POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Trajectory optimization for collaborative robotics applications

Laurea Magistrale in Automation and Control Engineering - Ingegneria Industriale

Author: Roberto La Commare

Advisor: Prof. Andrea Maria Zanchettin

Co-advisor: Prof. Paolo Rocco Ing. Niccolò Lucci

Academic year: 2020-2021

1. Introduction

Today, the field of collaborative robotics is undergoing considerable development. A growing demand for new installations has gone hand in hand with an increasing focus on safety and comfort of the operator that works with the cobot. Considering the distinctive features of cobots compared to ordinary robots, they are expected to lead the way to mass customization, decrease space requirements, increase product quality, production efficiency, and improve working conditions for humans, both safety and ergonomics aspects

2. Thesis purpose

The purpose of the thesis is to develop an effective and repeatable method to optimize the trajectory of a cobot before its deployment in a work cell, thus without the need of additional sensors in the final setup. The trajectory must be optimized with respect to execution time and operator safety, avoiding any obstacles in the path.
This is linked to the idea of virtual commissioning, i.e. the development of the entire work cell in simulation before it is developed in industry. For this reason, in order to acquire the operator's movement data, it is not possible to physically go on site and take measurements. Therefore, in line with the digital industry 4.0, it is necessary to reproduce the work cell with the help of virtual reality, making an operator wear the device in a controlled environment and acquire the motion data. Finally the cobot trajectory can be optimized, taking into account the motion data collected.

3. Optimization Algorithm

The optimization algorithm uses simulations outsourced to an ad-hoc program: RoboDK [1]. It is a powerful simulator for industrial robots. It has an extensive library of robot arms from over 40 different robot manufacturers, making it possible to simulate any robot controller.
The optimization algorithm run on MATLAB and by calling a Python function it is possible to run a simulation via RoboDK API.
The general idea is to optimize a pick and place task, being one of the most versatile and common functions we can find in robotics. The *start* and *end* configurations of the task depends on the position of the robot in the working area so they are not modifiable. The trajectory will be optimized by modifying an intermediate point as it is depicted in Figure 1. The optimization will look for the intermediate configuration that

results in the shortest task time, in other words the cost function is the working time and the optimization variable is an intermediate configuration.
The cost function takes into account the distance between the robot and the operator and the environmental obstacles. In total the optimization will look for three balancing objectives: minimize time, maximize safety and avoid obstacles.

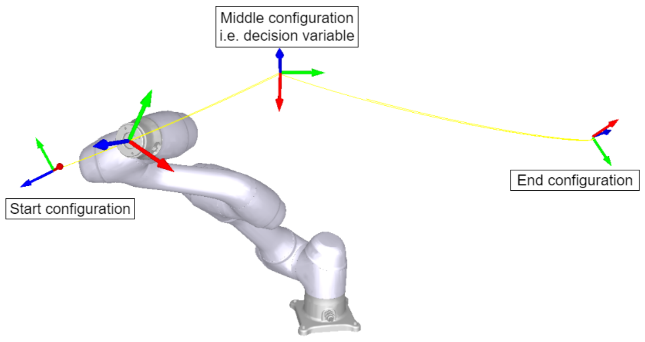


Figure 1: Visualization of the simulated cobot trajectory, along with the start and end fixed configuration and the middle one, which is the decision variable of the optimization.

3.1. Cost function

The cost function is the core of the optimization algorithm. Depending on how this function is defined it will determine where the robot's trajectory will converge.
Once the sequence of configurations q taken by the robot has been simulated, the positions of each link are found by solving the direct kinematic problem. At this point it is possible to calculate the segment-to segment distance [2] between robot and obstacles.
In order to keep the robot-path away from the obstacle a method similar to artificial potential was used. A penalty in the cost function is added if any of the robot link are close to any obstacle below a threshold. This penalty is inversely proportional to the distance.
If the robot gets too close to an obstacle, the cost will increase rapidly, making the trajectory converge to a safer distance

3.2. Velocity management

The speeds of the movements are defined according to the operator's distance. Essentially the closer the robot gets to the human the slower the robot can go. To improve the capability and the overall result the trajectory is discretized in n concatenated moves, each of which can take a different speed. Inside the simulation algorithm the minimum distance between the person and the robot links is computed once for each trajectory sub-segment $i = 1, 2, ..., n$. This distance is then used to define the speed of each sub-segment.
The maximum speed will be proportional to the distance according the following equation

$$v^i_{max} = \frac{1}{T_B} k d^i_{min} \tag{1}$$

where $v^i_{max}[mm/s]$ is the maximum velocity of the i^{th} sub-segment. The value of $d^i_{min}[mm]$ is the minimum distance calculated between the robot configurations in the i^{th} sub-segment and the human. The time $T_B[s]$ is computed from the sum of the reaction time of the controller and braking time of the robot, it is generally provided by the robot manufacturer. This kind of reasoning is generated by the speed Separation Monitoring Algorithm suggested by the ISO-TS-15066 [3] with a main difference: the ISO standard requires the presence of sensors in the field that measure the actual distance of the operator from the base of the robot in real time. In our scenario instead there is only a probabilistic study: there is no certainty that the person is not in proximity of the cobot, but only a high probability. An unintentional collision is more probable with respect to the SSM. For this reason the safety scaling factor $k < 1$ is added to make a possible accident more acceptable, reducing the velocity with respect to the SSM approach.
The safety scaling factor k must be chosen both on the basis of the reliability of the movements of the person and the level of risk and damage that can cause an unintentional contact.

4. Data acquisition: Virtual Reality and Kinect sensor

In this Section we are going to analyse the data acquisition process, which has the goal of obtaining the operator's occupancy area in the workplace in the form of a point-cloud.
Since one of the aims of the thesis is to prove the concept of virtual commissioning, the real plant

was not used (since it is hypothetically not yet in place) but virtual reality was implemented.
This type of approach has multiple benefits:

- it allows a preliminary study of the tasks the operators movement before the actual implementation of the cell;
- it does not require stopping or slowing down industry production, which is always economically problematic;
- it can be performed in any given location, and requires only the operator and two devices (a virtual reality visor and a depth camera).

An executable Android application was developed using Unity [4]. The procedure involves drawing the scene of the industry as if it were a video game, inserting the machinery and other necessary 3D elements. Then an Android application is generated from Unity and directly loaded into the Facebook Oculus Quest2 [5] via USB-C cable. This allowed the operator to simulate a complete working scene without being restricted by a cable.
A human-tracking device was required to record

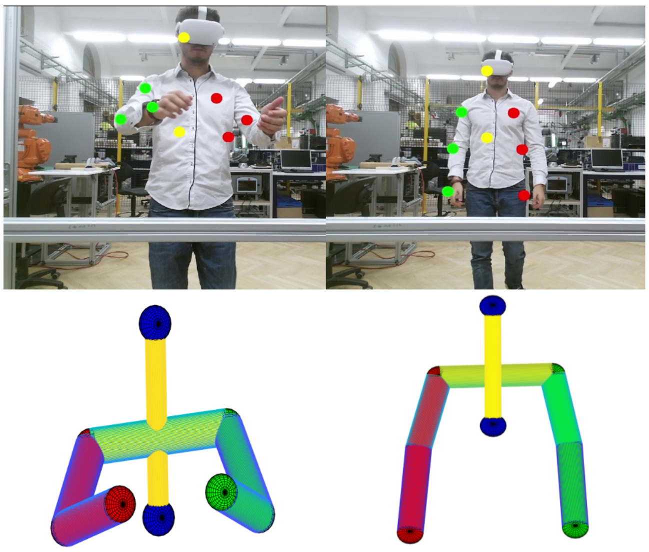


Figure 2: Kinect acquisition output and Matlab representation of the operator

the data on the movement of the worker. The Kinect [6] is a device developed by Microsoft® for motion detection. This device integrates an RGB camera with infrared projectors and detectors that allow to acquire a depth map of the environment. Then, through the random forest algorithm already integrated into the Kinect SDK, the recognition of gestures and the detection of the body skeleton is carried out. We are interested in: head, shoulders, elbows, hands and torso (Figure 2).
Once this phase was completed and the cloud-point of the operator's positions were obtained at one-second intervals, it was possible to start the optimization algorithms.

5. Experimental analysis

5.1. Human model in the optimization algorithm

Three different methods to interpret the human occupancy in the workspace and thus the optimization algorithm has been tested. Depending on the method of calculation of the distances, three distinct methods have been developed:

- ***static human volumetric sweep*** (Figure 3), where the algorithm considers all the positions the operator takes during the entire operation;
- ***dynamic human model*** (Figure 4), where the algorithm considers only the distance between the robot and the operator at the pose that depends on the given instant;
- ***time window human model*** (Figure 5), where the algorithm considers a window of positions at each considered instant.

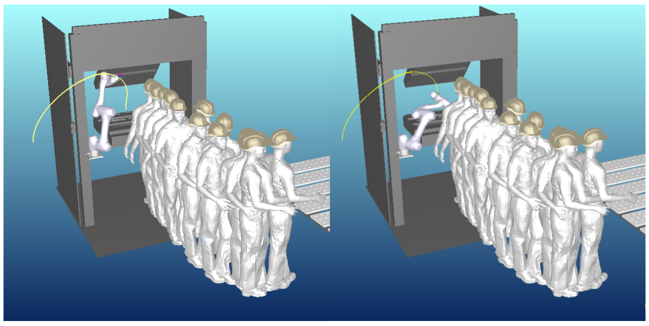


Figure 3: In each point of the robot trajectory the algorithm will consider all the operator's positions.

5.2. Experiment with cobot and operator

The experiments were carried out to verify in real life the quality of the trajectories generated by the three algorithms.
The challenge of this experiment is to faithfully represent the behaviour of an operator in the workplace. In reality, the worker cannot perform

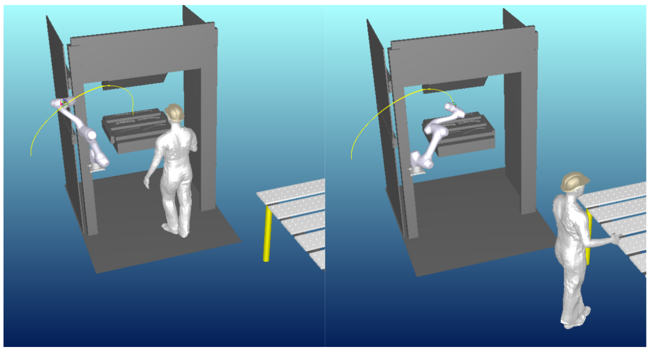


Figure 4: At each segment of the cobot's trajectory, the algorithm computes the distance between the robot and the operator's position at that second. When the operator is far the robot will move at a higher velocity.

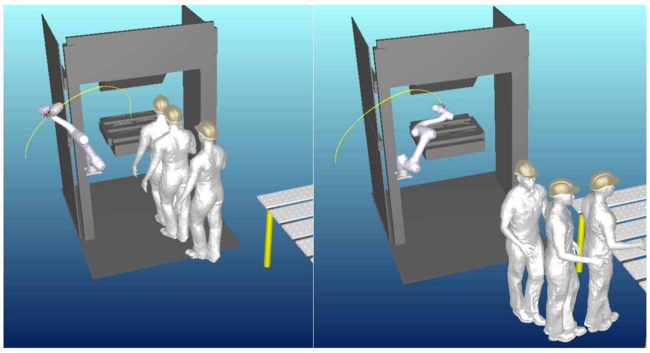


Figure 5: At each segment of the cobot's trajectory, the algorithm calculates the minimum distance between the robot and three operator positions: at that second, one second before and one second after.

all the tasks precisely at the same time, but depending on various factors such as fatigue, boredom, etc., his speed may vary. Trying to reproduce this variability, the experiment was carried out with different operator executions.
In order to understand and quantify the improvement of the three algorithms a basic trajectory was needed. This trajectory provides a benchmark against which the algorithms can be compared. This trajectory is the simplest way to get the robot to move from the starting point to the ending point. For programming this operation the servo-assist training method was used. The speed was maintained at 20deg/s, i.e. the default speed set on the cobot. This trajectory will be named "Base".
In order to compare these 4 trajectories (Base, Algorithm1, Algorithm2, Algorithm3) a standard index must be defined.

The first one will be the cycle time, so the total time in between the opening and the closing of the press.
The second index is the energy transferred in a transient contact with the operator's shoulder.
The last comparing index is the velocity variance admitted by the algorithm, i.e. how much the operator can vary its task time without colliding with the cobot.

5.3. Cycle time

Considering the operator at a standard speed, we will have the following cycle times:

Method	Trajectory Time	Cycle Time
Base	$54s$	$68s$
Alg1	$45s$	$60s$
Alg2	$27.1s$	$52s$
Alg3	$34.7s$	$52s$

Table 1: Comparison between Trajectory times and cycle times, i.e. from opening until closing the press

Those cycle times are directly taken from the experiments. The base trajectory and the first algorithm are particularly slow, so the operator has to wait for the robot to finish its task. This will increase the cycle time and add downtime to the process. This is to be always avoided.
The second and third algorithms have the same cycle time because when the operator returns he can directly start the press without waiting any longer. So in this comparison index the second and third algorithms are equally well performing.

5.4. Transient collision

To define this index, which is correlated with safety, the energy dissipated due to transient contact is considered. This value is usually calculated as part of the Power Force Limiting strategy. This criterion was used because there are no distance sensors as required by the SSM, therefore this task is subject to the PFL regulation (ISO-TS-5066 [3]), as contact could occur. The index is calculated from the formula:

$$E = \frac{1}{2}\mu v_{\text{rel}}^2 \tag{2}$$

where v_{rel} is the relative velocity between the end effector and the operator's shoulder when he is standing in front of the press, k is the elastic constant of the body part, μ is the reduced mass of the two-body system. Finally, it was possible to produce the graphs shown in Figure 6
It can be observed that the worst case is represented by the values:

Method	Maximum impact energy
Base	0.091J
Algorithm1	0.856J
Algorithm2	1.14 J
Algorithm3	1.068J

Table 2: Maximum energy transmitted in the worst case scenario from the cobot to the operator shoulder in a transient collision

Clearly, the first algorithm, which is the most conservative one, has low energy along with an equally low speed. On the other hand, Algorithms 2 and 3, have a higher energy transmission due to the faster movements. However, this increase in danger is necessary in order to have reasonable execution times. In all cases, even if the operator were to find himself close to the robot at the moment of maximum energy exchange, this would represent an acceptable risk according to the FPL criterion. Indeed all the 4 trajectories are well below the ISO threshold of pain $E_{shoulder} = 1.46$ J calculated using the pressure limit associated with the shoulder joint (Table A.2 of the ISO-TS-15066 [3]) in a 1 centimeter square area.

