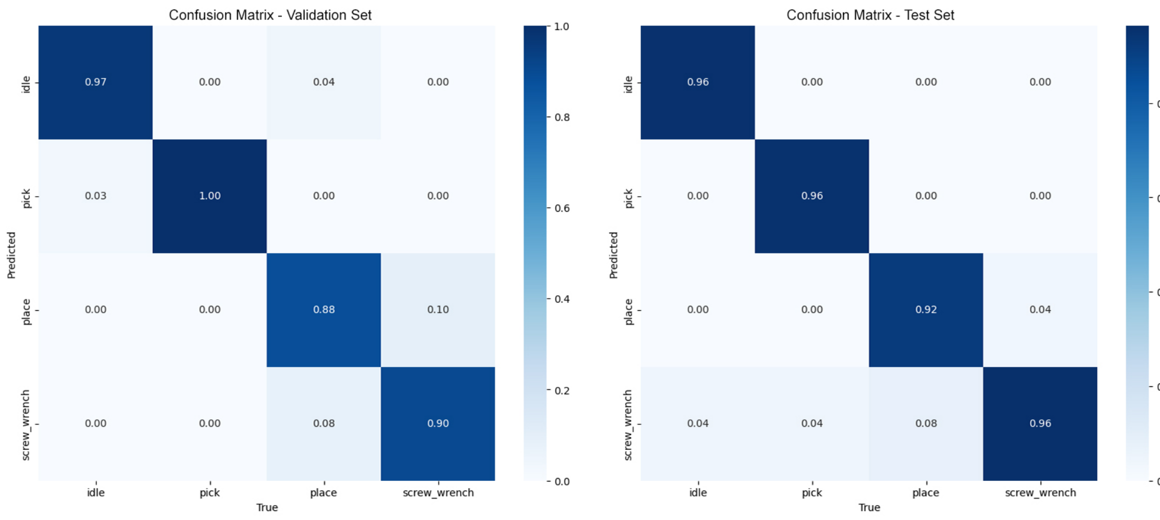
Table 4.5: Performance metrics comparison on validation set (double-handed models)

Metric	**Simple LSTM**	**Complex LSTM**	**Bi-LSTM**	**Conv1D**
Accuracy	0.9500	0.9300	0.9400	0.8600
Precision	0.9573	0.9367	0.9458	0.8667
Recall	0.9498	0.9304	0.9413	0.8643
F1-Score	0.9526	0.9328	0.9431	0.8641
ROC-AUC	0.9846	0.9847	0.9815	0.9763

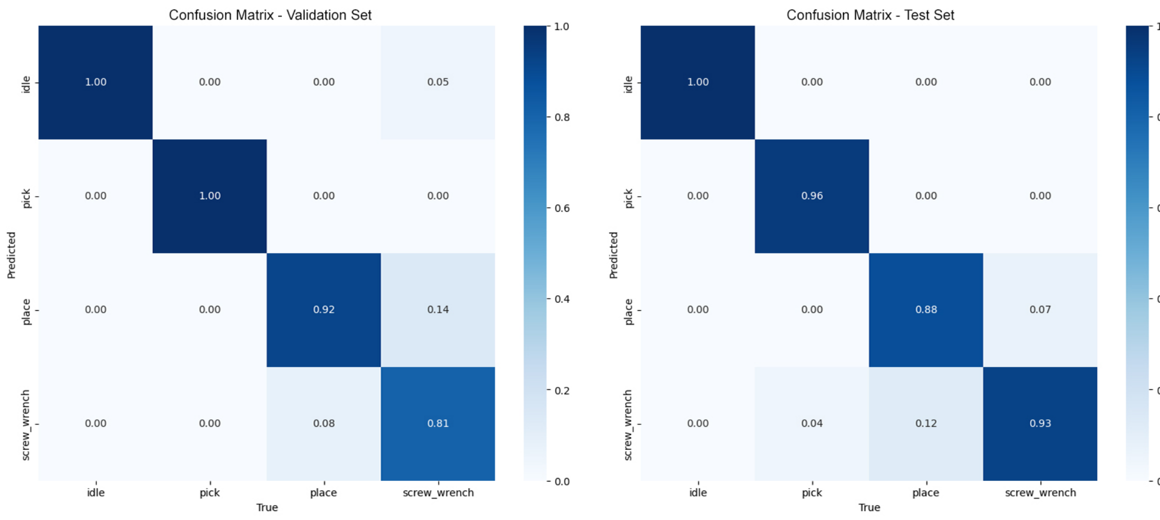
Table 4.6: Performance metrics comparison on test set (double-handed models)

Again, the performance metrics obtained for the validation and test sets are similar, indicating that also these models achieve a consistent level of generalization on unseen data. Performances are comparable with those obtained for the corresponding single-handed model for almost all models. The convolutional 1D (Conv1D) model, which already had lower performance than the others, now struggles even more to process a larger amount of data.

The confusion matrices and ROC curves of the two best performing models are reported:

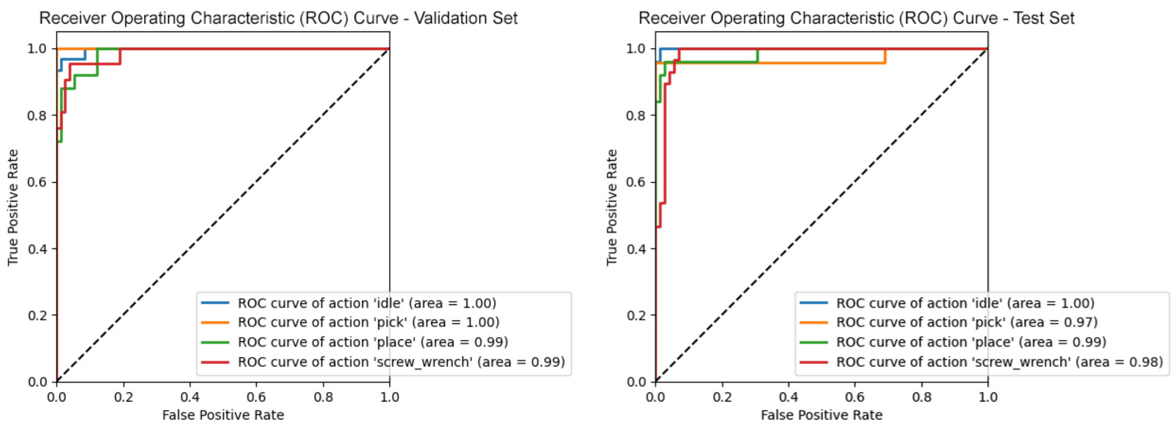


(a) Simple LSTM model

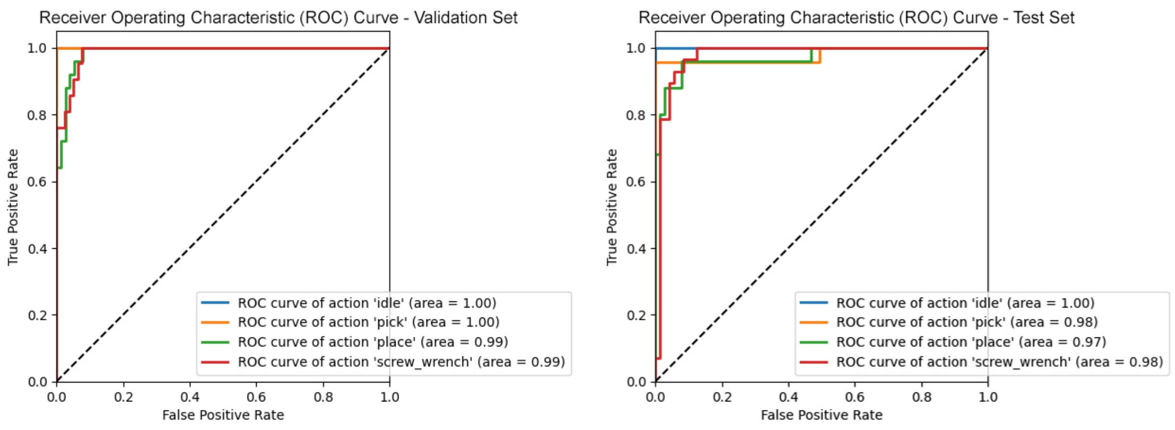


(b) Bi-LSTM model

Figure 4.6: Confusion matrices for the two best performing models on the validation set (left) and on the test set (right): Simple LSTM and Bi-LSTM (double-handed models)



(a) Simple LSTM model



(b) Bi-LSTM model

Figure 4.7: ROC curves for the two best performing models on the validation set (left) and on the test set (right): Simple LSTM and Bi-LSTM (double-handed models)

Confusion matrices and ROC curves reveal how the introduction of information about the auxiliary hand helps the models to better recognize the idle action. This can be explained by the fact that in the idle action the auxiliary hand is also in a static or quasi-static state, which increases the model's confidence in classifying this action. At the same time, it should be noted that the overall accuracy of the models is not improved, as reported in Table 4.5 and Table 4.6. In fact, confusion matrices and ROC curves show that while the true positive rate (TPR), i.e., the sensitivity, of the idle action is increased, it is decreased for the place and screw_wrench actions, which are increasingly confused with each other. This behavior can be interpreted as noise in the data introduced by the auxiliary hand data for these two actions.

These evaluations demonstrates how the set of actions under consideration can be effectively classified based on data from the hand performing the action alone. This is a very important result that leaves room for future implementations of two-independent-handed models, both for action recognition and prediction. As already mentioned, this complex system has not been investigated in the present study, as it has been considered outside the already sufficiently extended scope.

For this reason, and in light of the minimal performance improvement brought by adding object data at the cost of reduced flexibility, the real-time tests involve only the single-handed models without object data and depth values. Specifically, the two models with the best performance, i.e., the Simple LSTM and the Bi-LSTM, have been evaluated in real-time. After several tests, the Simple LSTM model has been chosen for more in-depth testing due to its excellent performance, which is fully comparable to the Bi-LSTM model, but with greater execution efficiency as it is lighter.

4.2 Action prediction

In this section, the results of the trained multi-task learning (MTL) action prediction model are listed. First, the evaluations of the model obtained by training on the dataset generated from variations of the hand-built reference sequence, named "hand-built sequence variations dataset", will be analyzed, then, they will be compared with those obtained by training on the dataset of real observed sequences, named "real sequences dataset".

The benchmark performance metrics are the same as those used to evaluate the action recognition models. These have been evaluated on both the validation and test sets, which give consistent results confirming a good level of generalization of the model, but only the results on the test set are presented to keep the discussion brief. The results obtained for the prediction of actions and objects will be distinguished, in order to thoroughly verify the capabilities of the model on both tasks on which it has been trained. Furthermore, the confusion matrices and ROC curves obtained will also be shown. Finally, qualitative results of action anticipation will be discussed, which represents the robot's application of human action predictions by leveraging the implemented decision making logic.

The MTL model developed for the action prediction is based on the sequences of action-object (held in the right hand) pairs in the training dataset, with twice the importance weight for actions than for objects, as they are central to defining the assembly sequence. For this reason, the expected performance of the model in predicting actions is higher than the performance in predicting the correct object associated with the action, also because the objects are more than the actions.

4.2.1 Model trained on hand-built sequence variations dataset

Table 4.7 shows the performance metrics evaluated on the test set for the model trained on the dataset generated from variations of the hand-built reference sequence, divided into those related to actions and those related to objects.

Metric	Actions	Objects
Accuracy	0.8132	0.8022
Precision	0.8685	0.8553
Recall	0.7678	0.8062
F1-Score	0.7442	0.8069
ROC-AUC	0.8468	0.9390

Table 4.7: Performance metrics for the MTL action prediction model on test set (trained on the hand-built sequence variations dataset)

The model achieved good performance in both tasks and the prediction accuracy of objects is comparable to that of actions.

Fig. 4.8 shows the learning curves of the model, reporting training and validation total loss, which is obtained as the weighted sum of the individual loss of the two tasks (cross-entropy loss).

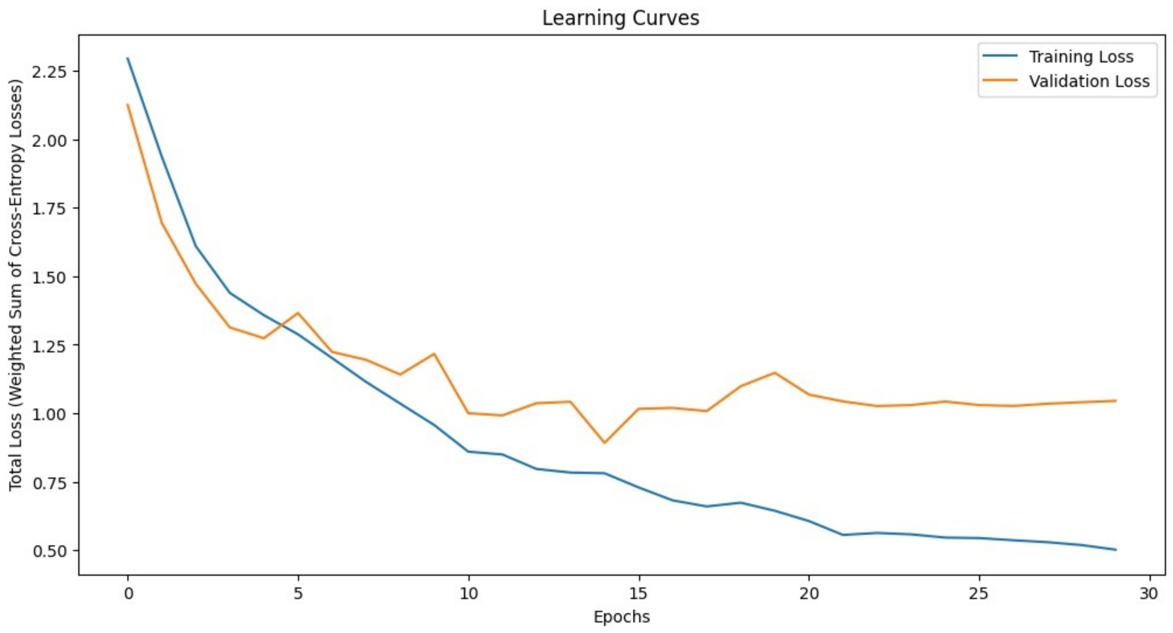


Figure 4.8: Learning curves for the MTL action prediction model (trained on the hand-built sequence variations dataset)

The learning curves show both total loss functions gradually decreasing for both the train and validation sets. The learning process is relatively fast, reaching almost constant values after only 20 epochs. The effect of the learning rate scheduler's first milestone at epoch 20, which reduces the learning rate by and order of magnitude, can be seen from the smoothing of both learning curves. This milestone has been chosen to be at a point where the curves are beginning to stabilize at a constant value. Also in this case, the second milestone, which is at epoch 40, is never actually reached and is used to help force convergence into early stopping. The early stopping mechanism stops the training after 15 epochs of patience with no improvement in validation loss. The model's best weight restoring is disabled because the training process is short, and restoring weights with a patience of 15 would mean eliminating approximately half of the training. Since the learning curves show stable behavior, especially after the first milestone of the learning rate scheduler, keeping the weights from the last epoch has a better effect than restoring the best ones.