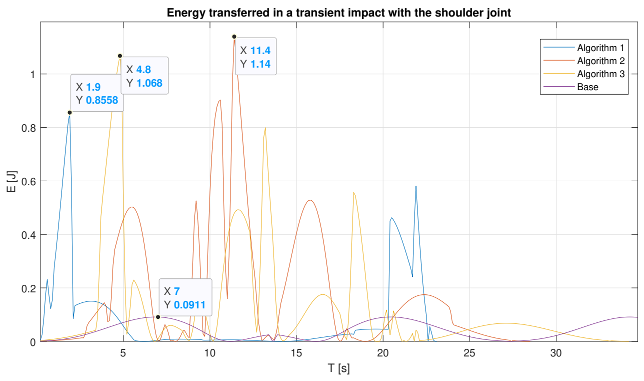


Figure 6: Visualization of the energy that can be transmitted in a transient contact between cobot and operator's shoulder. Note: the Base graph is truncated.

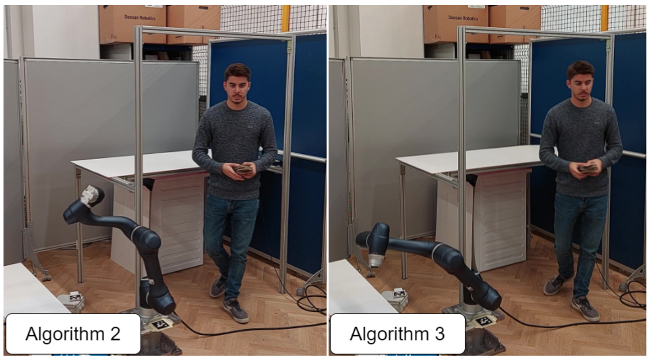


Figure 7: Time frame of the moment when the operator has finished the first task and is moving away from the press. It can be observed the cobot to be nearer the collaborative space in the second algorithm.

5.5. Velocity variance

The last index considers the synchrony of movements, i.e. what is the speed variation that the operator can have before coming into contact with the cobot. Allowing a greater variance is preferred since it results in a more versatile trajectory that can perform well at a range of operator speeds.
For this index it is worthwhile to examine what is the maximum permissible decrease in speed. This is because if the operator slows down too much, the cobot could enter the press while the operator is still finishing the task.
In order to validate this index, it is useful to study slowed-down experiments (Figure 7).
The comparison is made between only the second and third algorithms, as the first and base algorithms have already proved to be irrelevant due to excessive slow motion.

Method	Maximum speed variance
Algorithm2	27%
Algorithm3	42%

Table 3: Maximum speed variance admitted before the Human and robot presence overlap in front of the press

From the experiment it can be seen that the case of the 20% speed reduction is very close to the limit of Algorithm 2. If we study this algebraically we can see that the precise limit is 27%, while for Algorithm 3 is 42 %.
This index reveals the main feature of the third algorithm. In fact, by taking into account a

window of values of the operator's positions, it is able to better accommodate possible changes with respect to standard movements. Also, during the experiments it was found that this algorithm is slightly less aggressive than the second one, giving in some ways a more comfortable feeling to the operator.

6. Conclusions

In conclusion, the static human volumetric sweep algorithm (Algorithm 1) is certainly the most conservative and the safest out of the three, but its overall execution time does not allow it to be considered as a solution applicable in the real world. The second algorithm, with the dynamic human model, is the best performing algorithm from the point of view of trajectory time. However, it has a low variance of the allowed operator's speed, i.e. it only synchronizes well with the human if his/her speed does not deviate too much from the starting acquisition. This results in a higher risk of collision if the operator does not always work at the same speed. Combining this higher probability with higher impact energy than the other algorithms (although only slightly), the lower safety of the solution becomes evident. Finally, the third algorithm, the time window human model, which combines the dynamic approach of the second algorithm with the more conservative approach of the first one, manages to achieve satisfactory results. In particular, the overall trajectory time is well matched to the cycle times of the process in consideration. Indeed, the improved speed variability allows for improved collaboration with the operator.

References

[1] "Simulate robot applications with robodk." [Online]. Available: https://robodk.com/

[2] D. S. G. Algorithms, "Minimum distance between two line segments in 3d." [Online]. Available: https://www.it-swarm.it/it/algorithm/calcolo-della-distanza-piu-breve-tra-due-/linee-segmenti-di-linea-3d/957789635/

[3] O. for Standardization: Switzerland, "Iso 13849-1:2015 safety of machinery – safety related parts of control systems – part 1: General principles for design," 2015.

[4] "Unity (motore grafico)," Sep 2021. [Online]. Available: https://unity.com/

[5] "Controllers and hand tracking - oculus quest2," 2021. [Online]. Available: https://support.oculus.com/290147772643252

[6] "Kinect," Oct 2021. [Online]. Available: https://developer.microsoft.com/it-it/windows/kinect/

POLITECNICO DI MILANO

School of Industrial and Information Engineering

Master of Science in Automation and Control Engineering

POLITECNICO
MILANO 1863

Trajectory optimization for collaborative robotics applications

Supervisor:	Prof. Andrea Maria Zanchettin
Co-supervisors:	Prof. Paolo Rocco
	Ing. Niccolò Lucci

Master Thesis dissertation of:

Roberto La Commare

Id: 939831

Academic year 2020-2021

Contents

List of Figures

List of Tables

Abstract

Today, the field of collaborative robotics is undergoing considerable development. A growing demand for new installations has gone hand in hand with an increasing focus on safety and comfort of the operator that works with the cobot. This thesis aims at developing an effective and repeatable method for offline optimization of a cobot's trajectory before its actual deployment in a work cell. The trajectory is optimized with respect to its execution time and the safety of the operator but also avoiding any obstacles in the path. The data on the position of the operator are collected beforehand in an acquisition phase with the help of a virtual reality device. This is linked to the idea of virtual commissioning, i.e. the development of the entire work cell in simulation before it is deployed in industry. Three different methods were developed to set the robot's speed as a function of distance from the operator. Finally, performance indexes and experiments with the coexistence of an operator and a collaborative robot were used to compare and analyse the results obtained.

Sommario

Il settore della robotica collaborativa sta vivendo un considerevole sviluppo. Una crescente domanda di nuove installazioni è andata di pari passo con una crescente attenzione alla sicurezza e al comfort dell'operatore che lavora con il cobot. Questa tesi mira a sviluppare un metodo efficace e ripetibile per l'ottimizzazione offline della traiettoria di un cobot prima del suo effettivo impiego in una cella di lavoro. La traiettoria viene ottimizzata rispetto al tempo di esecuzione e alla sicurezza dell'operatore, cercando allo stesso tempo di evitare eventuali ostacoli nel percorso. I dati sulla posizione dell'operatore sono stati raccolti in anticipo avvalendosi della realtà virtuale. Tutto ciò si collega all'idea di virtual commissioning, vale a dire lo sviluppo dell'intera cella di lavoro in simulazione prima dell'effettiva implementazione.

Sono stati sviluppati tre metodi diversi per impostare la velocità del cobot in funzione della distanza dall'operatore.

Infine, sono state eseguite delle prove sperimentali che vedevano la compresenza del cobot insieme all'operatore. I dati rilevati, insieme a degli indici appositamente definiti, sono stati utilizzati per confrontare e analizzare i risultati ottenuti.

Chapter 1

Introduction

1.1 Collaborative robotics

A cobot (a term derived from collaborative robot) is a robot designed to work side by side with humans in the same workspace without the need for a safety cage surrounding it.

These robots are characterized by low inertia, low payload capabilities and the absence of trap points (i.e. points that can trap body parts) (ISO-TS 15066)[1]. Moreover they are made only by rounded edges padded with soft material that can absorb part of the energy of a possible impact.

(a) Serie A from Doosan Robotics

(b) Yumi from ABB

Figure 1.1: Examples of collaborative robots

Today, Collaborative Robotics is the fastest growing segment of industrial robotics, making great advances in the interaction between workforce and machines. Collaborative automation has shown, especially in the pandemic phase, all its potential as a game changer and its ability to support business productivity even in an extremely difficult time for industry and manufacturing in particular. For SMEs, the adoption of collaborative robotic arms is meant for keeping lines running even in conditions of staff shortage, guaranteeing social distancing and ensuring a continuous production flow without excessive entry costs and without the implementation and programming difficulties associated with traditional industrial robotics.

Considering the distinctive features of cobots compared to ordinary robots, they are expected to lead the way to mass customization, decrease space requirements, increase product quality, production efficiency, and improve working conditions for humans, both safety and ergonomics aspects [2].

The implementation of cobots allows operators to be relieved of the most uncomfortable, repetitive tasks while optimising the contribution that operators can make when employed in higher value-added tasks [3]. This concept was also highlighted by a recent McKinsey study. In fact, while less than 5 percent of jobs can be fully automated, as many as 60 percent have at least 30 percent of the operations that can be automated with current technologies, such as cobots [4]. Figure 1.2.

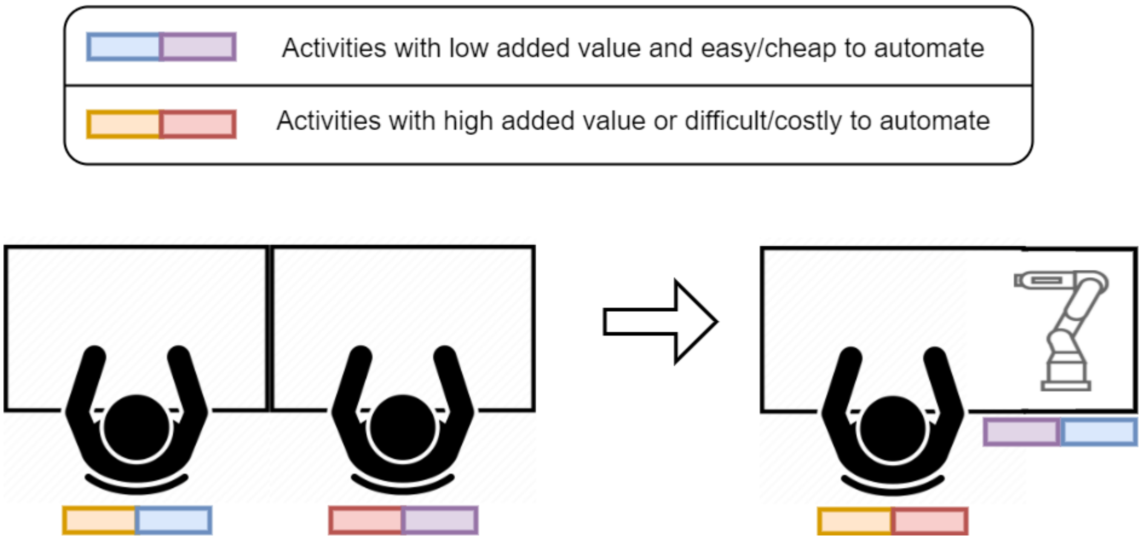

Activities with low added value and easy/cheap to automate
Activities with high added value or difficult/costly to automate

Figure 1.2: In order to partially automate jobs, it is possible to segment existing jobs into easy to automate activities and value-added activities

1.2 Thesis purpose

The purpose of this thesis is to develop an effective and repeatable method to optimize the trajectory of a cobot before its deployment in a work cell, thus without the need for additional sensors in the final setup. The trajectory must be optimized with respect to execution time and operator safety, avoiding any obstacles in the path. This is linked to the idea of virtual commissioning, i.e. the development of the entire work cell in simulation before it is developed in industry. For this reason, in order to acquire the operator's movement data, it is not possible to physically go on site and take measurements. Therefore, in line with the digital industry 4.0, it is necessary to reproduce the work cell with the help of virtual reality, making an operator wear the device in a controlled environment and acquire the motion data. Then the cobot trajectory is optimised, taking into account the motion data collected.

1.3 Thesis achievements

In this thesis the following objectives have been achieved:

- programming of a work cell on a Virtual Reality device;
- simulation and data acquisition on the most likely movements of an operator in the work cell through a Microsoft Kinect sensor;
- development of an optimization algorithm with simulation outsourced to an ad-hoc program (RoboDK);
- improvement in the execution time of a cobot compared to a manually designed path;
- test the results of co-working between one operator and a cobot Doosan A-series 0509 in three different operating strategies.

1.4 Thesis structure

The rest of this thesis is organized as follows. In Chapter 2 is given a Background knowledge about the cobot programming, the Virtual Commissioning by cobot plus human simulation and the safety standards that should be followed. In Chapter 3 it is proposed the state of the art about obstacle avoidance and trajectory optimization. The main Optimization Algorithm is presented and analyzed in Chapter 4, where will be also introduced the velocity management. In Chapter 5 three resolution approaches are exposed, one using a static human volumetric sweep, one considering a dynamic human model and a third that is the combination of the precedents. To obtain those models it has been used a Microsoft Kinect sensor and a Virtual Reality device; this data acquisition process is shown in Chapter 6. Finally, the analysis of the experimental result analysis and the conclusions are presented in Chapter 7 and Chapter 8, respectively.

Chapter 2

Background knowledge: Programming, Simulation and ISO-TS 15066

To understand how the proposed optimization method fits into today's industrial scenario it is necessary to make a quick excursus. In particular, we will briefly see which programming methods exist for cobots and their advantages and disadvantages, what is virtual commissioning and what can currently be done in the field of robotics. Finally will be discussed the standards to follow and the safety limits that must be respected.