It is worth analyzing the confusion matrices and ROC curves, which provide a deeper understanding of the model's behavior on both tasks. Fig. 4.9 shows the confusion matrices for the analyzed model for the test set, both for the action prediction (left) and the object prediction (right). Similarly, Fig. 4.10 shows the ROC curves of the model for the test set for both tasks.

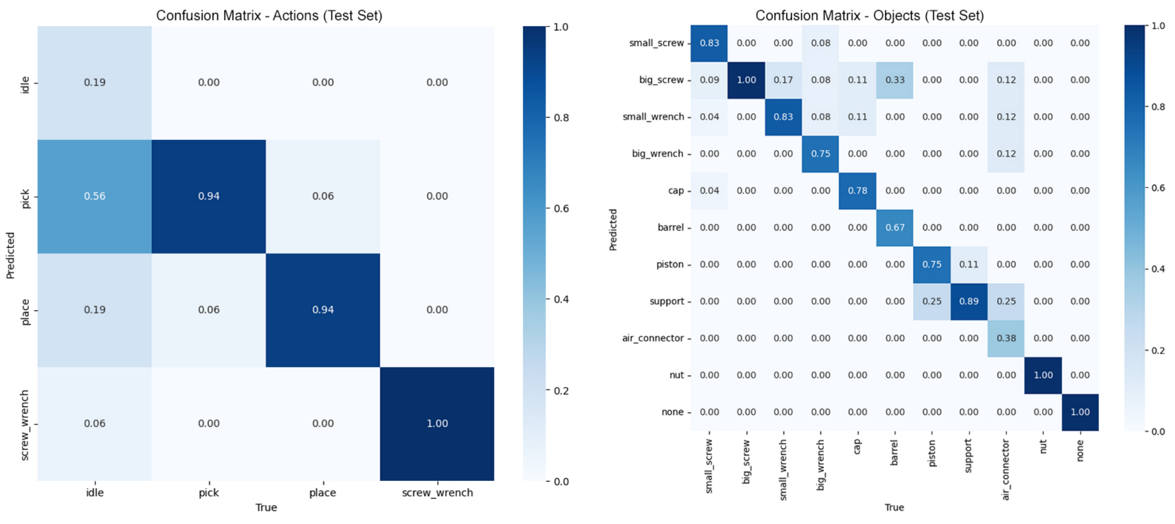


Figure 4.9: Confusion matrices for the MTL action prediction model on the test set (trained on the hand-built sequence variations dataset)

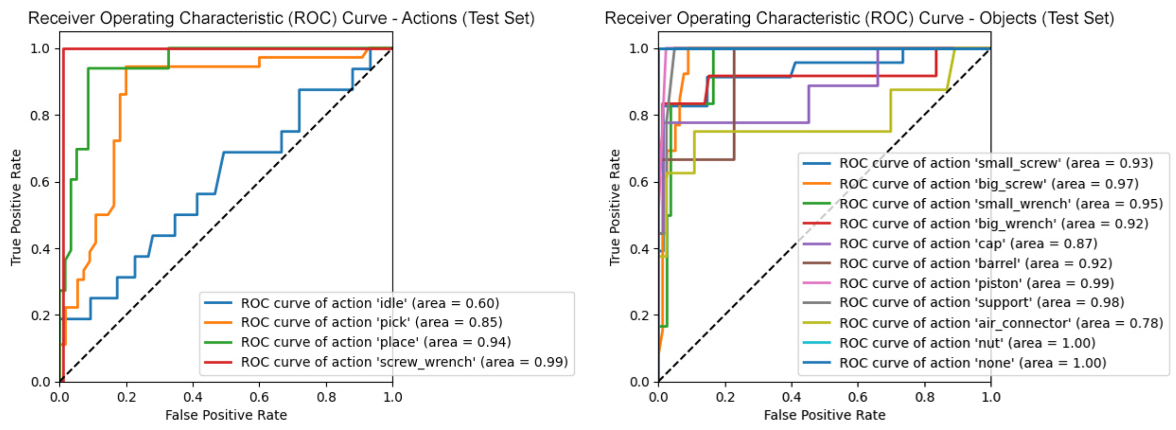


Figure 4.10: ROC curves for the MTL action prediction model on the test set (trained on the hand-built sequence variations dataset)

It can be seen that the idle action is in fact random. Indeed, as mentioned, the base sequence does not contain any idle actions except for the initial one, and thus all other idle actions in the dataset are randomly generated. In particular, it can be seen that the ROC curve of the idle action, shown in Fig. 4.10, is very adjacent to the 45-degree diagonal line, defined as "line of no-discrimination", which corresponds to random guessing. By evaluating the performance by eliminating predictions whose true ground value is idle action, the accuracy of action predictions increases to 0.9467, which is an excellent result.

4.2.2 Model trained on real sequences dataset

Regarding the results obtained by training the action prediction model on the dataset generated from real assembly sequences observed by the computer vision system, the results are worse, as expected, but still considerable good for action prediction, while they are quite inconsistent for object prediction. The performance metrics for this model are shown in the table below.

Metric	**Actions**	**Objects**
Accuracy	0.7282	0.5340
Precision	0.6332	0.4402
Recall	0.6679	0.4570
F1-Score	0.6244	0.4416
ROC-AUC	0.8838	0.8674

Table 4.8: Performance metrics for the MTL action prediction model on test set (trained on the real sequences dataset)

The reason for the high inconsistency for objects will be discussed in detail in

Sec. 4.6, which is dedicated to the results of the real-time implementation. It is anticipated that YOLO contributes more to object misidentification errors than the developed object tracker, which is a symptom of how the fine-tuning performed, while considered adequate in the offline evaluation, is inadequate for a real-time implementation, and requires a larger training dataset to fulfil this purpose. Again, by evaluating the performance by eliminating predictions whose true ground value is idle action, the accuracy of action predictions increases, specifically to 0.8082. The learning curves are not shown as they are very similar to those for the hand-built sequence variations dataset, reported in Fig. 4.8.

Fig. 4.11 shows the confusion matrices both for action and object prediction for this model. It is interesting to note that the pattern of the matrix related to action prediction is similar to that of the confusion matrix in Fig. 4.9, which has been obtained with the sequences generated as variations of the hand-built reference sequence, except for more screw_wrench actions confused with the place action. This indicates that the introduced random variations do indeed correspond to reality, so that the model can effectively learn action and object patterns in real sequences. The confusion matrix related to the prediction of objects confirms that they are predicted quite inaccurately, mainly due to the presence of many objects classified as "none", visible in the last row, because of many actions without a related object in the real sequences dataset. As mentioned, the reasons for this acquisition deficiency will be discussed in Sec. 4.6.

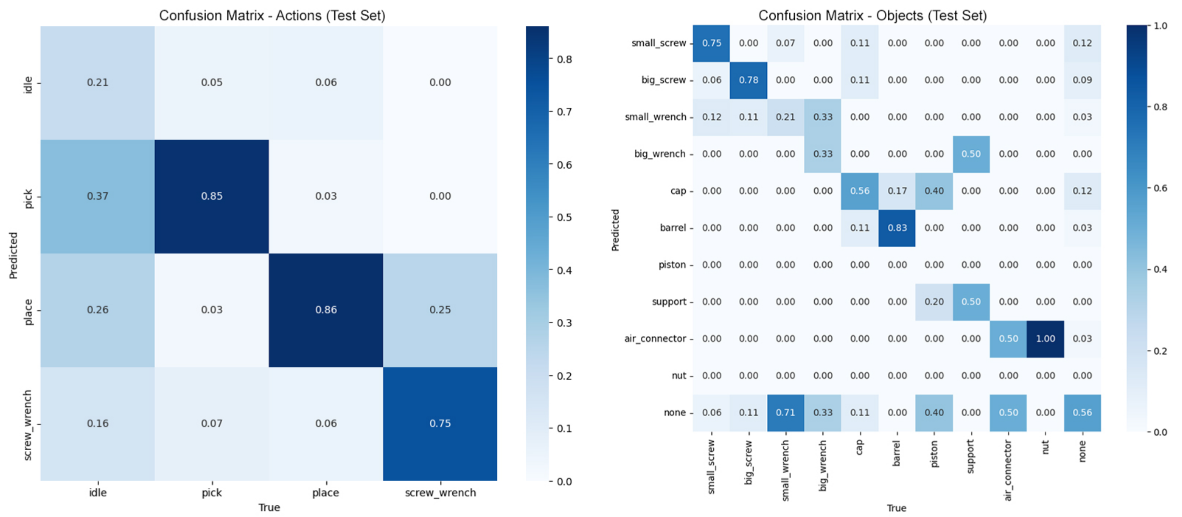


Figure 4.11: Confusion matrices for the MTL action prediction model on the test set (trained on the real sequences dataset)

4.3 Action anticipation

This section describes the results obtained for action anticipation, which in the present work has been intended as the application of action predictions by the robot, giving it decision-making power. The section is divided into the part on camera calibration errors, mainly highlighting the limitations of the depth sensor, and the results of action prediction in depth.

In-depth prediction

The prediction of the pick action deep into the future is performed through the `search_for_future_action` function, which is explained in detail in Sec. 3.3.3, where its pseudo-code is also given. The operation of the function itself is excellent, and any prediction errors depend mainly on the developed action prediction model. Actually, considering the correctness of the prediction of the next pick action, this improves the performance of the model compared to that of generic one-step prediction, since idle actions, which are the noisiest, are not possible predictions, which reduces errors. As expected, reduced confidence values affect the final conditional probability value for the predicted action, which is then compared to the defined threshold. This prevents the robot from performing actions of which it has low confidence and which are therefore likely to be incorrect. In fact, one of the very important goals both for safety, which is maintained even in the case of a wrong action by the robot, and for the correctness of the human-robot collaborative application, is that the robot only performs actions of which it has sufficient confidence, otherwise it is preferable for it to remain idle rather than perform a potentially incorrect action. The confidence thresholds chosen proved to be appropriate for the purpose.

Camera calibration errors and depth sensor limitations

As explained in subsubsection 3.3.3 on the real-time implementation of action anticipation, the intention was to use both camera sensors, RGB and depth, to obtain the coordinates of the components to be picked up in the robot's reference system by homogeneous transformation of the coordinates acquired by the camera. Although the camera calibration produced a fairly low reprojection error, the inaccuracies of the depth sensor do not allow robot picking to be based exclusively on the camera view. In fact, even small errors in the depth measurement can reflect as larger errors in the world reference system. Positional errors of up to $15\,mm$ in the horizontal plane are measured when transforming the coordinates of the centers of the bounding boxes of the objects, which are the points taken as a reference to locate them, which is not in line with the opening of the end-effector used and, in general, with robot picking operations.

This limitation, given by the imperfect accuracy of the depth sensor, forced to fix the positions of the objects in the working plane. The positions have been simply marked on the plane, since a rigid constraint is not required and the opening of the end-effector allows some tolerance in the gripping. Each of the positions fixed on the working plane is associated with a defined label id. Only for the two wrenches, V-shaped supports, attached to the working plane, have been used to elevate them, as they are particularly flat objects. During testing without these supports, it happened several times that the robot picked the wrenches attached to the surface with only the tip of the gripper for the reduced height, and this pressed the wrench against the surface while closing on it. This usually causes the robot to stop due to the forces generated by the contact, which are interpreted as a collision and put the robot into a warning state. Therefore, the use of a stand was mandatory for these two objects.

The position fixation performed is soft both from a physical point of view, as mentioned above, and from a software point of view. In fact, the computer vision system still relies on YOLO's object detections, which are processed by the object tracker. Their position is still calculated with respect to the center of the respective bounding box. Soft-fixed positions intervene when the robot has to pick up a given tracked object, by comparing the transformed coordinates in the robot's base frame with the list of fixed positions and its label id. If there is a label and proximity match between a fixed position and the detected position, the fixed position is used instead of the detected one, so that its imperfections are corrected.

4.4 YOLO fine-tuning

This section presents the results obtained from fine-tuning the `YOLOv9c` model, pre-trained on MS COCO dataset [111], on the custom object dataset specifically built for the application. The steps taken to build the dataset are detailed in Sec. 3.3.4. The results includes performance metrics, confusion matrices, and visualizations of detection outputs on validation set. The evaluation focuses on performance metrics such as loss convergence, precision, recall, and mean average precision (mAP). Additionally, this section discusses the visual inspection of detection outputs to assess the model's qualitative performance.

As anticipated, the images used to build the training and validation datasets have all been acquired with the camera used to develop the related computer vision system, in the fixed working position. This reduces the variability of the context in which the objects are located, reducing the flexibility of the fine-tuned object detection model to the specific application, but at the same time requiring much less data for training. For this reason, although the results obtained are excellent, they have to be considered as limited to this specific application and objects, and not repeatable with the same high performance in a different workstation or with similar but different objects.

The YOLO fine-tuning process has been executed in the Google Colab GPU environment previously described. Recall that the application-specific object dataset consists of 444 labeled images, on which data augmentation was performed by mirroring them, for a total of 888 images, split into 90% for the train set and 10% for the validation set. The total classes of custom objects on which the model has been trained are 10. The complete list with their label ids can be found in Sec. 3.3.4, while the image representing them can be found in Sec. 3.1.

Here is a summary of the configuration parameters used for the fine-tuning:

- Pre-trained model: YOLOv9c;
- Batch size: 16;
- Epochs: 100;
- Target image size: 640 (longest side is resized to 640 while maintaining the original aspect ratio, so 640x480 pixels);
- Optimizer: Stochastic Gradient Descent (SGD) with momentum;
- Initial learning rate: 0.01;
- Momentum: 0.937.

The fine-tuning process took 1 hour and 12 minutes to complete the 100 epochs training. The results have then been exported for evaluation and local use of the fine-tuned model weights within the robot program.

The training and validation losses have been monitored over the training epochs. The YOLO training process outputs three different types of loss:

- Box loss: measures the error in the predicted bounding box coordinates. A lower box loss indicates more accurate localization of objects;
- Classification (CLS) loss: measures the error in the predicted class probabilities. A lower classification loss indicates more accurate classification of objects;
- Distribution Focal Loss (DFL): measures the error in the precision of bounding box prediction, particularly in conditions where the boundary of a detected object may be ambiguous or challenging to discern. A lower distribution focal loss indicates better performance on difficult samples.

The loss convergence graphs, i.e., the learning curves, for each type of loss are shown in Fig. 4.12. They show a significant decrease in both training and validation loss, indicating effective learning and generalization of the model.

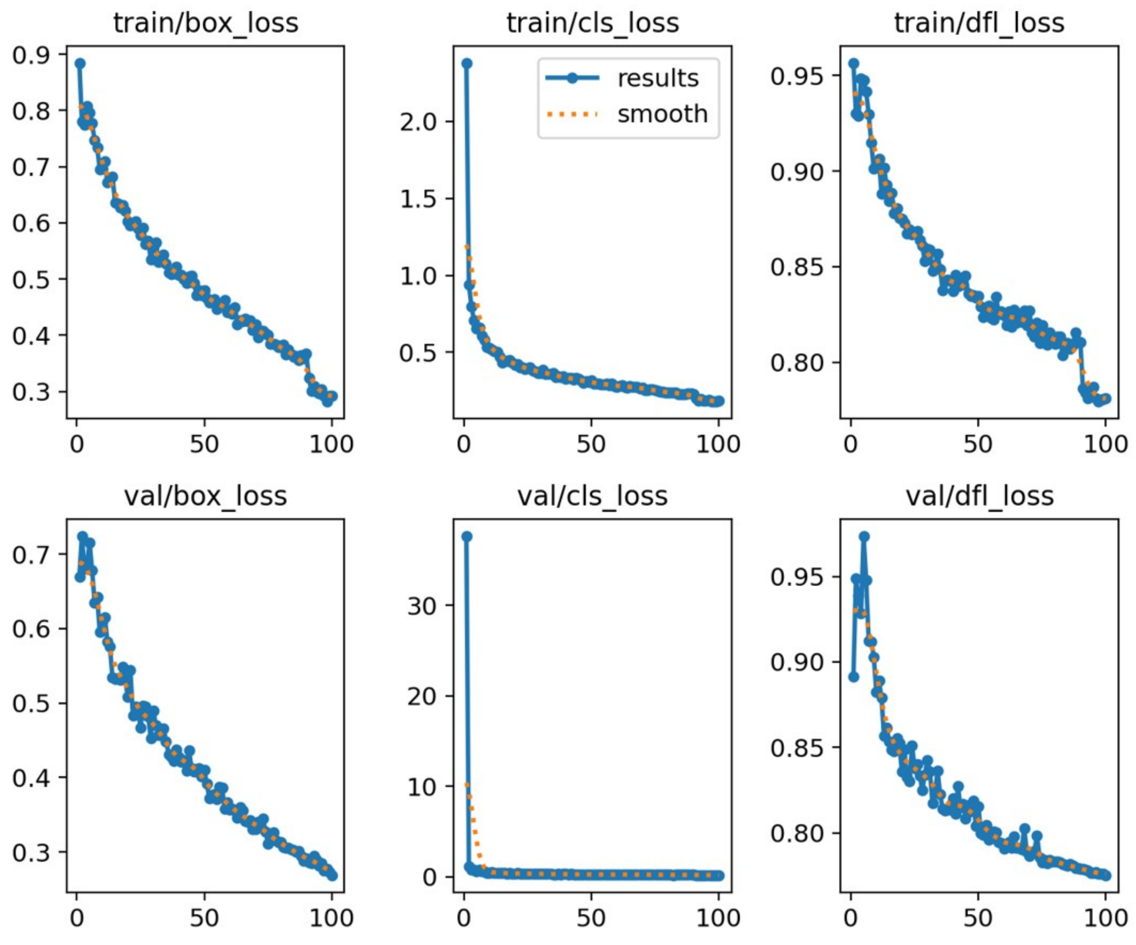


Figure 4.12: YOLO fine-tuning training and validation learning curves: box loss, classification (cls) loss, and distributional focal loss (dfl)

The training losses for the last epoch, which is also the best in terms of losses, are shown below:

Loss Type	**Train**	**Validation**
Box Loss	0.2926	0.2689
CLS Loss	0.1844	0.1871
DFL	0.7814	0.7754

Table 4.9: Train and validation losses for the last (and best) epoch of YOLO fine-tuning

No particular overfitting behavior is noticeable from the learning curves, and the model shows good generalization over the application-specific objects, with consistent results across training and validation sets.

Confusion matrices provide a detailed breakdown of the model's performance, highlighting the distribution of correctly and incorrectly classified objects. Fig. 4.13 show the confusion matrix for the validation set, in both normal and normalized form, to provide both an absolute and relative view of the model's detection errors.

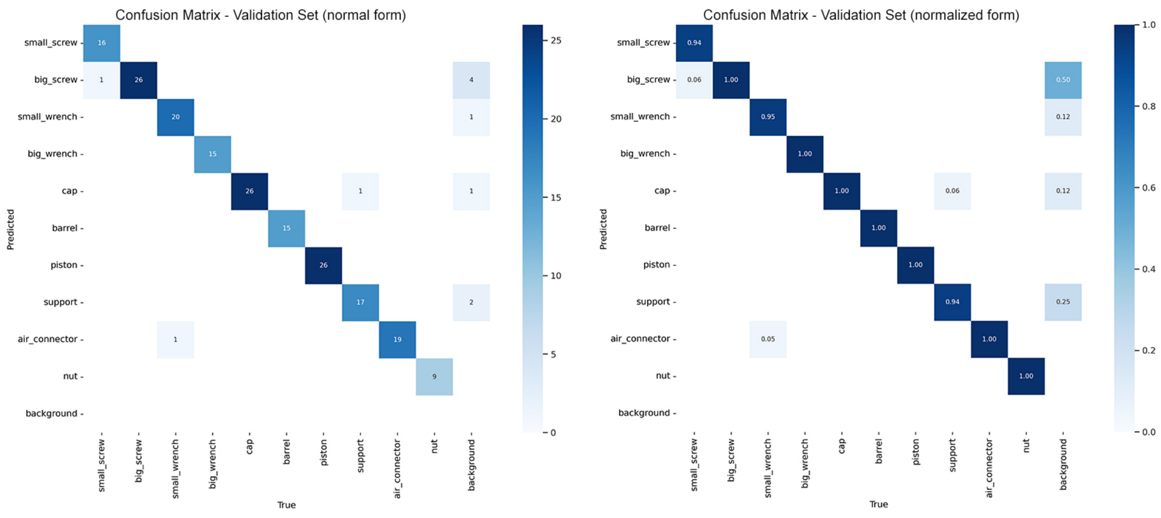


Figure 4.13: YOLO fine-tuning confusion matrix for validation set

The detection results of the model are excellent on all object classes. As can be seen, especially from the normalized form, the extra-diagonal elements, expressing erroneous detections, are very few and represent a very low percentage compared to the totality of objects in the classes. A "background" class is added to the confusion matrix, indicating misidentified objects where there are none. In general, this is the most common type of error rather than a class exchange, unless two classes of objects are particularly similar. The visualization of the background class makes more sense in the normal form of the confusion matrix, where it shows that only 8 objects were misidentified where there are none. Remarkable is the result obtained for the two types of screws used: small and big. These were confused in the entire validation set for only one instance out of a total of 43. Refer to Fig. 3.22 shown in the object detection section to view how similar these two screws are in shape and size. The difficulty increases further considering that the resolution of the processed images is 640x480 pixels, making it difficult in many cases even for a human to distinguish particularly small and/or partially obstructed objects.

Precision and recall are crucial metrics for evaluating the performance of object detection models. Precision measures the accuracy of the positive predictions, while recall measures the model's ability to detect all relevant objects. It is essential to evaluate both to get an overall view of the model's performance. Fig. 4.14 shows the precision and recall during training epochs. Both tend to converge to 1, which is the expected optimal behavior. Then, in Fig. 4.15, the precision-recall (PR) curve is reported, which showcases the trade-offs between precision and recall at various thresholds. This is a very common representation that helps visualize at which threshold both metrics are maximized. These thresholds are evaluated on the Intersection over Union (IoU), which is a measure that quantifies the overlap between a predicted bounding box and a ground truth bounding box.

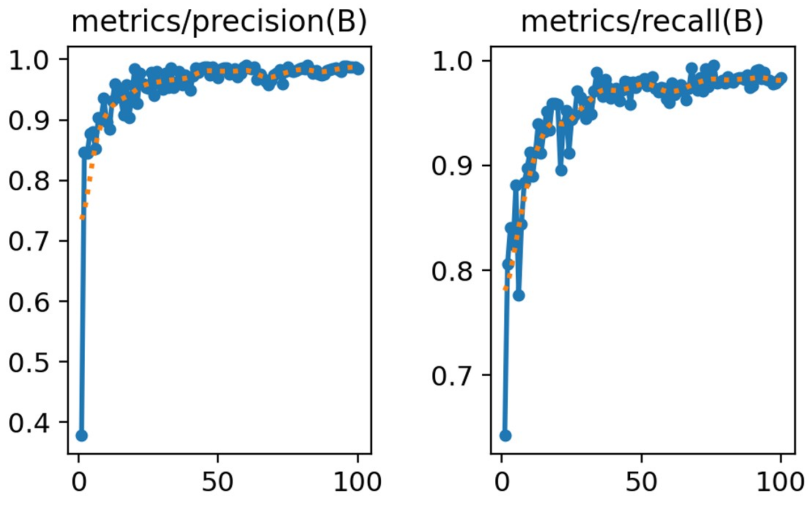


Figure 4.14: Precision and recall curves during training

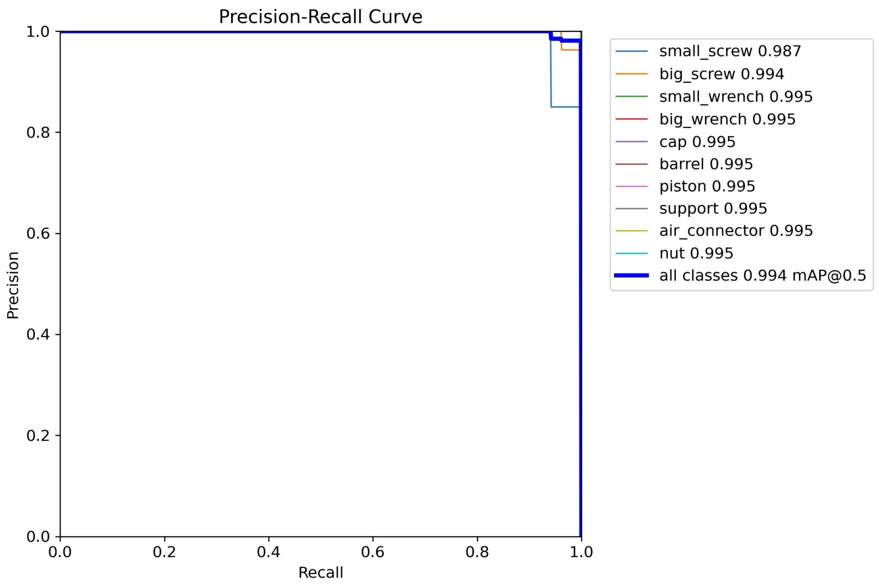


Figure 4.15: Precision-recall curve with average precision for each class

The precision-recall curve indicates a high level of precision and recall across different thresholds for all the classes. The average precision (AP) is the area under the precision-recall curve, providing a single value that encapsulates the model's precision and recall performance [113]. The mean average precision (mAP) is a standard metric used to evaluate object detection models. It extends the concept of AP by calculating the average AP values across all object classes. Fig. 4.15 also shows the AP for all classes at the IoU threshold of 0.50, which corresponds to a mAP (mPA@50) of 0.994. The threshold 0.50 is considered fairly permissive and is a standard for the assessment of mAP for object detection models. The class in which the model performs slightly worse is the small_screw, but the overall metrics can be considered excellent.

The qualitative performance of the model has been assessed by visualizing the detection results on a subset of the validation dataset. The following image, generated directly from the YOLO training process, shows the detections performed by the fine-tuned model on a batch of 16 elements of the validation set, with the corresponding confidence score. These detections have been compared with the actual labels of the corresponding images. The class classification is 100% accurate for this subset of images, and the bounding box overlap is also excellent.

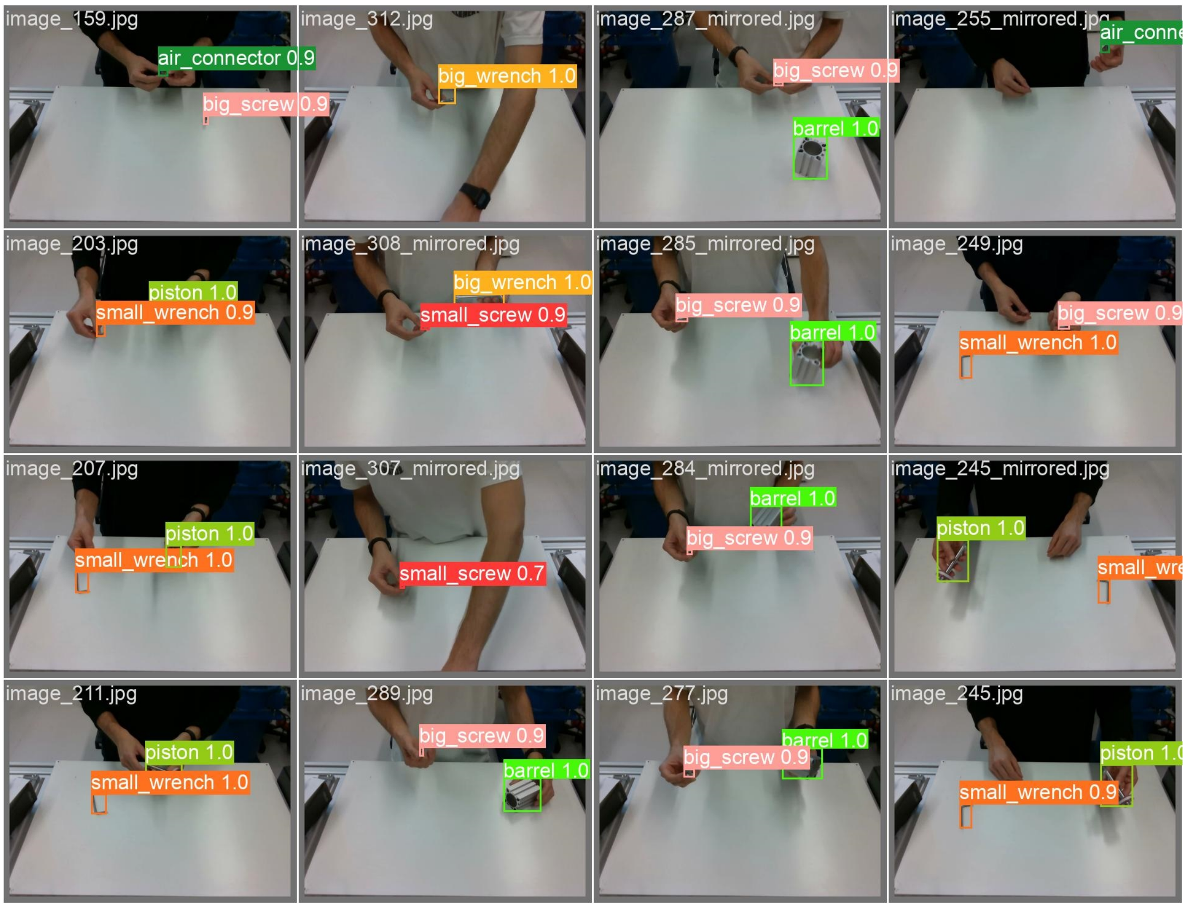


Figure 4.16: Detections performed by the fine-tuned model on a subset of the validation dataset

Regarding the visual inspection of the performance in the real-time implementation, the trained model performs well in detecting instances of the application-specific objects. The confidence threshold for making a detection has been set to 0.75 to reduce false positives. This value has been obtained from the analysis of the F1-score curve, reported in Fig. 4.17, which represents the F1-score across various thresholds. A high F1-score means that both precision and recall are high at the same time, making the corresponding confidence threshold suitable for good performance and for maintaining the model's balance between false positives and false negatives.