2.1 Current methods for programming cobots in industry

The computer system that controls the manipulator must be programmed to teach the robot the particular sequence of movements and other actions that have to be performed to accomplish the task.
There are several methods to program a cobot in industry: Servo-Assist Training, by means of a Teach pendant or trough programming. (Figure 2.1)

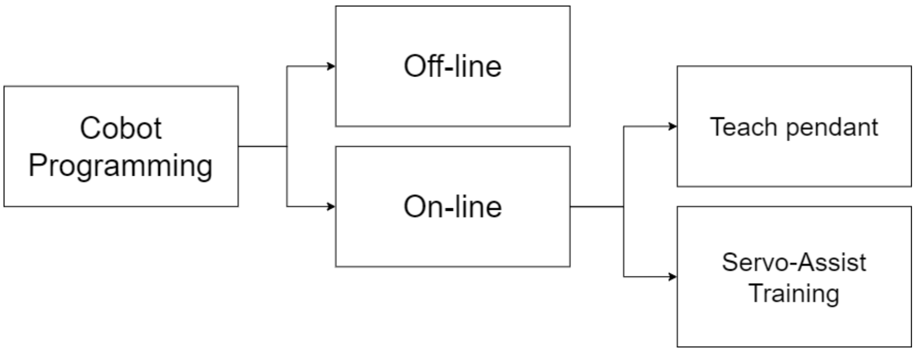


Figure 2.1: Cobots programming methods

2.1.1 Servo-Assist Training

One of the simplest methods for programming a cobot is the teach-by-showing method. The operator, by pressing a dedicated button and holding firmly the end-effector, can move the cobot to the desired position (Figure2.2). The cobot will follow the movements of the operator thanks to admittance or impedance control, facilitating the movement through the active use of servos. Once the desired configurations are reached, the checkpoints can be saved and simple routines can be created. This method has the advantage of being easy to use but it can perform only very simple tasks.

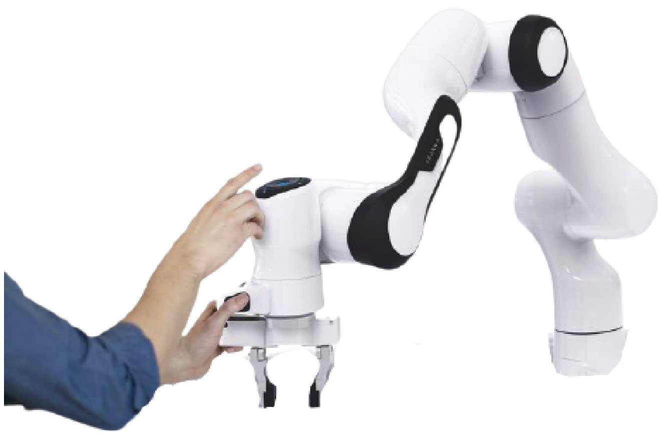


Figure 2.2: Cobot FRANKA EMIKA moved by an operator through the pressure of a dedicated button

2.1.2 Teach pendant

Another on-site programming can be performed using the Teach pendant (an example is provided Figure 2.3). It is a device connected to the controller of the robot that can be used directly to move the individual joints through a joystick or a touch-screen. Once the desired configurations are reached, it is possible to create tasks by specifying the type of motion (linear, arc of circumference, following joints, etc.). This method allows more actions to be taken rather than just using hand guiding and still has the advantage of having an immediate idea of the spatiality of the operation. On the other hand, it is not possible to carry out safety checks and it is not possible to connect speed control algorithms.

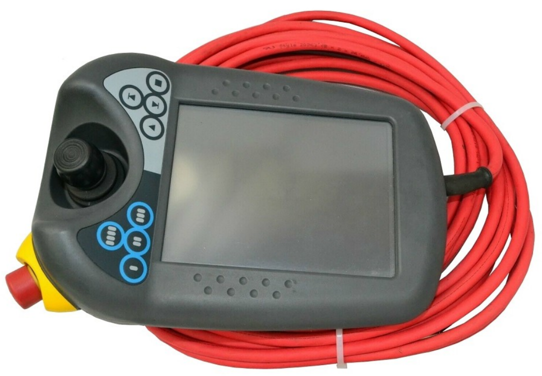


Figure 2.3: Teach pendant from ABB

2.1.3 Programming by coding

The offline method involves the use of a programming language that is very similar to a computer programming language. On top of many of the capabilities of a computer programming language (i.e. data processing, computations, communication with other computing devices, and decision making), the robot language also includes statements specifically designed for robot control. These capabilities include motion control, used to move the manipulator to a defined position in space, and input/output, used to interact with other devices as the end-effector. The result of this method is a code ready to be run that can be loaded in the robot's controller. The advantage of this method is that it enables full functionality of the robot, the management of any kind of safety algorithm and the simulation. On the other hand, it is clearly the slowest method and the only one which requires programming skills, [5].

2.2 Simulation & Virtual Commissioning

Virtual Commissioning is the practice of reproducing virtually the physical behavior of a machine or plant through a software simulation. The final goal of the emulation is to test all the designed elements and related automations, allowing to test the software development in order to remove system errors before implementing the new components and the whole manufacturing process. Because of the cost of commissioning new manufacturing facilities and production lines, applying simulation methods to manufacturing can yield significant benefits. Simulation helps achieve the best in industry in many ways. It reduces wasted time and resources and increases efficiency. It also helps to increase productivity and revenues. The simulation also plays a significant role in job safety [6] . It allows you to evaluate plant costs, overall efficiency and the ergonomics of workstations before actually starting up the plant. The only disadvantage is that this process requires additional time and resources for designing it on the specific tool [7].

2.2.1 Human and Robot Simulator

Recently the most advanced simulation tools have begun to give the possibility to integrate one or more operators in the simulation of the plant. Digital human modeling (DHM) is a key technology for addressing human factors during the production planning stages. The use of DHM, from initial concept development through all stages of the product engineering process, promises more comfortable and safer workplaces [8]. Some of these Simulation tool, together with their feature, are presented below.

- Visual Components 3D manufacturing simulation software [9] can simulate both industrial process and human activity such as loading a machine, pick and place, component assembling. (Figure 2.4)

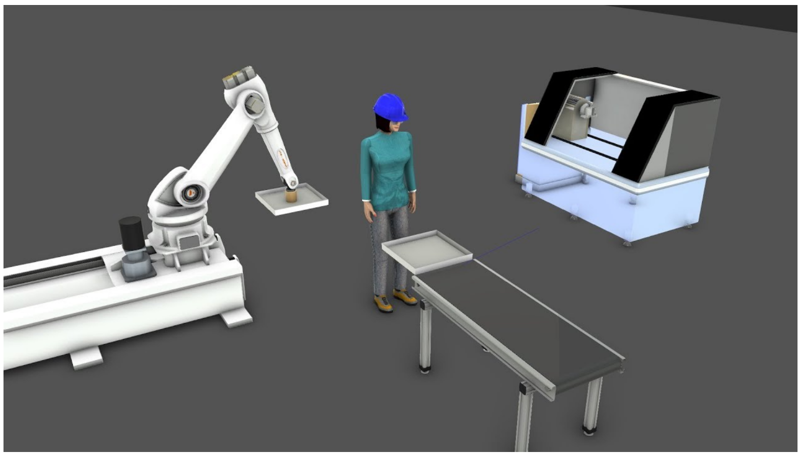


Figure 2.4: Human simulation from VisualComponents

- Delmia Dassault Systèmes [10] enable to simulate tasks, analyse the posture, estimate working time and translate the content of the action orders directly in written form. (Figure 2.5)

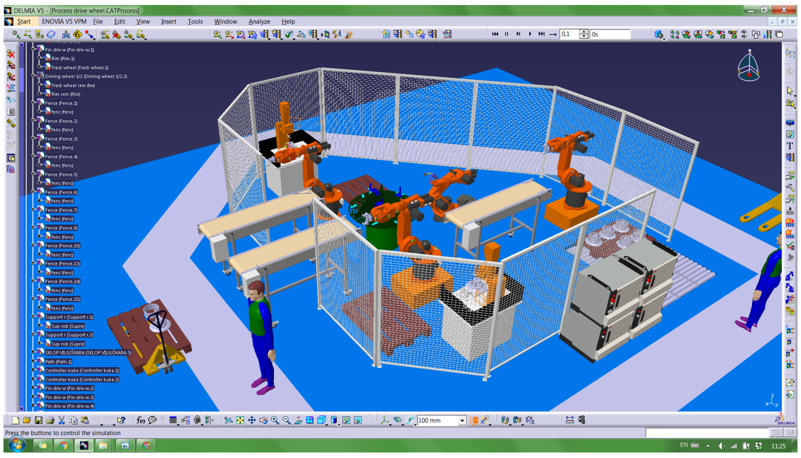

DELMIA V5 - [Process drive wheel.CATProcess]
Start
ENOVIA V5 VPM
File
Edit
View
Insert
Tools
Window
Analyze
Help
100 mm
Press the buttons to control the simulation

Figure 2.5: Human simulation from Delmia Dassault Systèmes

- Siemens Tecnomatix Plant Simulator [11] in addition to the possibility of simulating large complex plants allows to add a human model called 'Jack'.

This makes it possible to verify the size of the spaces that the operator can occupy, if collisions with robots might occur and the right timing of the operation, but also to analyse the ergonomics of the worker's positions. (Figure 2.6)

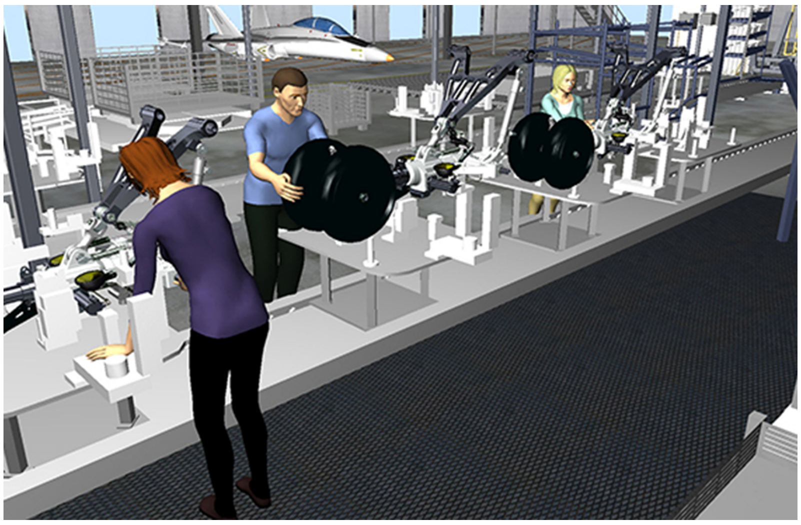


Figure 2.6: Human simulation from Siemens Tecnomatix

- ABB RobotStudio [12] is also providing a Virtual Human that is treated as a Robot: you can position a Human inside of a Station, program it with the RAPID code and create simple simulations of Humans and Robots.

One of the most common applications of production line simulation is the feasibility check, that is the simulation of movements already set up in order to verify that there will not be any collisions and the spaces of maneuver are adequate. In case of collisions between cobot and operator, the path can be changed manually and simulated again. What cannot be done with the tools on the market is to automate this process, which is done manually through trial and error, and make the result optimized.

2.3 Safety requirements

Each time a new robot is installed and a new task is designed a hazard identification must be performed. This procedure is part of the risk assessment, which is the identification, evaluation and estimation of the levels of risk involved in a situation. This process must consider the following factors: end-effector and workpiece hazards (including lack of ergonomic design, sharp edges, heaviness), robot motion characteristics (e.g. load, speed, force, torque, momentum, power), a determination as to whether contact would be transient (i.e operator body part is not clamped) or quasi-static (body part is clamped) and the parts of the operator's body that could be affected. After carrying out the risk identification, it is necessary to classify the risk. In one of the simplest classifications, the risk goes from negligible, low, medium, high and very high. (ISO 13849-1: 2015 [13])

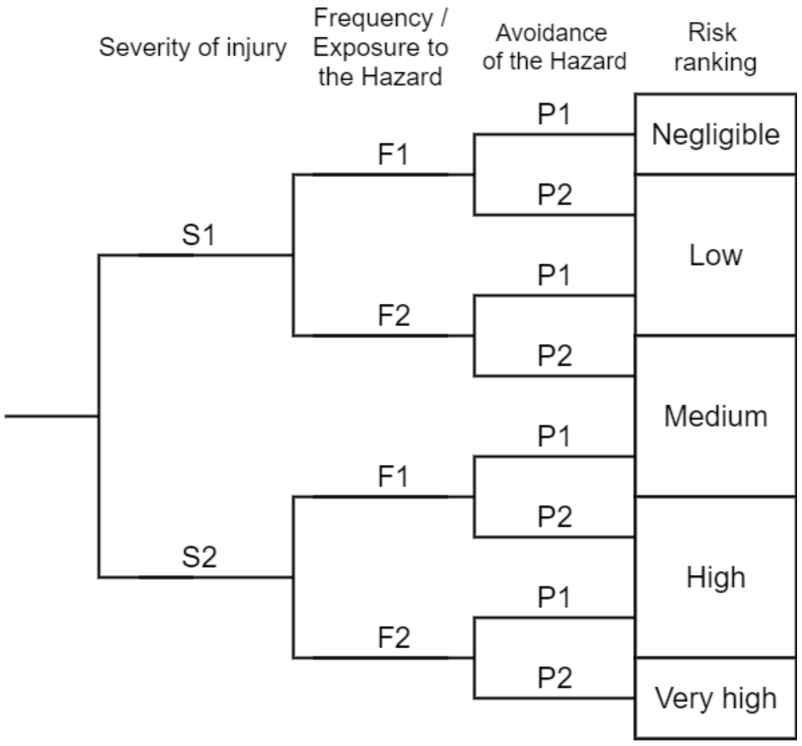


Figure 2.7: Performance Level Rating according to ISO 13849-1: 2015

S: Severity of injury

- S1: Slight (normal reversible injury)
- S2: Serious (normally irreversible injury or death)

F: Frequency and/or exposure to hazard

- F1: Seldom to less often and/or exposure time is short

- F2: Frequent to continuous and/or exposure time is long

P: Possibility of avoiding hazard or limiting harm

- P1: Possible under specific conditions

- P2: Scarcely possible

In case the risk is not low or negligible it requires a risk reduction process i.e. slow down the robot or activating further safety functions or even rethinking the entire task.

There are four types of collaborative applications presented in the ISO-TS 15066 [1] standard:

- safety monitored stops, the robot stops if the operator enters the collaborative area;

- hand guiding, the operator guide manually the robot (Section 2.1.1);

- speed and separation monitoring, the robot speed depends on the operator's distance, below a safety distance it stops;

- power and force limiting, the maximum speed of the robot is limited, in this way, in case of contact, the robot can transmit a limited amount of energy.