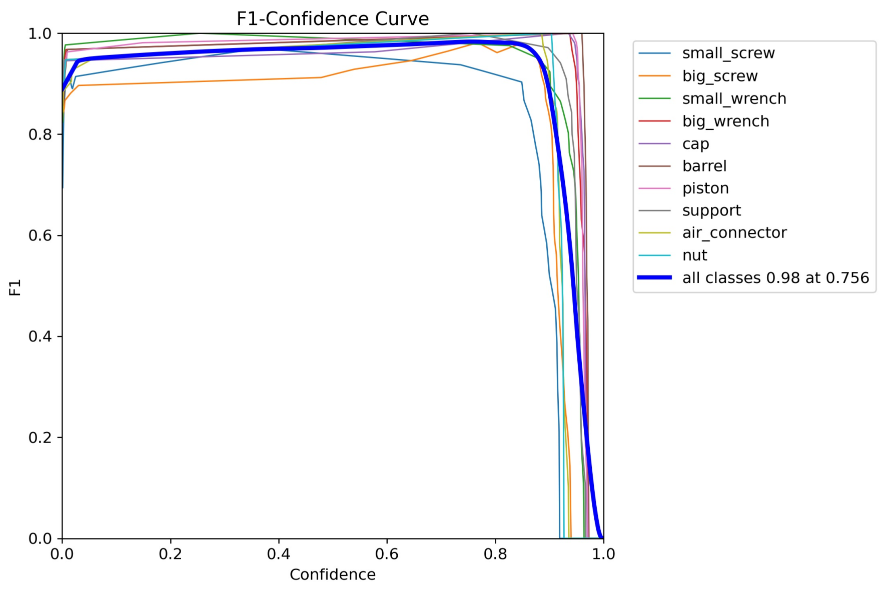


Figure 4.17: F1-score curve with corresponding best confidence threshold highlighted

As a result, there are very few false detections in the real-time application. Class exchanges sometimes occur, especially between the small and big screw, which are the two most similar classes, when these are partially obstructed by the hand. However, this deficiency has been considered more than acceptable. The specifically developed multi-object tracker complements the object detector, improving its shortcomings and achieving robust performance during object handling and movement.

In conclusion, the results indicate that the fine-tuned YOLO model on the application-specific object dataset performs really well, achieving high precision and recall, and a significant mAP across various IoU thresholds. Visual inspections of detection outputs further confirmed the effectiveness of the model. An extension of the dataset could be made to improve the fine-tuned model, since the dataset used is quite limited, but the level of performance obtained is consistent with that required.

4.5 Object tracker

This section presents the results obtained for the developed object tracker. Since the tracker is designed for real-time use, there are no quantitative numerical evaluations of its performance, but an attempt will be made to give an objective assessment and provide an overall overview of the qualities and limitations found.

Moreover, because of the direct dependence on the object detector, i.e., YOLO, an attempt will be made here to isolate the performance of the object tracker from that of YOLO, considering only cases in which the detections occur perfectly. In fact, the object tracker developed for the application has been designed assuming an object detector precision of 100%, i.e., in which true positives are equal to total positives, or better yet, in which there are no false detected objects. This assumption, necessary to enable the tracker to be developed from a knowledge base considered reliable and of general use for tracker development, is very strong. In fact, although the confidence threshold value adopted has been chosen quite high (0.75) and in agreement with the higher F1-score, thus with the better balance between precision and recall, YOLO provides some false detections during real-time use, especially for objects that are partially obstructed by the hand that is manipulating them. Obviously, this negatively affects the performance of the tracker, precisely because of this inconsistency of the assumed perfect precision. Regarding the performance of the tracker in the overall real-time implementation, which thus also takes into account the errors of the object detector, this is discussed in more detail in Sec. 4.6.

While this is not the main focus of a multi-object tracker, it is started by saying that the developed tracker performs excellently on a single object, or at least with a limited number of objects that do not interact with each other in any way, and in which the human operator's hands do not transit over other objects to interact with the object of interest. In this simplified scenario, assuming, as mentioned, a 100% precision from the object detector side, the tracker shows actually perfect accuracy in tracking the object of interest, providing correct information about its state of movement, visibility, and being held in a hand. At the same time, the unique id of the object is maintained throughout the tracking, with excellent re-association after the object is manipulated by the hand and then hidden from the camera for a few frames.

As complexity increases, i.e., the number of objects in the scene and their interactions, the following known errors can occur:

- It is possible that if an object held in the hand is released close to an object of the same class, it is incorrectly re-associated with the other object of the same class and not with the object actually released from the hand. Note that this only occurs if the view of the other object is also interrupted for some frames, for example, because the hand passes near this other object while leaving the current object in the hand, partially or completely blocking its view. Otherwise, if the other object of the same class remains visible all the time, the visible objects logic of the algorithm should maintain continuous tracking and make sure that this does not happen;

- The values of the threshold distances for hand interaction in the logic of the tracker algorithm, described in Sec. 3.3.5, are independent of the object under consideration and have been set to reasonable values for small objects such as the screws used in the related assembly. This means that, for large objects, or objects with at least the predominant dimension being quite large, e.g., the piston and the large wrench in the related assembly, it may happen that the point of finger contact is quite far from the center of the bounding box of the object. This distance is sometimes incompatible with the threshold distance values set, which results in missed updates of the state of objects held in the hand. More permissive threshold values could be set, but at the risk of increasing the number of objects falsely identified as hand, so it is necessary to keep these distance values small with this trade-off in mind.

- As described in detail at the end of the object tracker description in Sec. 3.3.5, it has been developed a feature that makes it possible to distinguish which object is actually being picked up, even if multiple objects are in close proximity to the hand and, more importantly, are simultaneously not visible to the camera. It has been explained why the system cannot rely on the actual grasping action, i.e., when clasping the fingers of the hand around an object, as a human would do to accomplish this task. The developed feature thus relies only on the current position of the hand, in particular the tips midpoint, adding a mechanism of object persistence in the hand, expressed in frames elapsed by the object in the proximity zone that makes it considered as being held in the hand. To summarize the main function of this feature, it is to "lock" the object held in the hand, so the tracker will not define a further object as being held in that hand until it is released or expires, even if the hand were to approach other objects. Hence, the factor of persistence in the hand before locking is essential, since in a working table where there are multiple components to be assembled, the situation where, in order to reach an object to be picked up, the hand passes over other objects, even covering them to the camera, occurs continuously. This persistence value in frames has been set to 3 in the real-time implementation. Its value depends strictly on the acquisition FPS and the dynamic nature of the actions performed.

Setting it too low means locking any object the hand transits over, while setting it too high means failing to lock any object unless the action is performed very slowly.

Having clarified the function of this very important feature, it is described here a problem that it has failed to remedy, and which has remained unsolved due to the lack of viable solutions. This is considered the greatest known limitation of the developed tracker. As mentioned, once an object has been locked as in hand, the tracker will not define a further object as being held in that hand until it is released or expires. In an application of only object handling, an object would be continuously picked up and released, causing very little trouble to the developed feature. The problem arises in the assembly application, which is the one of interest in this study. Here, objects are picked up and then assembled. When an object is assembled into the rest of the assembly, it is often no longer visible, or in any case not detected by YOLO because it is part of a larger object with a different shape and size. This means that, for the information available to the tracker, the object is picked up and is no longer released. At some point the expiration mechanism will operate to stop tracking the object, but until that time, the hand will be found to be busy, even though it may actually be free because it has already finished assembling the previously held object. If in the meantime, before the expiration of the held object, another object is picked up, the tracker is unable to react to the event. Also, since the next object has already been picked up, so it is probably no longer visible by the time the previous one reaches expiration, the new held object is never updated in any way.

It is intended to conclude the object tracker results section with some thoughts on the possible improvements to the last problem listed above, which, as mentioned, is the greatest known limitation of the developed tracker when used to track assembly operations.

With only the information available to the tracker, it would be very difficult even for a human to define when an object is actually released from the hand. Therefore, a detection system is needed that can notify this event, i.e., that can recognize the objects for which it has been trained even when they are assembled in an assembly, thus partially non-visible. This is not only extremely difficult, but may not be strictly necessary. In fact, some objects may be completely not visible after being assembled, not just partially. In this situation, where the manipulated object can no longer be seen, a human would be able to declare the end of the manipulation of the object by observing the currently empty hand. The definition of an empty hand falls into that elementary category of problems for a human being, in which even a child would achieve excellent results, but extremely complicated for an artificial system, in this case computer vision. This would involve the development of a dedicated neural network, either convolutional for direct image analysis or using a pose detector as done in related work, that can recognize hand positions in which it is likely to contain no object, e.g., hand with fingers fully extended. This must be matched with information from an object detector that verifies the actual absence of visible objects in the hand. This information must then be properly associated with the tracker, which is not trivial at all. This complex system has not been investigated in the present study, but it was deemed to provide the only hypothesized, albeit complex, idea to overcome the main object tracker limitation described.

4.6 Real-time implementation

In this final section of the results chapter, all considerations and qualitative evaluations of the implemented autonomous real-time system as a whole are reported. It is noted that, besides the objectives of the individual parts of the work, one of the main goals was to develop a system light and fast enough in all its parts to enable optimal real-time implementation. It is obvious to generally expect lower performance from any type of program or model implemented in real-time compared to an offline evaluation. This is because, in real contexts, many sources of disturbance in the input data are added that might not be present during evaluations with sampled data.

First, it is good to summarize the elements of which the system is composed and their interaction. For all the details, please refer to the respective sections in the materials and methods chapter.

- As explained in Sec. 2.4, perception is the principal element in making the robot collaborate with humans, and the perception of the computer vision system integrated with the robot is used to detect human poses and objects.

  The starting point of the program is provided by data from two state-of-the-art artificial intelligence models:

  - MediaPipe Hands solution: provides data related to tracking the human operator's hands, with 21 landmarks per hand, from which the coordinates previously described are extracted;
  - YOLO (v9): object detector fine-tuned with a dataset of application-specific objects. It provides data related to the class and position of objects in the workspace, detecting their bounding box.

- A neural network for action recognition has been designed and developed from scratch, including the dataset of clipped actions for training it. The model chosen, among those investigated, for real-time implementation is the Simple LSTM, for the conclusive considerations made in Sec. 3.3.1. This only takes into account data related to the position of the hands and not the objects. The model produces a classification of the action currently performed by the human operator on the set of 4 typical assembly operations on which it has been trained.

- A multi-object tracker has been specifically developed to effectively track objects on the work table and, especially, to keep track of the objects with which the hands interact, even when these are hidden from the camera by them, and therefore not detectable by the object detector. This is primarily based on the class and position detections of the objects provided by the object detector, but also on the hand landmarks provided by MediaPipe, particularly on the mid points of the thumb and index fingers of both hands, which have been taken as reference points as the most likely for grasping an object.

- A second neural network, for action prediction, has been designed and developed from scratch. For this as well, the training dataset as been specifically created, with two modes: one based on hand-built sequence variations and the other on real sequences captured by the computer vision system. Although the considered pneumatic cylinder can be assembled following very different sequences, since many components do not require a mandatory assembly order, it has been preferred in both datasets to adhere to a precise assembly sequence, to verify the collaborative system's capabilities in a condition where the sequence must be strictly followed. In more permissive assemblies in terms of possible assembly sequences, a system that performs well under stricter conditions can only perform even better with more freedom available.

  The neural network developed for action prediction is a multi-task learning classifier. It takes as input data those related to the performed actions, provided by the action recognition model, and the currently manipulated objects, provided by the combination of the object detector, i.e., YOLO, and the developed object tracker. The network produces the prediction of the most likely next action and the object involved in that action, based on the current condition of the provided data, relying on the assembly sequences on which it has been trained.

- A specific function searches deep into the future for a certain action that the robot can perform instead of the human, i.e., action anticipation, based on the predictions of the action prediction model. In the present study, only the picking action has been considered executable by the robot instead of the human, due to the technical limits explained in Sec. 3.3.3. Therefore, decision-making power is attributed to the robot, based on established necessary confidence levels.

The actual application of the pick action is made possible thanks to the transformation of coordinates from the reference system of the camera frame to that of the robot's base. Both the RGB and depth (D) sensors of the camera are used here. To obtain the homogeneous transformation matrix, a camera calibration procedure has been performed. Once the pick action is executed, the object is delivered directly to the operator, through a loop that obtains the coordinates of his hand in the real world, based on the landmarks provided by MediaPipe.

Communication with the robot is made possible by a server running on the robot controller and the specific Python library. A wrapper class has been developed to provide effective communication functions with the robot for the application.

As can be understood from this summary of the elements, the overall developed system is complex and articulated. Notably, there is a strong dependence on the parts of the program, with the main sequential structure, repeated multiple times before, which, in order, consists of: action recognition, action prediction, and action anticipation. This sequential dependence causes errors to accumulate during the program's execution, starting from the detection of the positions of the hands and objects, which can be considered as the starting data, up to the actual anticipation of the human action by the robot. For this reason, particular attention has been paid to creating an efficient neural network for action recognition, with the construction of an adequate dataset to train it, precisely because it is at the base of the subsequent parts of the program.

In this results chapter, generally excellent performance of all the developed and implemented models for offline evaluations has been shown. However, as mentioned, it is logical to expect lower overall performance of all models in real-time implementation compared to their offline evaluation. Furthermore, there will be errors due to their connection, which will significantly impact the performance of the individual models.

The developed computer vision system, integrated with the collaborative robotic arm, efficiently recognizes, predicts, and anticipates human actions in the studied collaborative assembly scenario. The complete assembly of the considered pneumatic cylinder has been repeatedly tested by a human operator with the robot's collaboration in supplying objects. The result is a natural and intuitive human-robot interaction, where, from the start to the end of the assembly, the human operator does not need to press buttons or pedals of any kind to communicate with the robot. The robot is given decision-making power over its actions, but the application is entirely human-centered, as the robot's actions are based on the information perceived about the human and the work environment, provided by the integrated computer vision system. The impression from firsthand use of the developed system is that of being in the presence of a thinking collaborator, where one feels to be the leader of the operation, yet is uncertain about how the collaborator will behave, due to the use of artificial intelligence models. This characteristic should be intended positively, as it represents the same uncertainty that characterizes human-human interaction, where, without programmed automatisms, the interaction is constantly based on the perception and intellect that the operators have of the other human and the environment, without a predefined scheme. This flexibility enables the completion of collaborative tasks between humans that would not be possible for a single operator. On the contrary, this uncertainty about the behavior of the robotic collaborator should not be intended as a fear of the actions it might take because they are wrong or even dangerous for safety. Indeed, the robot does not perform risky actions and generally does not make mistakes; at most, it remains inactive when it is not confident enough about the action to perform. This criterion has been used to program the entire system, to increase the safety and trust the operator places in the robot, fundamental to profitable human-robot collaboration [22]. Again, describing subjective sensations from firsthand use, besides perceiving the presence of a thinking collaborator, albeit robotic, there is a high level of trust in it, in its decisions, and its movements. After a few assembly sequences, any hesitation an operator might have in collaborating closely with the robot disappears. For example, after some initial trials, it becomes natural while keeping the eyes on the component being assembled and hearing the robot activate to pick up an object, to extend the hand without even observing the robot's behavior, convinced that the object it will hand over is probably the one most needed at that moment, and above all, that in no way will the robot or the object it manipulates hit the hand as it approaches the delivering point, where the actual physical interaction with the robot occurs. This level of trust represents the same necessary for a profitable human-human collaboration, which is very difficult to achieve with a cobot, especially if it has decision-making power and is not programmed to perform only repetitive actions.

These characteristics form the basis for creating flexible human-robot collaborative applications, where human capabilities such as knowledge of the component to be assembled and the ability to deal with unexpected situations are exploited, improving working conditions and increasing efficiency through simple and effortless communication with a collaborative robot.

With the modest computer hardware available, described in Sec. 3.2, a stable 10 FPS acquisition rate is achieved, using both the RGB and depth (D) sensors of the camera, and with the MediaPipe Hands model, YOLO, and the multi-object tracker running on each frame. The action recognition model runs at a frequency of 10 FPS, the same used to sample the clipped actions on which it has been trained, which corresponds to almost all acquired frames. Where the acquisition FPS is higher, some frames are skipped for this model. On the other hand, the action prediction model only runs when a new action is recognized, and the robot control part only runs when it decides to undertake an action. There are no FPS drops or program freezes during execution due to concentrated workloads, as most models run continuously and the required computational power is fairly constant. No particular bottlenecks have been identified, so the system's speed can be scaled without problems by increasing hardware power. However, consider that the frequency selected for the action recognition model, in this case 10 FPS, would make excessive increases in real-time acquisition FPS unnecessary unless the model is re-trained with a dataset of actions sampled at higher FPS. The model that most requires computational power and affects reducing acquisition speed is MediaPipe, which is acceptable given the excellent hand-tracking it performs.

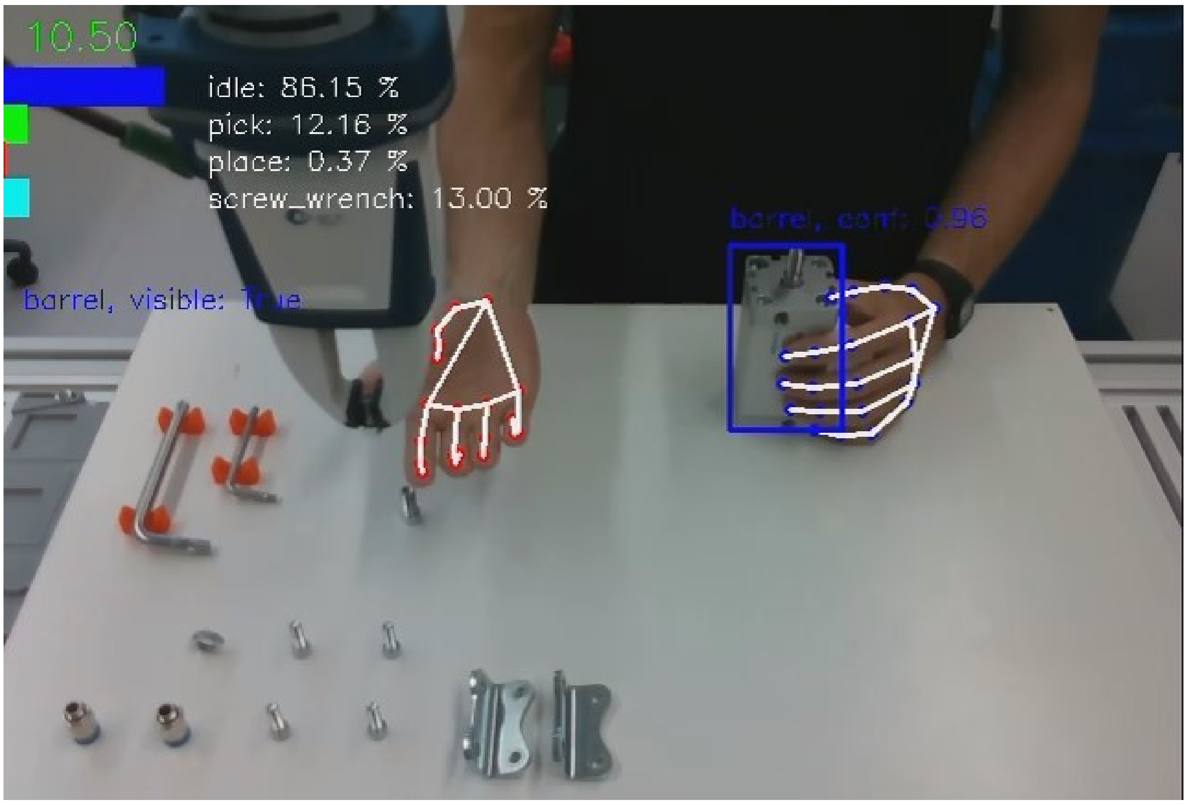
Now, the specifics of the real-time implementation of the individual models are reported, focusing on their interaction within the robot's program.

- The action recognition model, which is central to the functioning of the rest of the program, achieves optimal performance in real-time. It can be confidently stated that, in the realized work setup, which is the same in which the dataset of clipped actions used to train the model has been created, a qualitatively correct recognition of more than 90% of actions is achieved. The exponential smoother, which alleviates the model's output changes over time, generally results in more frequent false negatives, where the necessary confidence for classification is not reached, rather than false positives, where one action is incorrectly recognized as another. The two actions most subject to confusion are place and screw_drive, as highlighted by the study of confusion matrices, particularly tending to confuse place with screw_drive, while the latter is recognized quite reliably. The pick action is recognized almost perfectly, with only a few sporadic cases of error. The idle action is also reliably recognized, providing the necessary alternative classification to all other actions, which is why it has been introduced.
- The MediaPipe Hands solution provides the hand landmarks positions reliably and accurately. The tracking is very continuous, with interruptions occurring very rarely, mainly due to positions with poor light exposure or low contrast with the background, but recovering after a few frames as conditions change. There is a problem of landmarks distortion for a partially visible hand, for instance,

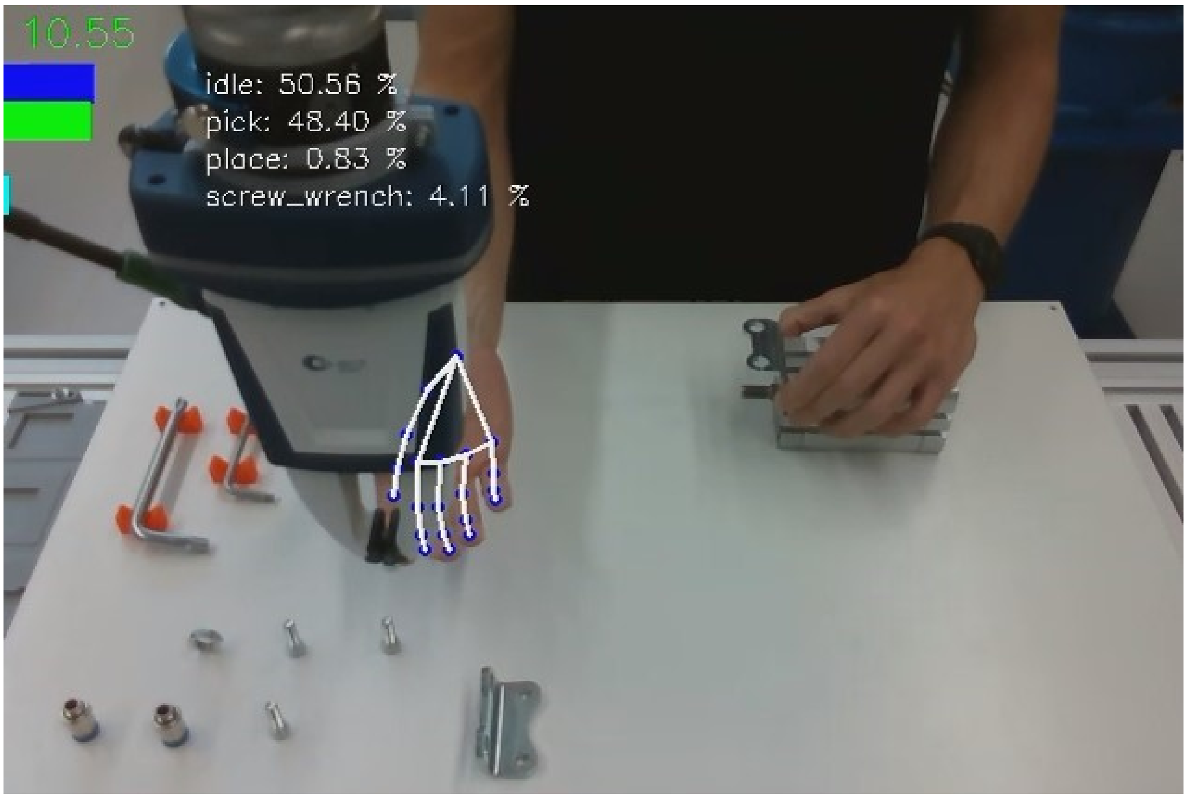
during interaction with a large object and, especially, when the robot delivers a component to the human operator, as the end-effector obstructs the hand's view from the camera. For this reason, the recording of recognized actions sequences is suspended during the robot's component delivering. In the same conditions of partial hand visibility, an even more serious issue than distortion is the incorrect swapping of the right and left hands, which has been noted to occur predominantly when the hand is fully open with fingers spread and something obstructs a clear view. Indeed, the two open hands, one with the palm and the other with the back facing up, differ only in appearance but not in the landmarks position, which is identical. In the current real-time implementation, where the action recognition model only considers the right hand, as does the action prediction model, and objects are delivered only to the right hand, there is no tangible difference between the non-recognition of the right hand or its incorrect swap with the left hand, but this error could lead to significant complications for two-handed applications. The following two images show the described cases:

(a) The right hand's thumb is severely distorted due to visual obstruction by the robot's end-effector. This distortion occurs dynamically and causes the action recognition model to lose confidence in the idle action, which was near 100% a few frames before;

(b) The right hand is incorrectly recognized as the left hand, indicated by the blue color.



(a) Landmarks distortion error for view occlusion



(b) Hand detection swapping error for view occlusion

Figure 4.18: MediaPipe hand landmarks distortion (a) and hand detection swapping (b) errors for view occlusion

- The YOLO object detector, as described in Sec. 3.3.4, has been fine-tuned on a relatively modest dataset of application-specific objects. It showed slightly lower real-time performance compared to its offline evaluation. Nevertheless, the results remain satisfactory, and the power and flexibility of this tool are highly appreciated, allowing it to be trained relatively easily on a custom object dataset. The confidence threshold value set at 0.75, following the study of the F1-score trend, proved suitable for avoiding almost all false detections. However, some misidentifications of detected objects occur, especially when a hand interacts with an object, partially obstructing its visibility. In Fig. 4.19, a misdetection of the "big wrench" object is shown, which is mistaken for the "support" object, with a confidence level of 0.78, just above the threshold. The metallic color and the 90-degree curvature of the wrench's end closely resemble the color and shape of the support, but a human observer would not make such a classification error.

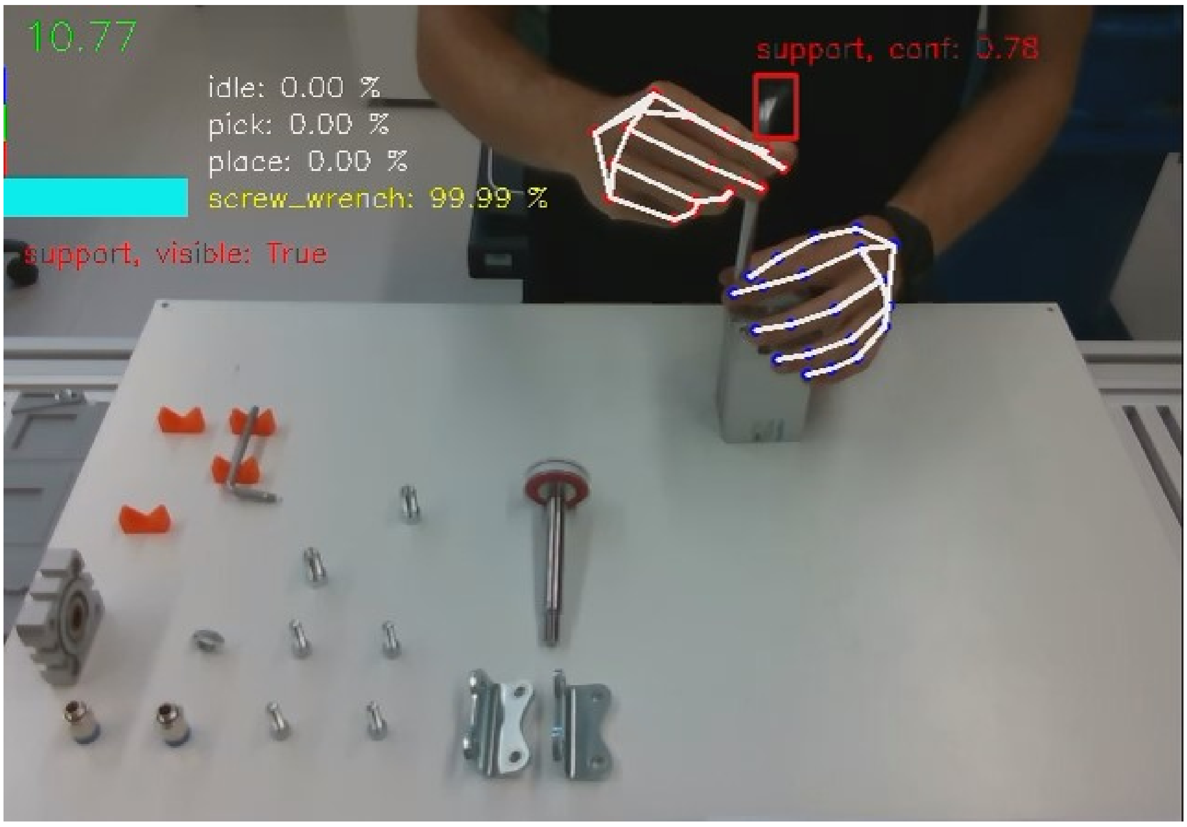


Figure 4.19: YOLO misdetection between big wrench and support objects

The differentiation between the two screws, small and large, is very good, despite their similarity, as shown in Fig. 3.22. Generally, detection errors by YOLO negatively impact the tracker's performance, which has to work with inconsistent detection data. In turn, making errors in tracking objects held by the operator, the object tracker provides incorrect information to the action prediction model, which suffers as a result. Better results with YOLO can be achieved simply by increasing the size of the application-specific object dataset.