The two most commonly used speed algorithms for collaborative robots are Speed separation monitoring (SSM) and Power and force liming (PFL). SSM can be useful in those situations where the robot and operator share the same workspace but there is no simultaneous work (Figure 2.8). Depending on the relative velocity between robot and human, and the stopping time of the robot it can be computed the safe separation distance. If the human is below this distance the robot should slow down or stop. The stopping time T_B is composed of a contribution due to sensor reaction time and uncertainty, the controller reaction time and the time needed by the braking system to actually stop the robot motion. This criterion can be used to calculate the optimal robot velocity to assign in order to have a safe motion based on the human position $D_{\text{hum-cob}}(t)$. If we consider for simplicity a constant speed for the human (1.6m/s from ISO [1]) since the other parameters such as braking time are constant then it is possible to define this approximate equation:

$$v_{\max}(t) = \frac{1}{T_B} D_{\text{hum-cob}}(t) \tag{2.1}$$

where $v_{\max}(t)$ is the maximum velocity that the cobot cannot exceed to be in safety state. This function can be considered piece-wise or continuous depending on the type of sensors used (Figure 2.9).

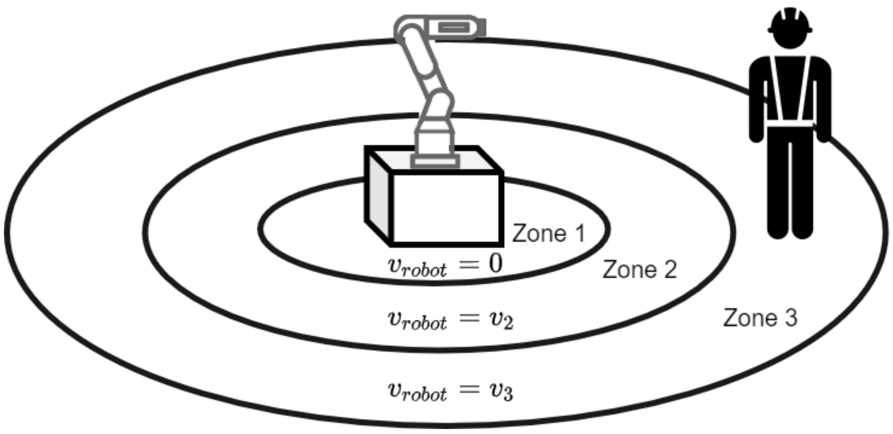


Figure 2.8: Speed Separation Monitoring approach, when the operator is in Zone 3 the max velocity of the robot is v_3, when he is in Zone 2 the max velocity is v_2 and in Zone 1 the robot stops

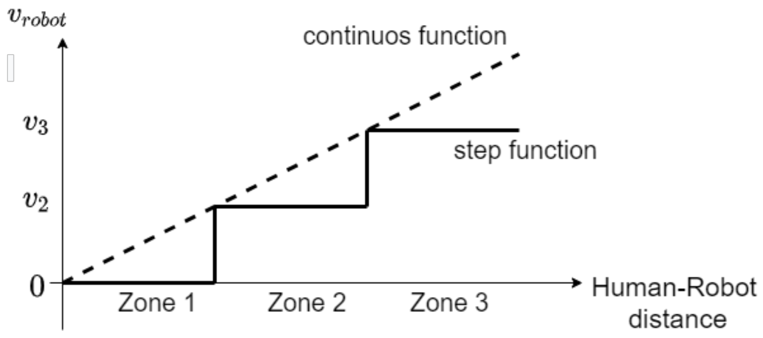


Figure 2.9: Step function vs continuous function for the robot max velocity in the Speed Separation Monitoring approach

When the operator has to cooperate with the cobot, so in a true collaborative

operation, the Power and Force limiting is necessary. In a collaborative situation, contacts can occur either voluntarily as part of the work task or involuntarily.
In order to analyze the safety of this event, it is useful to introduce the concept of dissipated energy. Each part of the body can absorb up to a limited amount of energy before producing pain in the subject. Therefore, we need to study the task and verify which parts of the body can be affected. It is necessary to investigate if a possible contact can be considered quasi-static. This type of collision occurs when the robot clamps or block a human part. To model the quasi-static impact it can be used the spring-mass model of the man.

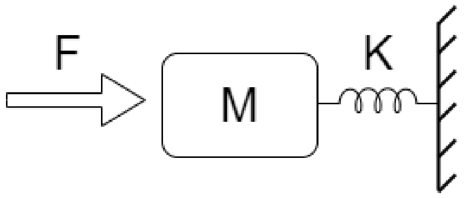


Figure 2.10: Human model as mass-spring system

where the energy absorbed by the operator is

$$E = \frac{F^2}{2K} \tag{2.2}$$

with K the elastic constant of the body part involved and F the force that is compressing the spring. If the body part is soft, like the abdomen, the elastic constant will be lower (10 N/mm according to Table A.3 of ISO/TS 15066 [1]) compared to other parts such as the lower leg (60N/mm). Hence we can approximate the energy of the robot as

$$E = \frac{1}{2} m_R v_R^2 \tag{2.3}$$

where m_R, v_R are the effective mass and velocity of the robot. Finally we can match the energies and get the following limit

$$\begin{gathered} \frac{1}{2} m_R v_R^2 = \frac{F^2}{2K} \\ v_R^{max} \leq \frac{F^{max}}{\sqrt{K}} \sqrt{m_R^{-1}} \end{gathered} \tag{2.4}$$

the value F^{max} is the maximum force allowed on a given body part, those values are found in the ISO/TS 15066 [1]. While v_R^{max} according to PFL is the maximum speed at which a robot can be set for the task being considered.

Chapter 3

State of the art

The problem we want to address in this work falls within the large set of motion planning problems. Motion planning is a computational problem where the objective is to find a sequence of valid configurations to make the robot move from an initial configuration to a final one. There are many ways of solving the problem in an automatic way, depending also on the time constraints imposed, i.e. whether the computation is to be done in real-time, involving the occurrence of events that significantly modify the robot's environment, or offline, where obstacles (even mobile ones) are known a priori. In our particular problem, we are dealing with offline optimization, so the algorithm computation time will not be as crucial as in the online case.

3.1 State of the art of obstacle avoidance

Path planning is particularly interesting when it involves obstacle avoidance. The most frequently used methods for solving this type of problem can be divided into three types: Grid-based search, sampling-based algorithm and Artificial Potential Fields. The sampling-based algorithms can be further divided into probabilistic road-map (PRM) and rapidly-exploring random tree (RRT).

Grid-based approaches divide the space of configurations (which in a 6DoF robot manipulator is $\mathbb{R}^6$) with a grid, placing a configuration at each grid node.

The robot can move towards adjacent grid points, i.e. by varying one or more joint angles, as long as the line between them does not pass through an obstacle. This approach discretizes the set of actions that the robot can perform. After this grid definition a search algorithm is used to find a path from the node that corresponds to the start configuration to the endpoint.
In order to be able to divide space with a grid, it is necessary to set a resolution, i.e. a minimum distance between two points. Grid definition and subsequent route finding is faster with coarser grids, but has some disadvantages: the end result will not be very smooth and it will not be possible to find paths through narrow parts. Furthermore, as the grid resolution improves, the number of points on the grid increases exponentially with the size of the configuration space, making the problem more difficult to solve (this is also known as "curse of dimensionality").
K. Kaltsoukalas and S. Makris [14] try to solve some of these problems by reducing the large search space by applying a set of parameters. In particular, two sub-grids are created, one with the joints that mainly influence the movement of the robot in the workspace (position of the end-effector) and another grid with the joints that mainly influence the orientation of the end-effector. D. Šišlák[15] focuses his work on the search algorithm, with the goal to find the best grid-based path more rapidly. D. Ferguson [16] focused on the post-processing aspect, trying to smooth out the grid-based trajectory by means of interpolation.

A more efficient method for partitioning the space of configurations is the *probabilistic road-map*, introduced by Kavraki [17]. Also in this method, a two-stage procedure is used: a learning phase and a query phase. In the learning phase, a probabilistic roadmap is created and stored as a graph whose points correspond to collision-free configurations and whose edges represent feasible paths between them. In the query phase, any start and finish configuration of the robot is linked to two nodes in the road-map; the road-map is then searched for a route that joins these two nodes. Since this first study, much further research has been carried out, also considering real-time mobile obstacles [18].

A similar approach is *Rapidly-Exploring Random Trees*, which has the advantage of requiring only one of the two phases of the previous approach [19].
The configuration space is also used in this method. It creates two structures of branching configurations in the form of a tree beginning in the start and end configuration. At each point in the tree the algorithm finds a random configuration using a uniform probabilistic distribution, then a neighbouring configuration is searched in the direction of the previous one. If the entire trajectory between the tree and the new configuration is free of obstacles the tree is increased. In more recent study this algorithm has been improved in the memory needed to store the tree [20] or in the search for paths in restricted environments [21].

These types of sampling-based motion planners have the advantage of being probabilistically complete and are computationally efficient. However, these planners often require a post-processing step to smooth and shorten the calculated trajectories. It must also be considered that some computational effort is spent on sampling and linking nodes in portions of the configuration space that may not be useful for the task.
These limitations lead to the need to introduce the concept of optimization.
The objective is to obtain a trajectory represented as a sequence of states in the space of configurations in order to optimize a given goal.
There are two main fields in which optimization can be used in robot motion planning. First, it can be as a post-processing of the three methods seen so far. In particular, it can be used to smooth and shorten the calculated trajectories. However, this usually can not provide enough flexibility in the generation of collision-free trajectories in presence of obstacles, since if the initial trajectory was close to an obstacle, the optimized trajectory could pass through it. [22]
Secondly, the optimization technique can be used to calculate locally optimal and collision-free trajectories from random trajectory initialization.
One type of path planning with obstacle avoidance that fits well with the optimization concept is the *artificial potential* [23]. In this article Khatib has provided a method of path planning using the artificial potential approach. One substan-

tial difference is that the approach works directly in the task space instead of in the joint configuration space. The obstacles are associated with a repelling potential and the target with an attractive potential. Those potentials can be either parabolic or conical. The motion to the next configuration of the path will go towards the direction of the artificial field, like an electrical charge. After this first implementation of the Artificial potential many more article have been written analyzing different strategies. R. Volpe [24] has generated different potentials depending on the joint under consideration, allowing the end effector to move towards an object of interest while all other joints try to keep as far away as possible. Similarly, Wang [25] solved the problem in a similar way except that he used an iper-redundant manipulator. A method conceptually similar to this one is used in this thesis, regarding the management of obstacle avoidance.

3.2 State of the art of Trajectory optimization

A more detailed overview of the types of optimization that have been used to date is given in this section.
Trajectory optimization is a technique for calculating the optimal path and dynamics for a robot in order to perform a given task. It is a relatively straightforward process. First it is needed to define a cost or merit function and the equations or simulations that govern the process under study. Then, using an iterative algorithm, starting from an initialization, it tries to move the decision variables to positions that produce the lowest cost function, i.e. a minimum. Since no universal analytical solution can be found for non-linear problems, a variety of numerical approaches have been developed for trajectory optimization. In most numerical approaches, the continuous-time problem is first converted into a discretized optimization problem through a procedure called transcription, i.e. instead of considering a trajectory composed of infinite points and states, a sampling is done [26]. The optimization problem then can be solved by a generic optimization solver. In numerical methods for trajectory optimization, two sections play the core role: discretization and optimization.

The problem can therefore be categorized in terms of discretization, i.e. the definition of the trajectory and the decision variables. Here we will analyse some of these different approaches.

In this paper [22] the used algorithm is a sequential convex optimization. It is an efficient way to optimize a problem with a non-convex cost function. As far as the definition of decision variables is concerned, the authors propose a vector of dimension $T \times K$, where T is the number of time-steps and K is the number of degrees of freedom. This method divides the continuous problem into a series of equal time steps. It does not take into account speeds or accelerations but only the obstacle avoidance as a function of the penalty cost and constraints.

In this study, [27] the trajectory is divided into a number of configurations equally spaced in time. These nodes are joined together using cubic spline interpolation. The optimization variables are then the position of the spline nodes and the total time of the operation T. The objective is to minimize the time together with the mean torque of the joint.

Here [28] Wang, C. et al. try to minimize the execution time of the trajectory. The first procedure is to run a simplified algorithm in which the decision variables are the points of the trajectory joined linearly to form an open polyhedron. After that another trajectory is generated with a B-spline between each of the points of the polyhedron. The B-splines will be handled by additional decision variables called virtual points. Unlike normal splines, the trajectory of B-spline does not pass through these virtual points but only through the beginning and the end.
The trajectory is not provided in analytical form but requires multiple evaluations that must be included in the trajectory optimization algorithm. A similar method was used by Jiangyu Lan et al [29], but the order of the B-spline was increased at 7, making the trajectory even smoother.

In this study [30] the problem of path and trajectory are taken into account simultaneously. This paper looks for a method to minimize the cycle time in conjunction with a parameter for smoothing the dynamics, i.e. minimizing the square of the jerk. The trajectory is discretized as $\{x(T_i), i = 0, \dots N\}$ where x is a configuration of 6 joint angles and $\{T_i, i = 0, \dots N\}$ are called knots and N is the number of knots. The interesting thing is that these knots are not equispaced in time, as in other methods, but instead are arranged in such a way that at the ends they become denser, and in the middle they are more sparse. This method of dividing knots is called Chebyshev-Lobatto and has many useful properties. Some of these are the incredible fidelity of approximation to practically any curve, avoiding the oscillatory phenomena that occur in the case of high order global polynomial interpolation, they are integrable and derivable by means of a static matrix, which makes the numerous derivations necessary for each iteration extremely simple.
In a similar way, Luo et al. [31] used Lagrange interpolation to discretize the trajectory. More specifically they used the Chebyshev points and create a trajectory by interpolating them with the Lagrange method. This was done to find an energy optimal trajectory.