- The multi-object tracker developed shows near-perfect behavior when the detections provided by YOLO are accurate. The known limits of the tracker under these ideal conditions provided by the object detector are detailed in Sec. 4.5. Recall that the developed object tracker has been designed assuming an object detector precision of 100%. When YOLO provides incorrect detections, the tracker loses the consistent data basis necessary for clear and continuous tracking. The tracker's problems are not so much related to the non-identification of an object, which it can handle perfectly, especially if the object is hidden by a hand manipulating it, but rather the misdetection of one object for another, as shown in Fig. 4.19. For tangible comparisons, imagine tracking the movement of objects in the scene with your own eyes. The non-identification of an object by YOLO would correspond to having an obstructed view for some moments, at least of the area concerned with the object. If this obstruction of view is not too prolonged and if, above all, no major changes occur in the scene meanwhile, when the view is clear again and vision resumes correctly, it should not be too difficult to continue tracking from where it left off. However, a misdetection of one object for another by YOLO could be compared to a visual defect in observing the scene, where it becomes difficult to distinguish the components seen, sometimes confusing them. It is clear how tracking now becomes much more difficult, as the certainty basis is lost.

  The object tracker has to face both problems simultaneously. Moreover, always remember that the tracker does not actually process the image but relies only on YOLO's object detections and hand tracking to define grip points. This means that an error by YOLO represents a deficiency in the only data available to the system. Fig. 4.20 shows the very useful but undoubtedly limited information provided by YOLO to the object tracker, including bounding boxes and label ids of detected objects.

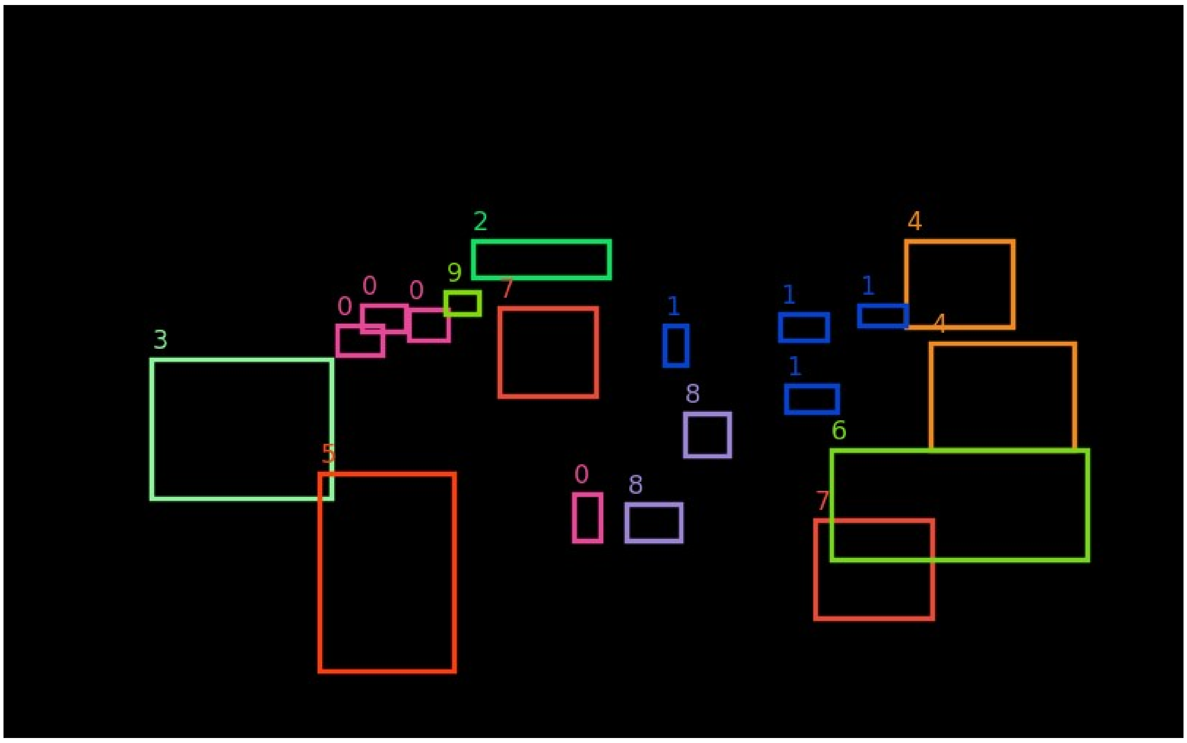


Figure 4.20: View of bounding boxes and label ids detected by YOLO on a blank frame

Therefore, the comparisons based on our vision to empathize with the tracker's difficulties, while useful for understanding, are overly permissive and relevant to the reality observable with our eyes, whereas the useful data obtainable from an object detector are very limited and, therefore, when they are wrong, they have a strong negative impact on the performance of the object tracker.

In the first described error case, a non-detection of the object that is then picked up, the hand simply remains "empty" for the tracker, without information about the object held. In the second error case, a misdetection where one object is confused with another, the false object presence mechanism in the tracker, described in Sec. 3.3.5, ensures that fluctuating misdetections received from the object detector are quickly discarded, which still results in an empty hand. These are the reasons for the presence of many "none" objects in the real sequences acquired to build the action prediction model training dataset, whose negative effects on object prediction are visible in Fig. 4.11.

- The action prediction model works very well in real-time, provided it is given accurate data. For data related to the actions performed by the human operator, it relies on the excellent classifications from the action recognition model. However, regarding objects, as mentioned earlier, YOLO performs well but not optimally in real-time, failing to identify some objects under certain conditions or misclassifying identified objects. These errors then impact the object tracker, which sometimes provides incorrect information about the object held, often indicating an empty hand even when it is not. Given the greater weight attributed to actions rather than objects in predicting future actions, as they are central to defining the assembly sequence, the action prediction model still manages to make satisfactory forecasts primarily based on solid data from the action recognition model.

Both the action prediction model trained on a hand-built sequence variations dataset and the one trained on real sequences dataset have been tested in real-time. Both performed very well, successfully following the defined sequence in detail. Additionally, both show good insensitivity to the idle action, with the next predicted action generally remaining unchanged but with reduced confidence. Naturally, the first model performs better, as evaluated in the offline assessment and as hypothesized from the beginning. However, the performance difference with the model trained on the real sequences dataset is not significant under conditions where the underlying models function correctly. The performance of the latter tends to degrade more quickly when the input data is noisy, such as when some erroneous objects are detected within the sequence. Generally, the model trained on the hand-built sequence variations dataset shows greater robustness in real-time use.

A very interesting result for application flexibility, especially in the manufacturing industry, has been achieved by creating a collaborative system with no prior knowledge of the assembly sequence. Additionally, there is ample room to enrich the real sequences dataset, which could significantly improve this model's performance. As previously noted, the no prior knowledge mentioned refers exclusively to the assembly sequence, not the component to be assembled, whose parts must be sampled with numerous images that must then be manually labeled to train YOLO. Moreover, consider what has been said about the modest yet insufficient size of the dataset used in this work for fine-tuning YOLO. This burdensome process necessary to train the object tracker constitutes the most concrete limitation in realizing a collaborative robotic system for assembly with no prior knowledge of the assembly sequence.

- Finally, concerning real-time action anticipation, the next pick action is very well searched in-depth based on the action prediction model. The predicted object associated with the pick action is generally consistent with the sequence. Qualitatively, the function that searches for the future action in-depth, combined with the action prediction model, manages to provide in real-time more than 3/4 of the correct predictions of the object associated with the next pick action, with the set confidence thresholds. As mentioned, these thresholds are set sufficiently high to give the robot a safer behavior, preferring to refrain from an action it is not sufficiently sure about rather than executing a wrong one.

Regarding the actual execution of actions by the robot, the implementation suffers from the main limitation of the camera's non-optimal depth sensor, which required soft-fixing of the objects, both physical and software, as described in Sec. 4.3. The robot has been run at a reduced speed for the tests but has also been tested at higher speeds without encountering issues. The correct safety levels are ensured by the collaborative robot used and the confidence thresholds set in the robot program for action execution.

Chapter 5

Conclusion and future work

This chapter presents the conclusions of the work carried out, focusing on the importance of the results obtained. Subsequently, some considerations and ideas on possible future work will be discussed.

The main objective of the proposed work has been fully achieved, developing a computer vision system integrated with a collaborative robotic arm to recognize, predict, and anticipate human actions in a collaborative assembly scenario. Different AI-based technologies have been investigated to see which one best fulfilled the purposes, and it has been possible to create a high-performing autonomous real-time system. The resulting system allows for natural and intuitive human-robot collaboration, remaining fully human-centric while giving the robot decision-making capabilities. This is made possible by the perception provided to the robot by the integrated computer vision system, achieving human-robot communication without the need for physical commands or specific explicit gestures. In this way, human capabilities such as knowledge of the component to be assembled, dexterity, and the ability to deal with unexpected situations are exploited, improving working conditions and increasing efficiency and accuracy through simple and effortless collaboration with a cobot.

In Sec. 4.6, the subjective feelings experienced from firsthand interaction with the collaborative system created are described. They are summarized here:

- The use of artificial intelligence models allows the robot to have actual decision-making power, although limited to a set of performable actions;
- There is a perceived presence of a thinking collaborator, given the variability of actions performed by the robot based on the interpretation of real-time information;
- The actions performed by the robot, which anticipates human actions, generally

correspond to the action to be undertaken immediately upon completion of the current one, enhancing the feeling of collaboration with a thinking and intelligent entity;

- The human action anticipation system is designed with the necessary confidence levels so that the robot does not perform an action that it is not very sure of, and therefore potentially wrong, but rather remains inactive. This precautionary behavior increases the level of trust the human has in the collaborative robot, which is fundamental to profitable human-robot collaboration [22], as it never takes risky actions. Obviously, the right compromise has been sought so that the robot does not remain almost always inactive due to excessive caution, otherwise the opposite effect would be obtained, reducing trust by not intervening when needed;

- Although collision avoidance methods have not been implemented, the delivery function of the object to the human operator's hand, based on the coordinates of the hand landmarks, works iteratively so that collisions between the hand and the cobot are very rare and often so superficial that they do not cause the robot to stop for safety. This further increases the level of trust given to the robot, this time concerning the aspect of safety, which, as emphasized several times, is the most important in collaborative robotics applications. As described previously, after some initial trials, it becomes natural while keeping the eyes on the component being assembled and hearing the robot activate to pick up an object, to extend the hand without even observing the robot's behavior, convinced that the object it will hand over is probably the one most needed at that moment, and above all, that in no way will the robot or the object it manipulates hit the hand as it approaches the delivering point.

g The integration of state-of-the-art models such as MediaPipe Hands and YOLO has allowed for the creation of a solid base for the perception of the computer vision system, starting from sufficiently robust and reliable hand tracking and object detection data. From here, it has been possible to build an optimal system for recognizing, predicting, and anticipating human actions in a human-robot collaborative assembly scenario. The robot program, which manages the functioning of the autonomous real-time system, involves: logic algorithms, neural networks, the computer vision control system, and the robot control system. Special attention has been given to the development of a custom multi-object tracker, based on information from YOLO object detection and hand positions provided by MediaPipe. This allowed for the tracking of objects, that is fundamental for determining the objects with which the human operator interacts during assembly, even when these are partially or totally obscured by the hand manipulating them, as explained in detail in Sec. 3.3.5. Besides the elements of the robot program, several programs and scripts have been developed for: data acquisition, data augmentation, data checking, data pre-processing, model training, and model testing.

The system has been tested in all its parts, both offline and online. In the latter, the overall performance of the individual models slightly decreases due to the greater noise present in real-time data, in which many sources of disturbance are added that might not be present during evaluations with sampled data. Moreover, the performance of the overall system tends to decrease due to the strong interdependencies between the models, where any incorrect output from one is an incorrect input to the next, amplifying the error. The robot program follows the main sequential structure: action recognition, action prediction, and action anticipation.

Of particular appreciation is the performance obtained for the action recognition model, completely developed from scratch, including the training dataset, and based on hand position data provided by MediaPipe. The final model, considered as the best compromise between accuracy and execution efficiency, and therefore extensively tested in real-time, is based on an LSTM network. The robustness and accuracy of this model provided a solid foundation on which to build the rest of the system, as it is at the start of the main dependency sequence. Without the excellent performance of this model, useful predictions of future human actions could not have been obtained, making the entire system unusable. Additionally, among the most interesting results from investigating various technologies to create an action recognition model, is the ability to effectively classify actions based on data from the hand performing the action alone, without the need for data on the auxiliary hand. This leaves room for future implementations of two-independent-handed models, both for action recognition and prediction, which has not been investigated in the present study.