In all these studies, the kinematics and dynamics of the manipulator were calculated numerically using equations that require specific data on the inertia matrices of each robot joint, which is often difficult to find. To overcome this problem, a simulator is used in this work that actually emulates the real movement of the robot together with the operations of the robot controller. This will allow us to focus more on the concept of obstacle avoidance and speed management in relation to an operator position, thus leaving the burden of simulation to an external tool.
As far as the discretization of the trajectory is concerned, the functions of the simulator were used directly. In particular, the trajectory is defined using two move commands for each path between the two trajectory ends. Therefore the decision variables are one intermediate configuration for each passage from start

to end configuration and vice versa. With regard to the dynamics, i.e. the speed management, a denser discretization was made, dividing the trajectory into n (with $n > 20$) sub-trajectories with an assigned maximum speed.

3.3 State of the art Safety algorithm

Regarding security monitoring, the literature has focused on the online problem. This first study [32], focused on the definition of a control algorithm that limits the speed of the industrial robot using the concept of SSM. An infrared camera determines the real-time position of the operator, defining a series of features such as the position of the head, elbows and hands, thus approximating the operator to a set of spheres and cylinders. Then an algorithm would calculate the minimum point-to-point distance to the robot. If the distance decreased below a threshold, a scaling factor δ was imposed to reduce the speed of the robot. The speed was reduced so that the distance to the operator was sufficient for the robot to brake safely. In particular, at $\delta = 1$ the robot moves at nominal speed along the predefined path, at lower δ the speed is reduced proportionally, if δ becomes 0 the robot stops completely.
In subsequent paper [33], the definition of the human was considerably improved. The operator has been described as a bi-cycle mobile vehicle for walking movements, while for the upper body the operator was approximated to a torso with two arms consisting of a series of joints. Based on the speed of the person, it was possible to define a volume of future possible configurations. By using this volume as the model against which calculate the distance to the robot, safety was improved, since a basic prediction of the possible configurations was also taken into account.
In [34], the definition of the constraints on the distance between the link and the obstacle was improved. In general, this made the constraints less conservative, reducing the robot's cycle time considerably without jeopardizing SSM. Another study [35] adds the PFL algorithm, which is specific to the world of collaborative robotics, to previous works. The aim is that when the cobot comes too close to

humans, instead of stopping completely as required by the SSM, the cobot will simply go to the speed that PFL indicates as safe for contact.
What these online algorithms cannot do is to change the nominal path depending on the operator's position. This is due to various issues, from the complexity required to the risk of running into other obstacles in the workspace. In fact, it is always recommended to test offline the paths to be executed. This thesis can actually make a contribution in this context. Optimizing the offline route once before the actual deployment of the robot can bring even greater benefits when applying online algorithms. This is because the robot with an optimized path will be on average at a greater distance from the operator than a non-optimized path, thus limiting the need of speed reduction in the online algorithm.

Chapter 4

Optimization Algorithm

In this Chapter the algorithm used for optimization of the cobot's trajectory will be presented. The optimization algorithm uses simulations outsourced to an ad-hoc program: RoboDK [36], which also is described in the following sections.

4.1 Robot simulation environment

RoboDK is a powerful simulator for industrial robots. It has an extensive library of industrial robot arms and tools from over 40 different robot manufacturers, making it possible to simulate any robot controller.
RoboDK has grown rapidly since it was founded and is now used by companies of all sizes, from startups to the largest corporations in the world. [37]

In this thesis RoboDK simulation is performed using the Python API. The RoboDK API for Python includes all the offline programming features of the RoboDK simulator and allows to implement automated applications for a wide variety of robots and mechanisms.
In order to simulate the robot's trajectory efficiently it has been used an API that receives as input the points (6 angle joint configuration) of a trajectory and the required velocities (one for each moves). The Python function then creates a robot program by cycling the input configurations: the list of commands is composed with joint moves or linear moves and with the set of maximum speed.
When all the configurations and moves have been added, the function asks for

an analysis of the newly created program, this will return the values of the joints angles, the actual speeds and accelerations of the joints and the cartesian position of the end effector. At the beginning of a new simulation the old program is deleted. In this way, in a fraction of a second all the data that may be useful to know for an optimization are obtained.

The truly remarkable aspect is that this function is independent from the chosen robot: if you want to change the brand of cobot you just need to load on the RoboDK GUI a new robot and the program will work immediately after. In addition, since the simulation includes not only the kinematics of the robot but also the behaviour of the controller, we can be sure that once the program has been loaded into the real robot it will perform exactly the same movements.

The main optimization program will be executed on MATLAB® because of its ease of use and simplicity in the management of matrix data. Furthermore, all the standard Python library content can be called from MATLAB®, as any other third-party or user-created modules. In this way it is possible to outsource the simulation of the cobot to RoboDK via Python function.

4.2 The Optimization Algorithm

The general idea is to optimize a pick and place task, being one of the most versatile and common functions we can find in robotics. The *start* and *end* configurations of the task depends on the position of the robot in the working area so they are not modifiable. The trajectory will be optimized by modifying an intermediate point as it is depicted in Figure 4.1.
The optimization will then look for the intermediate configuration that results in the shortest task time, in other words the cost function is the working time and the optimization variable is an intermediate configuration. The cost function takes into account the distance between the robot and the operator (Chapter 6) and the environmental obstacles. In total the optimization will look for three balancing objectives: minimize time, maximize safety and avoid obstacles.

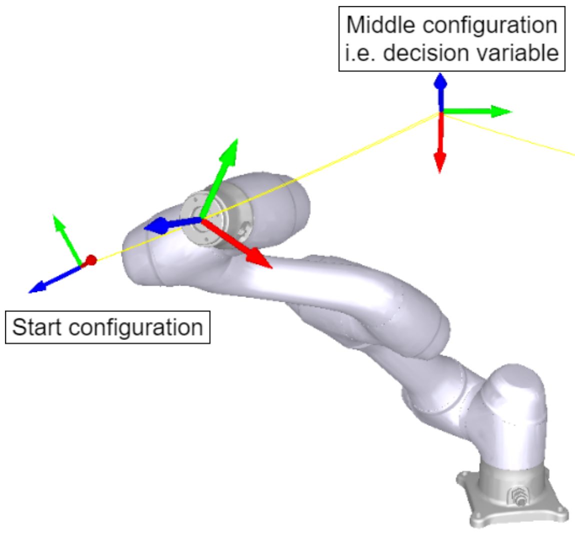


Figure 4.1: Visualization of the simulated cobot trajectory, along with the start and end fixed configuration and the middle one, that is the decision variable of the optimization.

4.2.1 Structure

The algorithm that has been implemented is an *unconstrained continuous non-linear problem solver with line search method*. In a previous draft it had been added the upper and lower constraints of the joint, making the problem a *constrained* one, but these limits were never reached, making the constraints superfluous. Therefore in a second draft they have been removed to accelerate the algorithm, considering that in case of wrong angles the simulation would stop anyway. Now the main notations are presented and the structure of the algorithm explained.

– Algorithm inputs: cost function $f(\boldsymbol{x})$, initial guess $\boldsymbol{x}^0 \in \mathbb{R}^{12}$ (2 times 6 joint angles, one for the outward and one for the return of the pick and place).

– Parameters: termination tolerances TOL_∇, TOL_x, $TOL_f \in (0,1)$, maximum number of iterations $N_{\max} \in \mathbb{N}$.

– Algorithm:

1. Compute the derivative of the cost function $\nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^k\right)$ (see the following subsection 4.2.2);

Decision variables	$x \in \mathbb{R}^n$
Initialization value	$x^0 \in \mathbb{R}^n$
Cost function	$f : \mathbb{R}^n \to \mathbb{R}$
Global minimizer	$\boldsymbol{x}^* : f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*)\ \forall x \in \mathbb{R}^n$
Global minimum	$f(\boldsymbol{x}^*) : \boldsymbol{x}^*$ is a global minimizer
Local minimizer	$\boldsymbol{x}^* : \exists$ a neighborhood $\mathcal{N}$ of $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*)\ \forall \boldsymbol{x} \in \mathcal{N}$
Local minimum	$f(\boldsymbol{x}^*) : \boldsymbol{x}^*$ is a local minimizer
Line search direction	p^k

Table 4.1: Main notation and definitions pertaining to the optimization problem.

2. Compute a line search direction p^k using Broyden–Fletcher–Goldfarb–Shanno (BFGS)[38] algorithm: $\boldsymbol{p}^k = -\left(H^k\right)^{-1} \nabla_x f\left(\boldsymbol{x}^k\right)$, where $H^0 = I$ and iterating H^k as a sub-routine. This method use only the gradient evaluation of the cost function and it is able to approximate the Hessian matrix H^k, improving it with each iteration. This vector $\boldsymbol{p}^k$ is the descent direction, i.e. the direction where the cost function decrease more rapidly (Figure4.2)

3. Carry out the line search algorithm (see below) to compute the step length $\boldsymbol{t}$, then compute the next iteration $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + t\boldsymbol{p}^k$. Figure 4.2

4. Update the relative changes of optimization variables and cost function

$$\Delta x^k = \frac{\left\|x^{k+1} - x^k\right\|}{\left\|x^k\right\|} \text{ and } \Delta f^k = \frac{\left|f\left(x^{k+1}\right) - f\left(x^k\right)\right|}{\left|f\left(x^k\right)\right|} \tag{4.1}$$

5. Check if the directional derivative is too small $\left|\nabla_x f\left(x^{k+1}\right)^T p^k\right| \leq TOL_\nabla$, check if the relative changes of optimization variables and cost function are negligible $\Delta x^k \leq TOL_x$ or $\Delta f^k \leq TOL_f$, check if the maximum number of iteration are reached $k \geq N_{\max}$. If any of the above conditions are true the algorithm will finish. This means either $x^* = x^{k+1}$ is a possible local minimizer (and $f(x^*)$ a minimum) or the algorithm has ended prematurely.

6. Set $k = k + 1$ and go to 1.

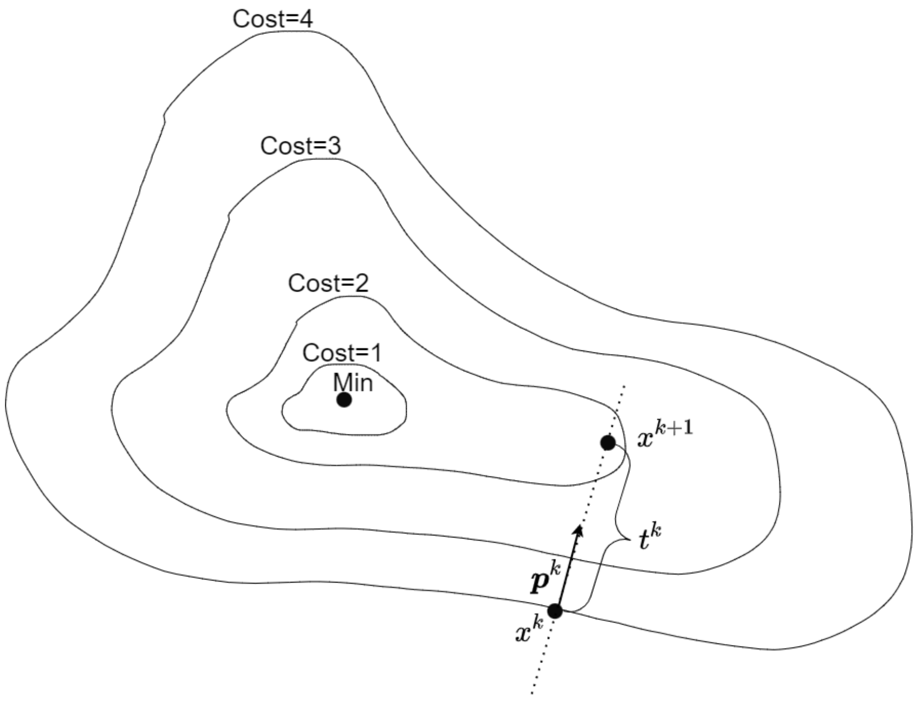


Figure 4.2: 2-dimensional example where are visualized the decision variable x^k, the search direction $\boldsymbol{p}^k$, the step length t^k and the next iteration x^{k+1}

One of the most important sub routines is the line search algorithm: it returns the length of the step to be taken in the direction of the descending gradient of the cost function to calculate the next decision variable iteration $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + t\boldsymbol{p}^k$. The optimal solution would be to perform a minimisation algorithm to search for the step t that returns the minimum cost along the search direction $\boldsymbol{p}^k$:

$$t^k = \arg\min_t f\left(\boldsymbol{x}^k + t\boldsymbol{p}^k\right) \tag{4.2}$$

This approach is not feasible as it would require a minimisation within each iteration of the main algorithm.

The line search algorithm instead use a iterative approach. It starts from a user defined max step $\bar{t}$, and then run an iterative sub-routine where at each iteration $i = 0, 1, 2, \ldots$ the step length is reduced until the *Armijo sufficient decrease condition* is satisfied:

$$f\left(\boldsymbol{x}^k + t^i\boldsymbol{p}^k\right) \leq f\left(\boldsymbol{x}^k\right) + t^i c \nabla_x f\left(\boldsymbol{x}^k\right)^T \boldsymbol{p}^k \tag{4.3}$$

where $c \in (0, 1)$ is a user-defined parameter. If this condition is met then the improvement obtained in the cost function, $f(\boldsymbol{x}^k + t^i p^k) - f(\boldsymbol{x}^k)$, is at least a fraction c of the improvement expected using the linear approximation. This approximation is obtained by truncating to first order the Taylor expansion of $f(x^k + tp^k)$ computed at $t = 0$ (Figure 4.3).

The sub-routine works as follow:

– Inputs: cost function $f(x)$, current iterate $x^k \in \mathbb{R}^n$, cost value $f\left(x^k\right)$, gradient $\nabla_x f\left(x^k\right)$, line search direction $\boldsymbol{p}^k \in \mathbb{R}^n$.

– Parameters: maximum step length $\bar{t}$, scalars $\beta, c \in (0,1)$, maximum number of iterations $N_{LS,\max}$.

– Algorithm:

Algorithm 1 Line Search Algorithm

```
$t^0 = \bar{t}$
$i = 0$
**while** $f\left(\boldsymbol{x}^k + t^i \boldsymbol{p}^k\right) > f\left(\boldsymbol{x}^k\right) + t^i c \nabla_x f\left(\boldsymbol{x}^k\right)^T \boldsymbol{p}^k$ and $i < N_{LS,\max}$ **do**
    $t^{i+1} = \beta t^i$                ▹ the next step is a fraction of the old one set
    $i = i + 1$
**end while**
```

If this back-tracking line search sub-routine with Armijo condition is used along with a quasi-Newton approach as BFGS it is theoretically guaranteed to converge to a local (or global) minimum. [39]

Since the cost function is non-convex, there may be several local minima. This is one of the major problems in the field of non-linear optimisation, and so far no solution has been found.
What is commonly done is to reiterate the optimisation several times with different initialization so as to compare the solutions and choose the best one, which theoretically is not yet guaranteed to be an absolute minimum but at least heuristically is actually better than a single attempt.

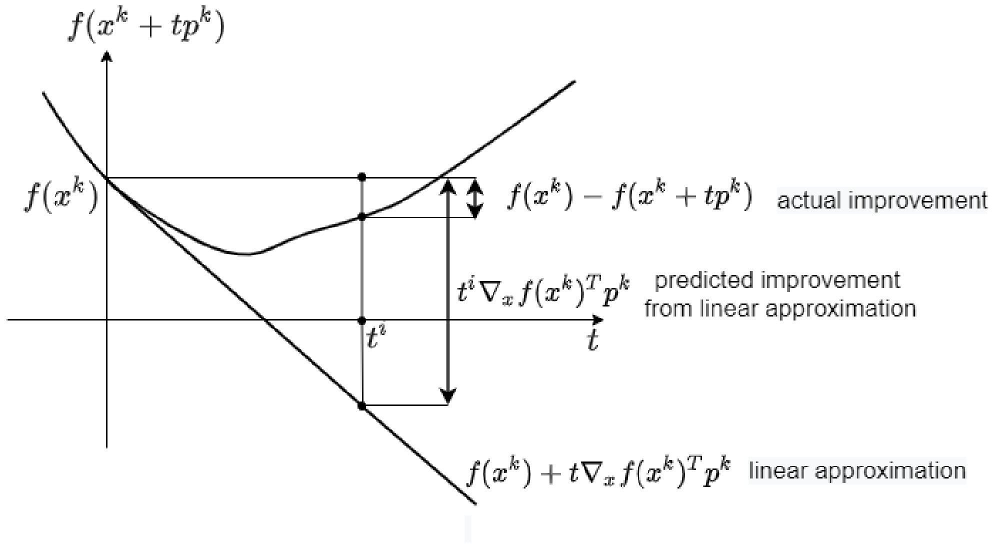

$f(x^k + tp^k)$
$f(x^k)$
$f(x^k) - f(x^k + tp^k)$ actual improvement
$t^i \nabla_x f(x^k)^T p^k$ predicted improvement from linear approximation
t^i
t
$f(x^k) + t\nabla_x f(x^k)^T p^k$ linear approximation

Figure 4.3: Visualization of the actual improvement vs the predicted linear improvement

4.2.2 Computing derivative

Since the cost function is the result of a complex simulation it can not be derived analytically. Among the many methods that exist to compute approximate derivatives it was decided to use finite difference method. This category of methods is divided into Forward difference and Central difference. To calculate the derivative of a function $f(x) : \mathbb{R}^n \to \mathbb{R}$ with forward finite difference it has to compute separately each $i = 1, 2, ...n$ component. To accomplish this, consider a vector $\boldsymbol{p} \in \mathbb{R}^n$ equal to zero except for the i-th entry, which is set to 1. Then

$$\nabla_x f(\boldsymbol{x})^T \boldsymbol{p} = \frac{df(\boldsymbol{x})}{dx_i} \approx \frac{f(\boldsymbol{x} + \eta \boldsymbol{p}) - f(\boldsymbol{x})}{\eta} \tag{4.4}$$

where η is a very small positive number with respect to the function magnitude. The number of function evaluation is one for each component and one for $f(\boldsymbol{x})$, for a total of $n + 1$. The expected accuracy of the gradient estimate of functions with unit order of magnitude is of the order of 10^{-8} due to truncation error and numerical round-off.

To improve the accuracy with respect to forward differences, one can use the central finite difference. Then, the approximation of the i-th component of the gradient is:

$$\nabla_x f(\boldsymbol{x})^T \boldsymbol{p} \approx \frac{f(\boldsymbol{x}+\eta\boldsymbol{p}) - f(\boldsymbol{x}-\eta\boldsymbol{p})}{2\eta} \tag{4.5}$$

Its accuracy is of the order of 10^{-11}. However, the number of function evaluation for one gradient computation increases at each component it needs two of them so $2n$ for the whole gradient. On the other hand, the larger computational cost may be compensated by a lower number of iterations in the numerical optimization algorithm, thanks to the better accuracy. Since the entire optimization only needs to be performed once before the actual implementation, it is not time sensitive, so accuracy is preferable to fewer calculations.

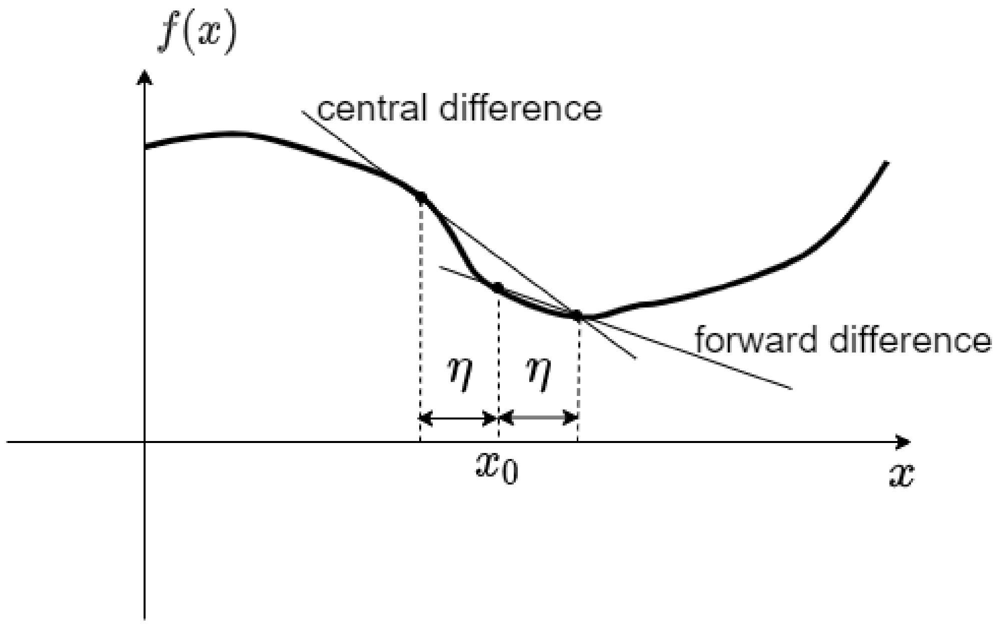


Figure 4.4: Visualization of central and forward derivative approximation in x_0

4.2.3 Cost function

The cost function is the core of the optimization algorithm. Depending on how this function is defined it will determine where the robot's trajectory will converge.

The function takes as input the configurations that define the pick and place. The trajectory will pass by: start, middle 1 (forward), end; then coming back passing by: middle 2 (backward), start (Figure 4.5). The two middle configuration can not be the same, since the optimum in one direction can be different from the optimum in the other one. The start and end configurations are fixed, while the two middle represent the $6+6$ variables to optimize.

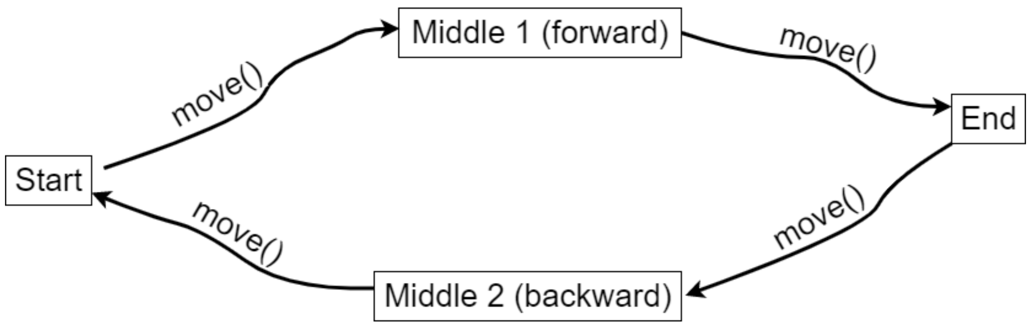


Figure 4.5: Visualization of the configuration the robot will pass through during the trajectory

The trajectory is simulated using RoboDK. From this simulation, a list of successive joint configurations sampled at $0.1s$ is extracted.
For the calculation of the distances between robot and obstacles the joint configuration q is not enough, it is necessary the 3D position of each link with respect to the base frame. So it is required to solve the direct kinematic problem. The direct kinematic equation can be expressed through the homogeneous transformation matrix of each link frame with respect to the base frame.
The Denavit-Hartenberg parameters of the robot are needed in order to define a frame with respect to the preceding one. Following the Denavit-Hartenberg convention [40] it is possible to define α, θ, d and consequently derive the forward kinematic for a generic robot. Then the *transformation matrix* from a generic frame $i-1$ to the following one i is:

$$\mathbf{A}_i^{i-1}(q_i) = \begin{bmatrix} c_{\vartheta_i} & -s_{\vartheta_i}c_{\alpha_i} & s_{\vartheta_i}s_{\alpha_i} & a_i c_{\vartheta_i} \\ s_{\vartheta_i} & c_{\vartheta_i}c_{\alpha_i} & -c_{\vartheta_i}s_{\alpha_i} & a_i s_{\vartheta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

where c_{α_i} and s_{θ_i} are the abbreviation for $cos\alpha_i$ and $sin\theta_i$. It is possible to construct a function that takes as input the Denavit-Hartenberg parameters of the robot with the current configuration q and returns the position of each link's reference system with respect the base frame thanks to a concatenation of transformations.

$$\mathbf{T}_1^0(\mathbf{q}) = \mathbf{A}_1^0(q_1) \tag{4.7}$$

$$\mathbf{T}_2^0(\mathbf{q}) = \mathbf{A}_1^0(q_1)\mathbf{A}_2^1(q_2) \tag{4.8}$$

$$\mathbf{T}_n^0(\mathbf{q}) = \mathbf{A}_1^0(q_1)\mathbf{A}_2^1(q_2)\ldots\mathbf{A}_n^{n-1}(q_n) \tag{4.9}$$

At this point the distance between each segment of the robot and the obstacles can be calculated. These obstacles are represented by 3D segments in the Matlab space (as show Figure 4.10). In our use case the obstacles are the edge of the other machines present in the scene. Since both the obstacle and the robot link are approximated as segments we need a function that calculates the distance between two segments. The algorithm used is inspired by the one of Dan Sunday [41].
The algorithm takes as input the two segments defined by the pair of points l_1 r_1 and l_2 r_2.
Then it looks for the minimum distance between the two lines identified by the segments. The closest points are d_1 on line 1 and d_2 on line 2.
Now it check if d_1 is actually between l_1 and r_1.
If d_1 falls beyond the limits of the segment's endpoints then according to the side where it falls the closest point is one of the endpoint $d_1 = l_1$ or $d_1 = r_1$.
At this point the distance from d_1 to the second line is calculated. As before we check if point d_2 is inside segment 2, in that case the algorithm stops and the distance is $\overline{d_1 d_2}$ Figure 4.6, otherwise we find which end is closer and define $d_2 = l_2$ or $d_2 = r_2$ in this case the distance is between the closest end-points of the two segments (Figure 4.7).

We can approximate each link of the robot as a cylinder with a hemispherical base. The distance calculated is therefore between the central axis of the link cylinder and the obstacle cylinder (Figure 4.9, 4.10). In order to find the actual distance then it is necessary to subtract an offset which represents the radial thickness of a robot link. The same reasoning is applicable to the segment repre-

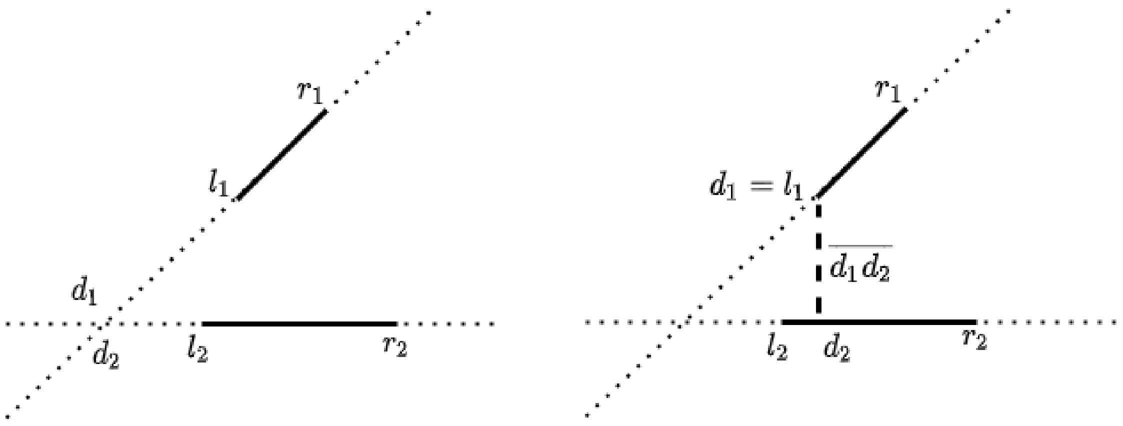


Figure 4.6: Bi-dimensional example of the segment-distance algorithm in case of one-limit distance

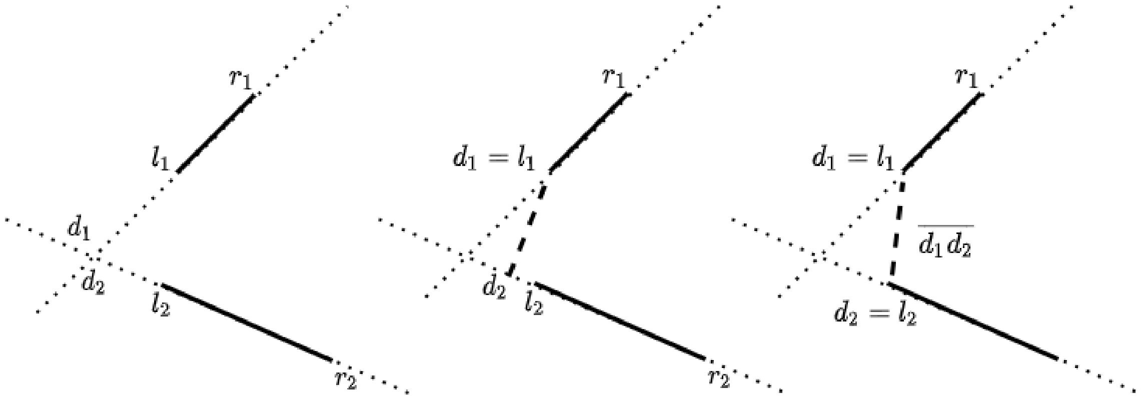


Figure 4.7: Bi-dimensional example of the segment-distance algorithm in case of two-limit distance

senting the human body (Figure 4.8). These values are available on the standard ISO 7250-1:2017 and on the cobot design tables.