Regarding action prediction, both the model trained on a hand-built sequence variations dataset and the one trained on real sequences dataset have shown good results. As expected, the first managed to achieve better results, showing greater robustness, but with a slight difference from the other model. The most interesting result for action prediction concerns precisely the model trained on real sequences dataset. Although the results obtained with this model are less efficient, what has been achieved is a collaborative system with no prior knowledge of the assembly sequence. This is a very interesting result for application flexibility, especially in the manufacturing industry, where assemblies can change very rapidly and there would be a need to continually re-train the action prediction system. Moreover, it should be noted that the real sequences dataset used contains a fairly small number of sample sequences, specifically 20. Increasing the size of the dataset can significantly improve the model's performance. Moreover, it is necessary to take into account the execution of the assembly sequence by the human operator alone for a number of times that can be considerable, so the solution is not suitable for contexts where the speed of implementation of the collaborative system is preferred. An interesting option, which includes the integration of user feedback, will be discussed in the next section.

5.1 Future work

While the developed system shows promising results, several avenues for future enhancement and research remain. Below are some of the most interesting potential developments to enhance the system's performance and capabilities:

- Dataset enrichment: as highlighted in the discussion, all the datasets built to train the models are modest in size due to limited time and resources. This is especially true for the fine-tuning of YOLO on the application-specific object dataset and for the real sequences dataset used to train the action prediction model in one of the two investigated modes. These datasets have been sufficient to obtain models with good, but not optimal, performance, especially in the real-time implementation. Enriching the dataset would certainly bring benefits to the model's performance in both cases. This is particularly true for the YOLO fine-tuning dataset, which is quite small compared to the sample quantities indicated in the guidelines [110]. The action prediction model trained with the real sequences dataset would also benefit from dataset enrichment, but due to the way it needs to be enriched, it is believed that there are better methods to improve its performance, as described in the next point.

  Generally, enriching the datasets, including the one for the action recognition model, already excellent for the specific test workstation, would allow extending the system's applicability to a wider range of assembly tasks and environments, ensuring scalability and adaptability.

- User feedback integration: as previously highlighted, achieving good performance in the autonomous real-time system implementing the action prediction model trained on the real sequences dataset effectively creates a collaborative system with no prior knowledge of the assembly sequence. However, since the real sequences dataset is constructed by requiring the human operator to execute the assembly sequence alone many times, this method is slow and time-consuming to reach the actual start of human-robot collaboration. The best method to quickly achieve optimal and flexible human-robot collaboration performance for different assemblies could be starting with an initial training based on variations of a hand-built sequence, providing prior knowledge to the model, specific to the component being assembled, and then integrating direct human feedback to refine and improve the system's performance. The goal is to transform the action prediction model, which in this study was an MTL classifier, into a reinforcement learning (RL) model, or more precisely, reinforcement learning from human feedback (RLHF), still implementing an MTL structure since the sequences are constructed as action-object pairs. To do this, a reward function needs to be defined for pre-training the model on the hand-built sequence variations, providing a higher score when the model correctly predicts the following actions and objects in the sequence. This can be manually designed to reflect human preferences or obtained through inverse reinforcement learning (IRL) algorithms, which extract a reward function given observed, optimal behavior [114]. Subsequently, user feedback can be integrated by providing a direct reward score to the model during the execution of assembly sequences in collaboration with the robot. In this way, the system learns from human feedback, progressively improving its performance.

- Two-independent-handed model: it has been suggested during the discussion how the action recognition model has been able to effectively classify actions based on landmarks data from the hand performing the action alone, without needing data on the auxiliary hand. Since the developed object tracker also works by distinguishing the two hands while tracking the objects manipulated by each, it is possible to create a two-independent-handed system, paving the way for very interesting developments in collaborative assembly. For example, in actions such as screwing, which requires the auxiliary hand to hold a component while the main hand performs the actual action, the robot can take on the role of the auxiliary hand, providing a stable hold on the piece and reducing the physical workload of the human operator. In addition, this allows both hands to be used for component delivery without forcing the interruption of the current action being performed by the right hand, enhancing the naturalness of human assembly operations.

- Introducing Large Multimodal Models (LMMs) for YOLO auto-training: as highlighted during the discussion, the most time-consuming and least generalizable process, that separates from the creation of a fully flexible collaborative system applicable to any component to be assembled, is the training or fine-tuning of the object detector, which in this case is YOLO. This requires building a dataset consisting of a significant number of manually labeled images of application-specific objects, i.e., the parts to be assembled and the necessary tools. In Sec. 3.3.4, the initial idea was expressed to use the Segment Anything Model (SAM) [108], so that the robot program could autonomously segment objects affected by actions recognized by the action recognition model. In this way, creating the custom dataset for YOLO training would require much less effort since the objects would already be segmented and would only need to be manually labeled with the correct class. Refer to the relevant section for the discussion on why the idea was discarded. Among the problems encountered, the main one is that YOLO, to be properly trained, needs all the objects to be identified within the images to be labeled, not just some of them. In the same section, the ways to recognize and sample a picked object, once the pick action is classified by the action recognition model, have been explained, considering the joint use of the developed object tracker and SAM. At this point, the task remains to search for similar objects scattered across the scene in all other images, without a precise reference given by hand interaction. The only hypothesized way is to introduce an LMM that, given the sampled objects, can extrapolate a description and use this to search for similar objects in all other images, through a cyclic process.

In the original SAM case study [108], text prompts to extract specific segmented objects from the image are explored, modifying the training procedure to make it text-aware, but the capability is not released in the public project. An attempt to implement text prompts has been made on Fast Segment Anything (FastSAM) [115], based on SAM but developed by another team, where the text prompt processing is based on CLIP [116], which is a neural network from OpenAI to predict the most relevant text snippet given an image. However, from what has been tested, the usability and performance of the FastSAM model with text prompts are not suitable for the purpose, with often incorrect or too generic detections. Along the same lines of tests, better results have been obtained using ChatGPT-4o [117], leveraging its broader knowledge base, but there is certainly room for improvement that could come from a model designed specifically for the task.

From the approximate localization of objects in images carried out by an LMM, accurate bounding boxes and labels can then be created through a modified and automated version of the image labeling tool developed based on SAM [109], thus creating a YOLO auto-training system. Although this goal is very ambitious, it is emphasized that the individual tools to achieve such a system are available, and it requires developing a system that connects them. Achieving this objective would mean having a collaborative assembly system that can be used without any prior knowledge of the component to be assembled but learns from some real demonstrations by the human, just as a human would. The same concept could expand to many other fields beyond assembly and, in general, beyond the manufacturing industry.

5.2 Final remarks

This work demonstrates the potential of advanced computer vision and machine learning techniques in enhancing human-robot collaboration. By enabling cobots to recognize, predict, and anticipate human actions, significant progress is made towards achieving more intuitive and efficient collaborative environments for the manufacturing industry.

The results achieved represent a significant step forward in making human-robot interaction more natural and intuitive, which is a prerequisite for building a society where humans and robots collaborate safely and productively in every scenario, not only in manufacturing.

Bibliography

[1] Eloise Matheson, Riccardo Minto, Emanuele GG Zampieri, Maurizio Faccio, and Giulio Rosati. Human–robot collaboration in manufacturing applications: A review. *Robotics*, 8(4):100, 2019.

[2] Muhammad Awais. *Intuitive Human-Robot Interaction by Intention Recognition*. PhD thesis, Universität Bayreuth, 2013.

[3] Yanan Li and Shuzhi Sam Ge. Human–Robot Collaboration Based on Motion Intention Estimation. *IEEE/ASME Transactions on Mechatronics*, 19(3):1007–1014, 2013.

[4] Kristen Stubbs, Pamela J Hinds, and David Wettergreen. Autonomy and common ground in human-robot interaction: A field study. *IEEE Intelligent Systems*, 22(2):42–50, 2007.

[5] Amos Freedy, Ewart DeVisser, Gershon Weltman, and Nicole Coeyman. Measurement of trust in human-robot collaboration. In *2007 International symposium on collaborative technologies and systems*, pages 106–114. Ieee, 2007.

[6] Pablo Segura, Odette Lobato-Calleros, Alejandro Ramírez-Serrano, and Eduardo Gamaliel Hernández-Martínez. Safety assurance in human-robot collaborative systems: A survey in the manufacturing industry. *Procedia CIRP*, 107:740–745, 2022.

[7] Hang Su, Chenguang Yang, Giancarlo Ferrigno, and Elena De Momi. Improved human–robot collaborative control of redundant robot for teleoperated minimally invasive surgery. *IEEE Robotics and Automation Letters*, 4(2):1447–1453, 2019.

[8] Cynthia Breazeal. Social interactions in HRI: the robot view. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 34(2):181–186, 2004.

[9] Prasad Akella, Michael Peshkin, ED Colgate, Witaya Wannasuphoprasit, Nidamaluri Nagesh, Jim Wells, Steve Holland, Tom Pearson, and Brian Pea-

cock. Cobots for the automobile assembly line. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 1, pages 728–733. IEEE, 1999.

[10] Nicole Robinson, Brendan Tidd, Dylan Campbell, Dana Kulić, and Peter Corke. Robotic vision for human-robot interaction and collaboration: A survey and systematic review. *ACM Transactions on Human-Robot Interaction*, 12(1):1–66, 2023.

[11] Zehua Sun, Qiuhong Ke, Hossein Rahmani, Mohammed Bennamoun, Gang Wang, and Jun Liu. Human action recognition from various data modalities: A review. *IEEE transactions on pattern analysis and machine intelligence*, 45(3):3200–3225, 2022.

[12] Dare A Baldwin and Jodie A Baird. Discerning intentions in dynamic human action. *Trends in cognitive sciences*, 5(4):171–178, 2001.

[13] Mohammad Safeea, Pedro Neto, and Richard Bearee. On-line collision avoidance for collaborative robot manipulators by adjusting off-line generated paths: An industrial use case. *Robotics and Autonomous Systems*, 119:278–288, 2019.

[14] Clint Heyer. Human-robot interaction and future industrial robotics applications. In *2010 ieee/rsj international conference on intelligent robots and systems*, pages 4749–4754. IEEE, 2010.

[15] ISO Central Secretary. Robots and robotic devices – Safety requirements for industrial robots. Standard ISO 10218-1/2:2011, International Organization for Standardization, Geneva, CH, 2011.

[16] Steffen Landscheidt and Mirka Kans. Method for assessing the total cost of ownership of industrial robots. *Procedia Cirp*, 57:746–751, 2016.

[17] Fahad Sherwani, Muhammad Mujtaba Asad, and Babul Salam Kader K Ibrahim. Collaborative robots and industrial revolution 4.0 (ir 4.0). In *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, pages 1–5. IEEE, 2020.

[18] Mitchell M. Tseng, Yue Wang, and Roger J. Jiao. Mass Customization. In The International Academy for Produ, Luc Laperrière, and Gunther Reinhart, editors, *CIRP Encyclopedia of Production Engineering*, Berlin, Heidelberg, 2017.

[19] Jeffrey Too Chuan Tan, Feng Duan, Ye Zhang, Kei Watanabe, Ryu Kato, and Tamio Arai. Human-robot collaboration in cellular manufacturing: Design and

development. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 29–34. IEEE, 2009.

[20] Holly A Yanco and Jill Drury. Classifying human-robot interaction: an updated taxonomy. In *2004 IEEE international conference on systems, man and cybernetics (IEEE Cat. No. 04CH37583)*, volume 3, pages 2841–2846. IEEE, 2004.

[21] Alessandro De Luca and Fabrizio Flacco. Integrated control for pHRI: Collision avoidance, detection, reaction and collaboration. In *2012 4th IEEE RAS & EMBS international conference on biomedical robotics and biomechatronics (BioRob)*, pages 288–295. IEEE, 2012.

[22] Peter A Hancock, Deborah R Billings, Kristin E Schaefer, Jessie YC Chen, Ewart J De Visser, and Raja Parasuraman. A Meta-Analysis of Factors Affecting Trust in Human-Robot Interaction. *Human factors*, 53(5):517–527, 2011.

[23] Nicolas Padoy and Gregory D Hager. Human-machine collaborative surgery using learned models. In *2011 IEEE International Conference on Robotics and Automation*, pages 5285–5292. IEEE, 2011.

[24] Jur Van Den Berg, Stephen Miller, Daniel Duckworth, Humphrey Hu, Andrew Wan, Xiao-Yu Fu, Ken Goldberg, and Pieter Abbeel. Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations. In *2010 IEEE International Conference on Robotics and Automation*, pages 2074–2081. IEEE, 2010.

[25] Giorgia Chiriatti, Luca Carbonari, Maria Gabriella Ceravolo, Elisa Andrenelli, Marzia Millevolte, and Giacomo Palmieri. A Robot-Assisted Framework for Rehabilitation Practices: Implementation and Experimental Results. *Sensors*, 23(17):7652, 2023.

[26] Rubén Mitnik, Matías Recabarren, Miguel Nussbaum, and Alvaro Soto. Collaborative robotic instruction: A graph teaching experience. *Computers & Education*, 53(2):330–342, 2009.

[27] Karen N Gregorczyk, Leif Hasselquist, Jeffrey M Schiffman, Carolyn K Bensel, John P Obusek, and David J Gutekunst. Effects of a lower-body exoskeleton device on metabolic cost and gait biomechanics during load carriage. *Ergonomics*, 53(10):1263–1275, 2010.

[28] Xi Vincent Wang, Zsolt Kemény, József Váncza, and Lihui Wang. Human–robot collaborative assembly in cyber-physical production: Classification framework and implementation. *CIRP annals*, 66(1):5–8, 2017.

[29] Dario Antonelli, Seygey Astanin, Maurizio Galetto, and Luca Mastrogiacomo. Training by demonstration for welding robots by optical trajectory tracking. *Procedia Cirp*, 12:145–150, 2013.

[30] Mike Phillips and Maxim Likhachev. SIPP: Safe Interval Path Planning for Dynamic Environments. In *2011 IEEE international conference on robotics and automation*, pages 5628–5635. IEEE, 2011.

[31] Shen Li and Julie A Shah. Safe and Efficient High Dimensional Motion Planning in Space-Time with Time Parameterized Prediction. In *2019 international conference on robotics and automation (ICRA)*, pages 5012–5018. IEEE, 2019.

[32] Christian Vogel, Christoph Walter, and Norbert Elkmann. A projection-based sensor system for safe physical human-robot collaboration. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5359–5364. IEEE, 2013.

[33] Bram Vanderborght, Alin Albu-Schäffer, Antonio Bicchi, Etienne Burdet, Darwin G Caldwell, Raffaella Carloni, Manuel Catalano, Oliver Eiberger, Werner Friedl, Ganesh Ganesh, et al. Variable impedance actuators: A review. *Robotics and autonomous systems*, 61(12):1601–1614, 2013.

[34] ISO Central Secretary, IEC Central Office. Safety aspects – Guidelines for their inclusion in standards. Guide ISO/IEC 51:2014, International Organization for Standardization, International Electrotechnical Commission, Geneva, CH, 2014.

[35] ISO Central Secretary. Robots and robotic devices – Collaborative robots. Standard ISO/TS 15066:2016, International Organization for Standardization, Geneva, CH, 2016.

[36] Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55:248–266, 2018.

[37] IEC Central Office. Safety of machinery – Electrical equipment of machines – Part 1: General requirements. Standard IEC 60204-1:2016, International Electrotechnical Commission, Geneva, CH, 2016.

[38] Martin J Rosenstrauch and Jörg Krüger. Safe human-robot-collaboration-introduction and experiment using ISO/TS 15066. In *2017 3rd International conference on control, automation and robotics (ICCAR)*, pages 740–744. IEEE, 2017.

[39] Sami Haddadin, Alin Albu-Schäffer, and Gerd Hirzinger. Requirements for safe robots: Measurements, analysis and new insights. *The International Journal of Robotics Research*, 28(11-12):1507–1527, 2009.

[40] Niccolò Lucci, Bakir Lacevic, Andrea Maria Zanchettin, and Paolo Rocco. Combining Speed and Separation Monitoring With Power and Force Limiting for Safe Collaborative Robotics Applications. *IEEE Robotics and Automation Letters*, 5(4):6121–6128, 2020.

[41] ISO Central Secretary. Safety of machinery – General principles for design – Risk assessment and risk reduction. Standard ISO 12100:2010, International Organization for Standardization, Geneva, CH, 2010.

[42] Heonseop Shin, Sanghoon Kim, Kwang Seo, and Sungsoo Rhim. A Real-Time Human-Robot Collision Safety Evaluation Method for Collaborative Robot. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 509–513. IEEE, 2019.

[43] Roland Behrens and Norbert Elkmann. A Revised Framework for Managing the Complexity of Contact Hazards in Collaborative Robotics. In *2021 IEEE International Conference on Intelligence and Safety for Robotics (ISR)*, pages 252–258. IEEE, 2021.

[44] Tiago Rodrigues De Almeida, Andrey Rudenko, Tim Schreiter, Yufei Zhu, Eduardo Gutierrez Maestro, Lucas Morillo-Mendez, Tomasz P Kucner, Oscar Martinez Mozos, Martin Magnusson, Luigi Palmieri, et al. THOR-Magni: Comparative Analysis of Deep Learning Models for Role-Conditioned Human Motion Prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2200–2209, 2023.

[45] Laura Duarte and Pedro Neto. Classification of primitive manufacturing tasks from filtered event data. *Journal of Manufacturing Systems*, 68:12–24, 2023.

[46] Ronal Singh, Tim Miller, Joshua Newn, Eduardo Velloso, Frank Vetere, and Liz Sonenberg. Combining gaze and AI planning for online human intention recognition. *Artificial Intelligence*, 284:103275, 2020.

[47] Qiyue Wang, Wenhua Jiao, Rui Yu, Michael T Johnson, and YuMing Zhang. Virtual reality robot-assisted welding based on human intention recognition. *IEEE Transactions on Automation Science and Engineering*, 17(2):799–808, 2019.

[48] Zequn Zhang, Yuchen Ji, Dunbing Tang, Jie Chen, and Changchun Liu. Enabling collaborative assembly between humans and robots using a digital twin system. *Robotics and Computer-Integrated Manufacturing*, 86:102691, 2024.

[49] Mélodie Daniel. *Optimizing decision-making for human-robot collaboration*. PhD thesis, Université Clermont Auvergne, 2022.

[50] Samsu Sempena, Nur Ulfa Maulidevi, and Peb Ruswono Aryan. Human Action Recognition Using Dynamic Time Warping. In *Proceedings of the 2011 international conference on electrical engineering and informatics*, pages 1–5. IEEE, 2011.

[51] Lasitha Piyathilaka and Sarath Kodagoda. Gaussian Mixture Based HMM for Human Daily Activity Recognition Using 3D Skeleton Features. In *2013 IEEE 8th conference on industrial electronics and applications (ICIEA)*, pages 567–572. IEEE, 2013.

[52] Seol-Hyun Noh. Analysis of Gradient Vanishing of RNNs and Performance Comparison. *Information*, 12(11):442, 2021.

[53] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent Advances in Recurrent Neural Networks. *arXiv preprint arXiv:1801.01078*, 2017.

[54] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural computation*, 9(8):1735–1780, 1997.

[55] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *Advances in neural information processing systems*, 30, 2017.

[57] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE transactions on neural networks and learning systems*, 33(12):6999–7019, 2021.

[58] Mona Fathollahi Ghezelghieh, Rangachar Kasturi, and Sudeep Sarkar. Learning camera viewpoint using CNN to improve 3D body pose estimation. In *2016 fourth international conference on 3D vision (3DV)*, pages 685–693. IEEE, 2016.

[59] Konrad Gadzicki, Razieh Khamsehashari, and Christoph Zetzsche. Early vs Late Fusion in Multimodal Convolutional Neural Networks. In *2020 IEEE 23rd international conference on information fusion (FUSION)*, pages 1–6. IEEE, 2020.

[60] Cees GM Snoek, Marcel Worring, and Arnold WM Smeulders. Early versus late fusion in semantic video analysis. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 399–402, 2005.

[61] Reem Alfaifi and Abdel Monim Artoli. Human action prediction with 3D-CNN. *SN Computer Science*, 1(5):286, 2020.

[62] Hema S Koppula and Ashutosh Saxena. Anticipating Human Activities Using Object Affordances for Reactive Robotic Response. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):14–29, 2015.

[63] Liangchen Song, Gang Yu, Junsong Yuan, and Zicheng Liu. Human pose estimation and its application to action recognition: A survey. *Journal of Visual Communication and Image Representation*, 76:103055, 2021.

[64] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. MediaPipe: A Framework for Building Perception Pipelines. *arXiv preprint arXiv:1906.08172*, 2019.

[65] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. MediaPipe Hands: On-device Real-time Hand Tracking. *arXiv preprint arXiv:2006.10214*, 2020.

[66] Moritz Kassner, William Patera, and Andreas Bulling. Pupil: An Open Source Platform for Pervasive Eye Tracking and Mobile Gaze-based Interaction. In *Proceedings of the 2014 ACM international joint conference on pervasive and ubiquitous computing: Adjunct publication*, pages 1151–1160, 2014.

[67] Vaibhav V Unhelkar, Przemyslaw A Lasota, Quirin Tyroller, Rares-Darius Buhai, Laurie Marceau, Barbara Deml, and Julie A Shah. Human-Aware Robotic Assistant for Collaborative Assembly: Integrating Human Motion Prediction With Planning in Time. *IEEE Robotics and Automation Letters*, 3(3):2394–2401, 2018.

[68] Claudia Pérez-D'Arpino and Julie A Shah. Fast Target Prediction of Human Reaching Motion for Cooperative Human-Robot Manipulation Tasks using Time Series Classification. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 6175–6182. IEEE, 2015.

[69] Jianjing Zhang, Hongyi Liu, Qing Chang, Lihui Wang, and Robert X Gao. Recurrent neural network for motion trajectory prediction in human-robot collaborative assembly. *CIRP annals*, 69(1):9–12, 2020.

[70] Kitani, Kris M and Ziebart, Brian D and Bagnell, James Andrew and Hebert, Martial. Activity forecasting. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part IV 12*, pages 201–214. Springer, 2012.

[71] Kelsey P Hawkins, Nam Vo, Shray Bansal, and Aaron F Bobick. Probabilistic human action prediction and wait-sensitive planning for responsive human-robot collaboration. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 499–506. IEEE, 2013.

[72] Zhujun Zhang, Gaoliang Peng, Weitian Wang, Yi Chen, Yunyi Jia, and Shaohui Liu. Prediction-Based Human-Robot Collaboration in Assembly Tasks Using a Learning from Demonstration Model. *Sensors*, 22(11):4279, 2022.

[73] Paul Schydlo, Mirko Rakovic, Lorenzo Jamone, and José Santos-Victor. Anticipation in Human-Robot Cooperation: A Recurrent Neural Network Approach for Multiple Action Sequences Prediction. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5909–5914. IEEE, 2018.

[74] Chien-Ming Huang and Bilge Mutlu. Anticipatory robot control for efficient human-robot collaboration. In *2016 11th ACM/IEEE international conference on human-robot interaction (HRI)*, pages 83–90. IEEE, 2016.

[75] Enrique Vidal, Franck Thollard, Colin De La Higuera, Francisco Casacuberta, and Rafael C Carrasco. Probabilistic finite-state machines-part I. *IEEE transactions on pattern analysis and machine intelligence*, 27(7):1013–1025, 2005.

[76] Muhammad Awais and Dominik Henrich. Human-robot collaboration by intention recognition using probabilistic state machines. In *19th International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD 2010)*, pages 75–80. IEEE, 2010.

[77] Ayano Hiranaka, Minjune Hwang, Sharon Lee, Chen Wang, Li Fei-Fei, Jiajun Wu, and Ruohan Zhang. Primitive Skill-based Robot Learning from Human Evaluative Feedback. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7817–7824. IEEE, 2023.

[78] Soroush Nasiriany, Huihan Liu, and Yuke Zhu. Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7477–7484. IEEE, 2022.

[79] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[80] Disa Alexandra Queiroz Palma. Enhancing Indoor Human Detection: A Comprehensive Study of YOLOv5 Algorithm with Thermal Imagery. Master's thesis, University of Coimbra, 2023.

[81] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, et al. YOLOv6: A single-stage object detection framework for industrial applications. *arXiv preprint arXiv:2209.02976*, 2022.

[82] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. *arXiv preprint arXiv:2402.13616*, 2024.

[83] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *Acm computing surveys (CSUR)*, 38(4):13–es, 2006.

[84] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, and Tae-Kyun Kim. Multiple object tracking: A literature review. *Artificial intelligence*, 293:103448, 2021.

[85] Laura Leal-Taixé, Anton Milan, Konrad Schindler, Daniel Cremers, Ian Reid, and Stefan Roth. Tracking the Trackers: An Analysis of the State of the Art in Multiple Object Tracking. *arXiv preprint arXiv:1704.02781*, 2017.

[86] Nir Aharon, Roy Orfaig, and Ben-Zion Bobrovsky. BoT-SORT: Robust Associations Multi-Pedestrian Tracking. *arXiv preprint arXiv:2206.14651*, 2022.

[87] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. Bytetrack: Multi-object tracking by associating every detection box. In *European conference on computer vision*, pages 1–21. Springer, 2022.

[88] Anthony Quenehen, Jérôme Pocachard, and Nathalie Klement. Process optimisation using collaborative robots-comparative case study. *Ifac-papersonline*, 52(13):60–65, 2019.

[89] Luca Gualtieri, Federico Fraboni, Matteo De Marchi, and Erwin Rauch. Development and evaluation of design guidelines for cognitive ergonomics in human-robot collaborative assembly systems. *Applied ergonomics*, 104:103807, 2022.

[90] Gregory Flandin, François Chaumette, and Eric Marchand. Eye-in-hand/Eye-to-hand Cooperation for Visual Servoing. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 3, pages 2741–2746. IEEE, 2000.

[91] Giorgio Zoppi. GitHub - human-action-recognition-and-anticipation. `https://github.com/grgzpp/human-action-recognition-and-anticipation`, 2024. Accessed: 2024-06-18.

[92] Google. MediaPipe - Hand landmarks detection guide. `https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker`, 2024. Accessed: 2024-04-18.

[93] Yuhan Bai. RELU-function and derived function review. In *SHS Web of Conferences*, volume 144, page 02006. EDP Sciences, 2022.

[94] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[95] Hedvig Kjellström, Javier Romero, and Danica Kragić. Visual object-action recognition: Inferring object affordances from human demonstration. *Computer Vision and Image Understanding*, 115(1):81–90, 2011.

[96] Fadime Sener, Dibyadip Chatterjee, Daniel Shelepov, Kun He, Dipika Singhania, Robert Wang, and Angela Yao. Assembly101: A Large-Scale Multi-View Video Dataset for Understanding Procedural Activities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21096–21106, 2022.

[97] Fadime Sener, Dipika Singhania, and Angela Yao. Temporal Aggregate Representations for Long-Range Video Understanding. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pages 154–171. Springer, 2020.

[98] Lin Song, Shiwei Zhang, Gang Yu, and Hongbin Sun. TACNet: Transition-Aware Context Network for Spatio-Temporal Action Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11987–11995, 2019.

[99] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I Jordan. How Does Learning Rate Decay Help Modern Neural Networks? *arXiv preprint arXiv:1908.01878*, 2019.

[100] Matthew Inkawhich. PyTorch Tutorials - Saving and Loading Models. `https://pytorch.org/tutorials/beginner/saving_loading_models`, 2024. Accessed: 2024-04-02.

[101] Yu Zhang and Qiang Yang. A Survey on Multi-Task Learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609, 2021.

[102] Michael McCloskey and Neal J Cohen. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[103] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018.

[104] Yongjun Li, Shasha Li, Haohao Du, Lijia Chen, Dongming Zhang, and Yao Li. YOLO-ACN: Focusing on Small Target and Occluded Object Detection. *IEEE access*, 8:227288–227303, 2020.

[105] MathWorks. MATLAB - Computer Vision Toolbox. `https://www.mathworks.com/products/computer-vision`, 2024. Accessed: 2024-05-31.

[106] M. Safeea and P. Neto. Model-based hardware in the loop control of collaborative robots: Simulink and Python based interfaces. *International Journal of Computer Integrated Manufacturing*, 0(0):1–13, 2023.

[107] Ultralytics. Ultralytics - Object Detection Datasets Overview. `https://docs.ultralytics.com/datasets/detect`, 2024. Accessed: 2024-04-29.

[108] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment Anything. *arXiv:2304.02643*, 2023.

[109] Giorgio Zoppi. GitHub - sam-yolo-image-labeling-tool. `https://github.com/grgzpp/sam-yolo-image-labeling-tool`, 2024. Accessed: 2024-06-18.

[110] Ultralytics. Ultralytics - Tips for Best Training Results. `https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results`, 2024. Accessed: 2024-05-08.

[111] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context, 2015.

[112] Giorgio Zoppi. GitHub - yolo-mp-object-tracker. `https://github.com/grgzpp/yolo-mp-object-tracker`, 2024. Accessed: 2024-06-18.

[113] Rafael Padilla, Sergio L Netto, and Eduardo AB Da Silva. A Survey on Performance Metrics for Object-Detection Algorithms. In *2020 international conference on systems, signals and image processing (IWSSIP)*, pages 237–242. IEEE, 2020.

[114] Andrew Y Ng, Stuart Russell, et al. Algorithms for Inverse Reinforcement Learning. In *Icml*, volume 1, page 2, 2000.

[115] Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast Segment Anything. *arXiv preprint arXiv:2306.12156*, 2023.

[116] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning Transferable Visual Models From Natural Language Supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[117] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language Models are Few-Shot Learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.