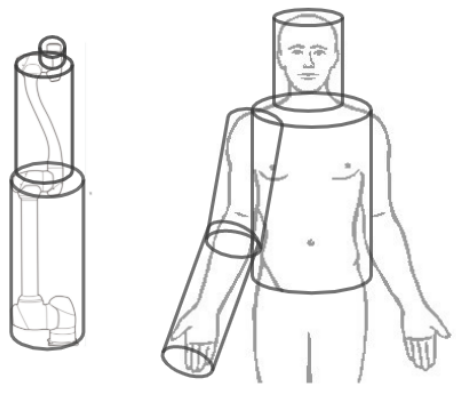


Figure 4.8: Human and cobot approximation using cylinders

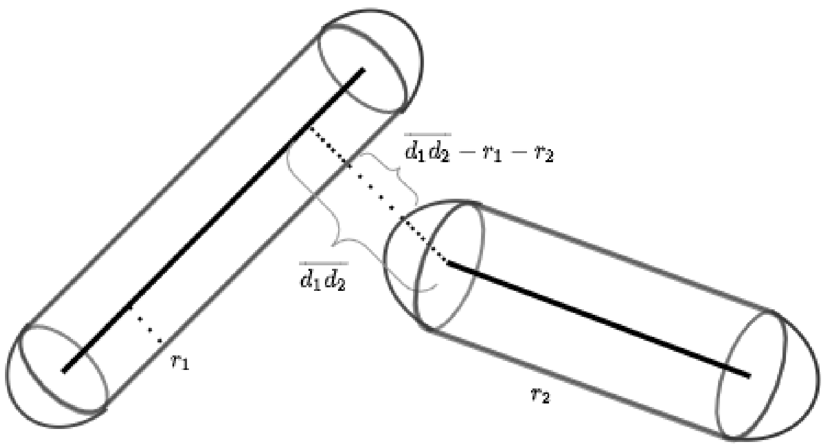


Figure 4.9: Distance between two cylinders with a hemispherical base

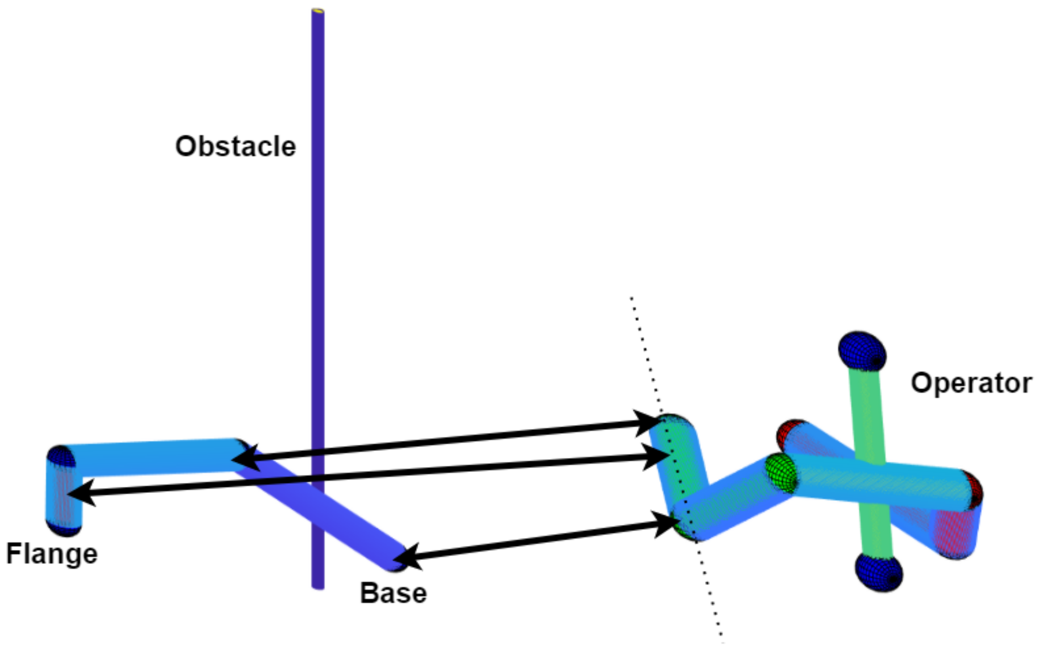


Figure 4.10: Visualization of some distances calculated between three robot joints and the lower left arm of the operator. Notice that the radius of the cylinders are not the real offset used (Fig:4.8).

In order to keep the robot-path away from the obstacle and not pass through it, a method similar to artificial potential was used. A penalty in the cost function is added if any of the robot link are close to any obstacle below a threshold h. This penalty is inversely proportional to the distance.

```
**if** dist_edge $< h$ **then**
    cost $=$ cost $+ (h -$ dist_edge$)$
**end if**
```

This procedure is similar to the method of artificial potential with conic shape: if the robot gets too close to an obstacle, the cost function will increase rapidly, making the trajectory converge to a safer distance.

4.2.4 Velocity management

The trajectory, is defined by 4 major movements (start-middle1, middle1-end, end-middle2, middle2-start). When the simulation is carried on it is possible to define a velocity for each movement. In this setup the trajectory can change velocity up to 4 times. To improve the capability and the overall result the trajectory is further discretized. The result is a trajectory composed by n concatenated moves: Figure 4.11 and Figure 4.12.

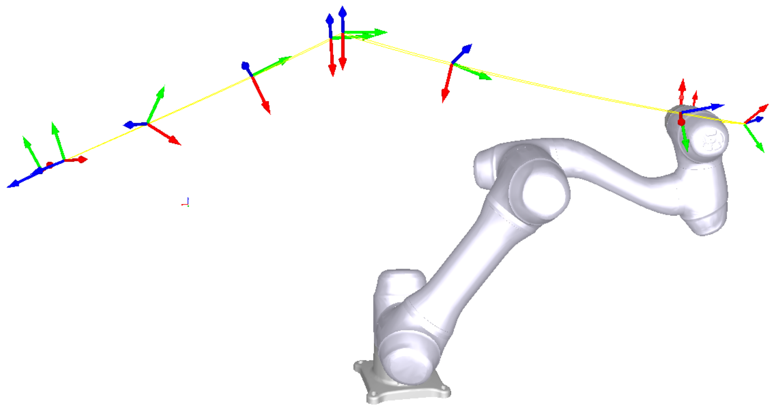


Figure 4.11: Trajectory of a pick and place divided in n segment

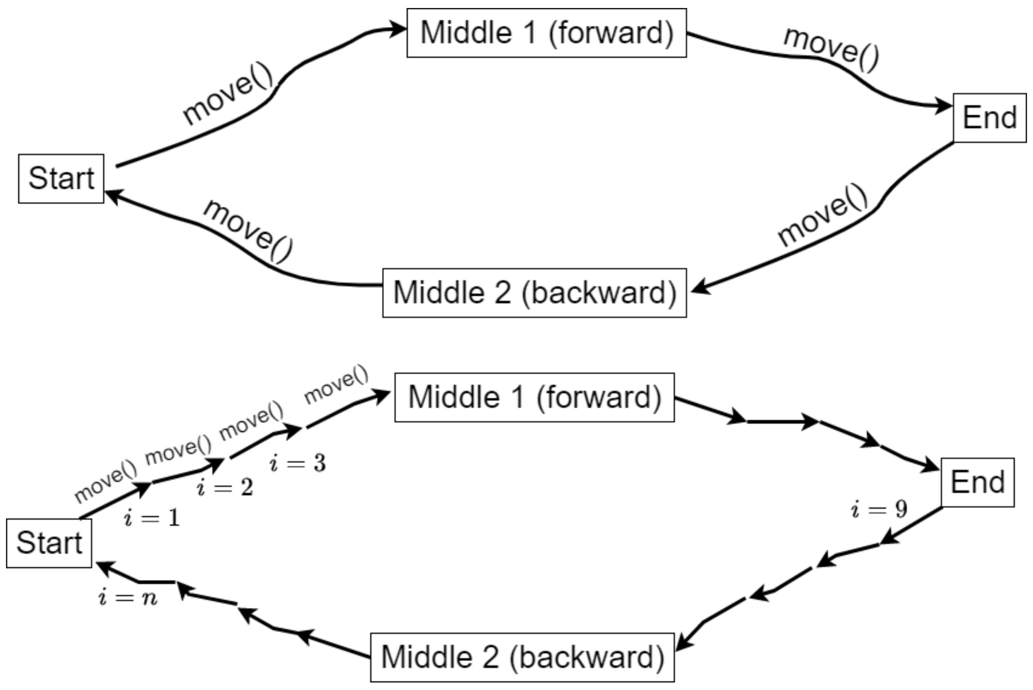


Figure 4.12: Representation of the subdivision of the trajectory into n sub-segments, but keeping the path unchanged

Inside the simulation algorithm the minimum distance between the person and the robot links is computed once for each trajectory sub-segment $i = 1, 2, ..., n$. This distance is then used to define the speed of each sub-segment.
The maximum speed will be proportional to the distance according the following modified version of the Equation2.1

$$v^i_{max} = \frac{1}{T_B} k d^i_{min} \tag{4.10}$$

where $v^i_{max}[mm/s]$ is the maximum velocity of the i^{th} sub-segment.
The value of $d^i_{min}[mm]$ is the minimum distance calculated between the robot configurations in the i^{th} sub-segment and the human. The time $T_B[s]$ is computed from the sum of reaction time of controller and braking time of the robot, it is generally provided by the robot manufacturer. This kind of reasoning is generated by the speed separation monitoring algorithm suggested by the ISO-TS-15066 with a main difference: the ISO standard requires the presence of sensors in the field that measure the actual distance of the operator from the base of the robot in real time. In our scenario instead there is only a probabilistic

study: there is no certainty that the person is not in proximity of the cobot, but only a high probability. An unintentional collision is more probable with respect the SSM. For this reason the safety scaling factor $k < 1$ is added to make a possible accident more acceptable, reducing the velocity with respect the SSM approach. In Figure 4.13 it is shown how our algorithm is more conservative than SSM. The safety scaling factor k must be chosen both on the basis of the reliability of the movements of the person and the level of risk and damage that can cause an unintentional contact.
The actual k to be used in the task must be computed during the risk assessment according to the data provided by the robot manufacturer and the ISO standards. In our experiment a value of $k = 0.175$ is considered.

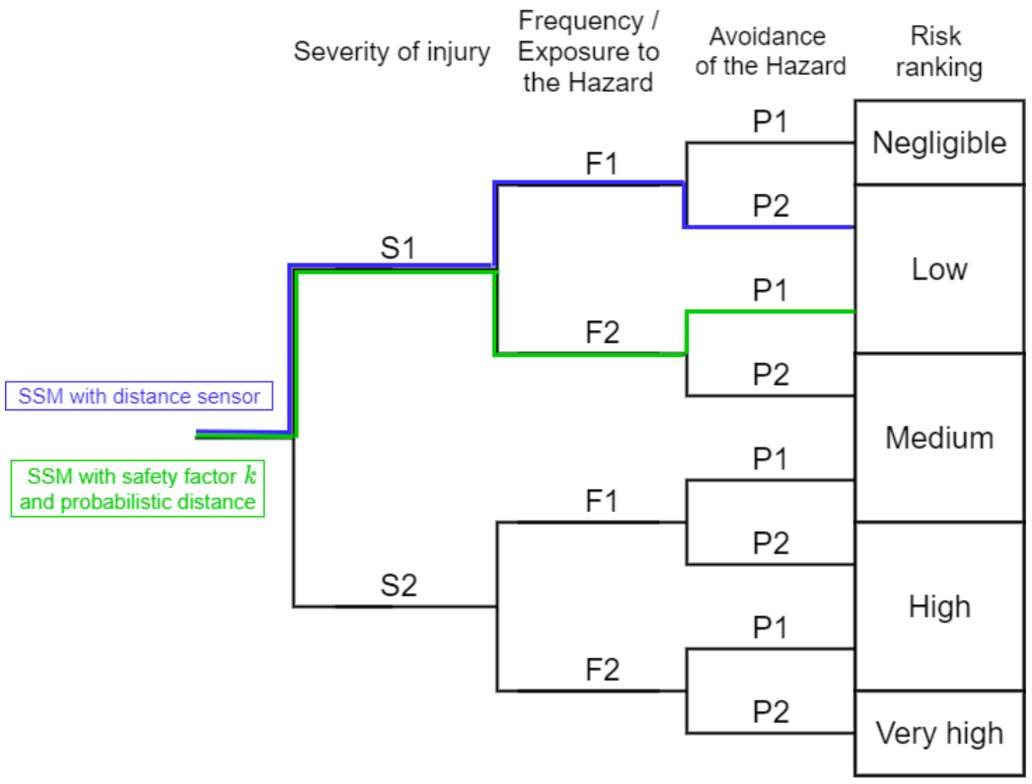
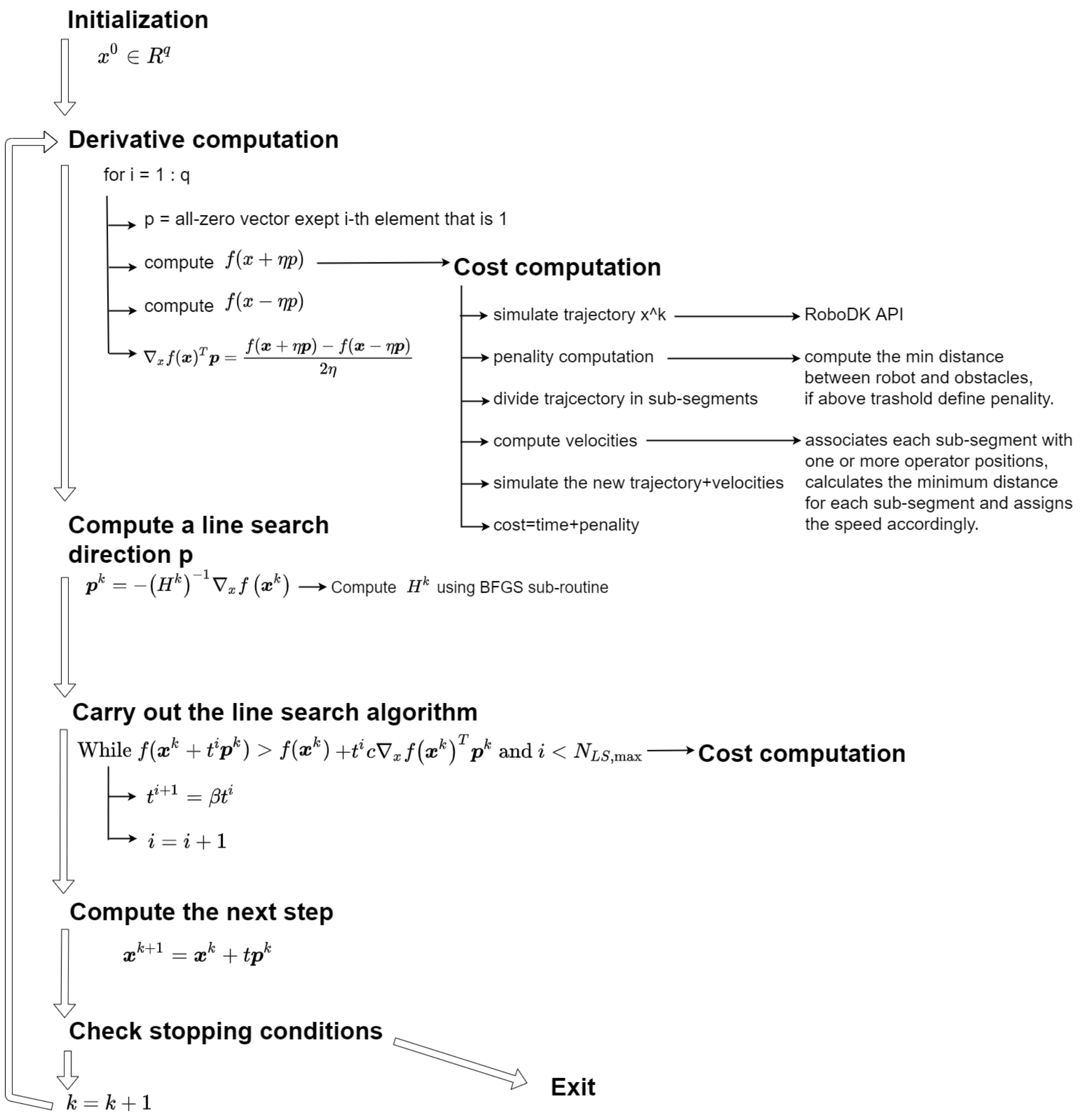


Figure 4.13: With reference to the Chapter 2 the diagram show the different path taken by SSM with sensor and SSM probabilistic but more conservative: in the first one the collision is very unlikely to happen, but in case of collision it is more dangerous and difficult to avoid, in the second path the collision is more probable but it is less dangerous and easy to avoid thanks to the lower velocity of the robot

4.2.5 Complete overview

In order to give a complete overview of the algorithm, a block diagram was created so the purpose and order of execution of the various processes are clearer.


Initialization
$x^0 \in R^q$
Derivative computation
for i = 1 : q
p = all-zero vector exept i-th element that is 1
compute $f(x+\eta p)$
compute $f(x-\eta p)$
$\nabla_x f(\boldsymbol{x})^T \boldsymbol{p} = \frac{f(\boldsymbol{x}+\eta\boldsymbol{p}) - f(\boldsymbol{x}-\eta\boldsymbol{p})}{2\eta}$
Cost computation
simulate trajectory x^k
RoboDK API
penality computation
compute the min distance between robot and obstacles, if above trashold define penality.
divide trajcectory in sub-segments
compute velocities
associates each sub-segment with one or more operator positions, calculates the minimum distance for each sub-segment and assigns the speed accordingly.
simulate the new trajectory+velocities
cost=time+penality
Compute a line search direction p
$\boldsymbol{p}^k = -\left(H^k\right)^{-1} \nabla_x f\left(\boldsymbol{x}^k\right)$
Compute H^k using BFGS sub-routine
Carry out the line search algorithm
While $f(\boldsymbol{x}^k + t^i\boldsymbol{p}^k) > f(\boldsymbol{x}^k) + t^i c \nabla_x f\left(\boldsymbol{x}^k\right)^T \boldsymbol{p}^k$ and $i < N_{LS,\max}$
Cost computation
$t^{i+1} = \beta t^i$
$i = i+1$
Compute the next step
$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + t\boldsymbol{p}^k$
Check stopping conditions
Exit
$k = k+1$

Chapter 5

Human model in the optimization algorithm

We will now present three different methods to interpret the human occupancy in the workspace and thus the optimisation algorithm.
The operator positions in the workcell is modelled as a sequence of point-cloud sampled at a constant sampling time. For each second we have 8 three-dimensional points, representing head, shoulders, elbows, hands and torso (Figure 6.5). The acquisition procedure will be explained in detail in the next Chapter.
Before entering into the details of the different methods, the industrial process under consideration will be presented.

5.1 Industrial Process: task presentation

This process is a facsimile of an actual process implemented in a boot industry. The purpose of the process is to produce boot soles from rubber sheets. The machinery present is a press and a cobot.
The operation steps are:

1. the operator approaches the press and extracts the soles from the mould, removing any residue;

2. the cobot sprays some anti-adhesive liquid on the empty moulds; in the meantime, the operator moves the finished soles to a nearby workbench and refines the shapes using a cutter;

3. the cobot places the plastic logos inside the moulds, which will be incorporated during the moulding process;

4. the operator returns to the press and inserts a rubber sheet inside the press;

5. the operator moves back in order to turn on the press.

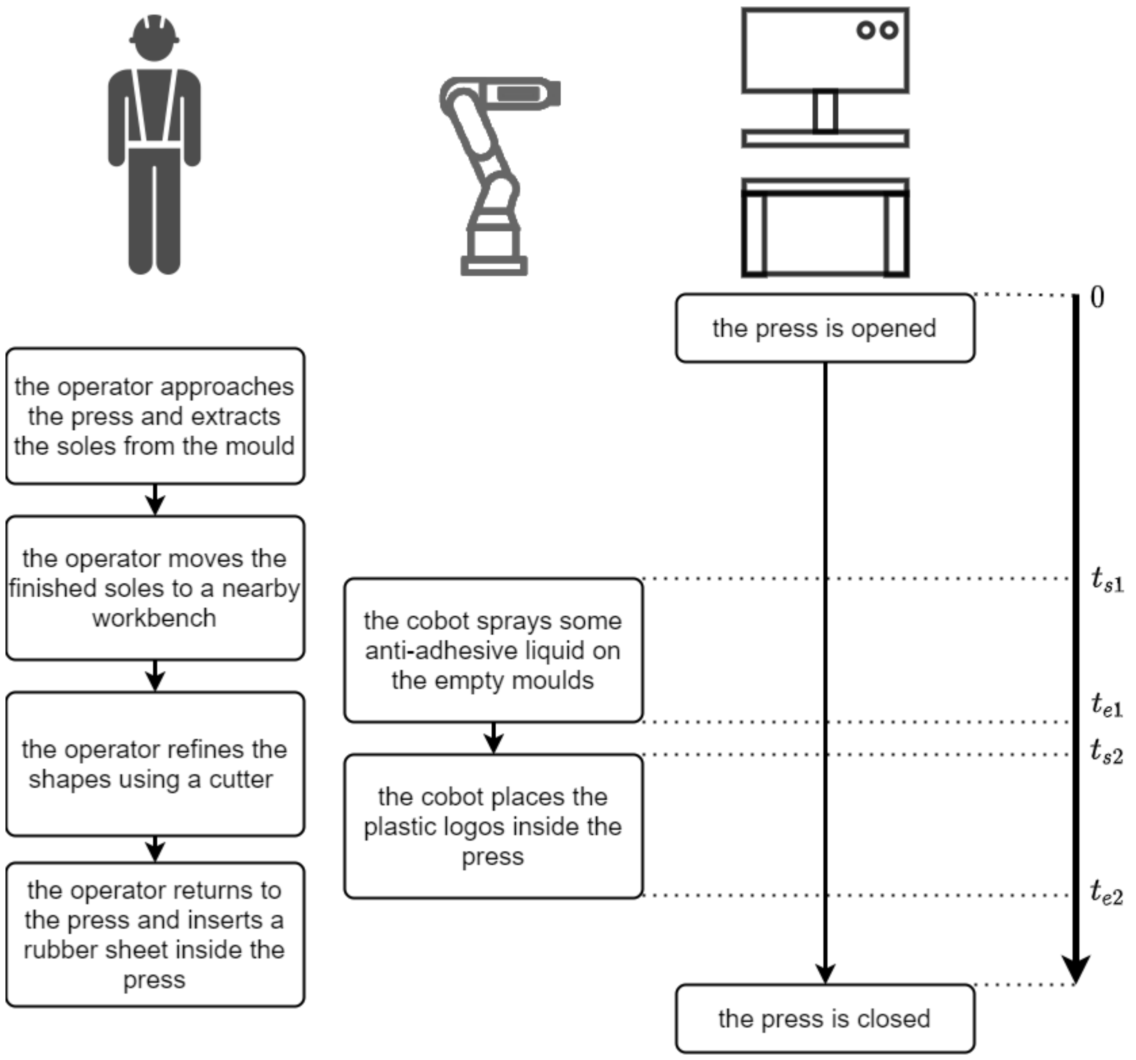

the press is opened
0
the operator approaches the press and extracts the soles from the mould
the operator moves the finished soles to a nearby workbench
t_{s1}
the cobot sprays some anti-adhesive liquid on the empty moulds
t_{e1}
t_{s2}
the operator refines the shapes using a cutter
the cobot places the plastic logos inside the press
t_{e2}
the operator returns to the press and inserts a rubber sheet inside the press
the press is closed

Figure 5.1: Workflow of the operator, the cobot and the press. Definition of t_s and t_e as the pick and place operation start time and end time.

From this workflow it can be seen that there are two pick and place operations, one for placing the logo and one for spraying the release liquid. The last stages of the cobot's approach to the mould, positioning the logos, changing the end effector and spraying are not discussed here, as they are predefined movements carried out at very low speed. Instead of these activities the cobot will stand still and wait for a period of time. What we are interested in optimising is instead the trajectory that the cobot makes from the table where the logos and end effectors are located to the press and back.

5.2 Static human volumetric sweep

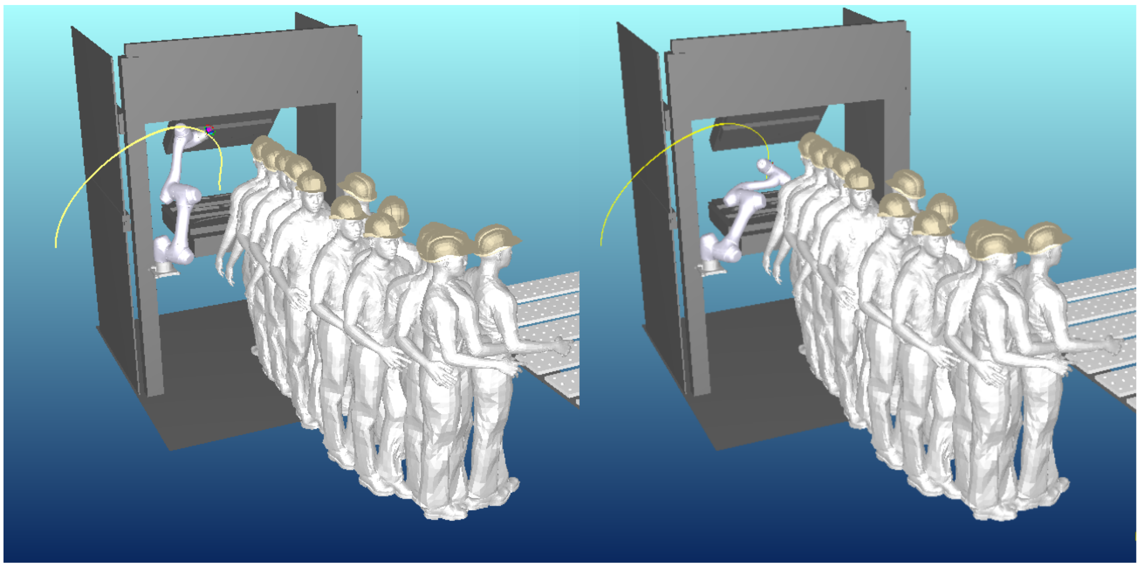


Figure 5.2: In each point of the robot trajectory the algorithm will consider all the operator's positions.

With this approach we consider all the spheres and cylinders that constitute the operator throughout a work cycle as a single volume. In a way this approach returns a reachability domain of the operator, i.e. the space it can reach and will probably occupy during the whole task. Since the operator moves around and occupies different areas during his work, the result is a single volume from which the algorithm calculates the distance.
It can be imagined as many men standing still in the path of the operator (Figure 5.2) with the algorithm looking for the shortest distance between the robot and

all of them.

By optimizing we obtain a trajectory that will try to avoid any possible contact with the operator, thus lowering the probability of an unwanted collision. Moreover the robot will slow down progressively when it approaches the area that can be occupied by the operator. It can be visualized as an aura that starts from the volume of each operator's volume and goes to define a sequence of concentric volumes in which the robot can move at different speeds. Differently from the Figure 2.8 where the human moves in different zones and the robot must adjust its speed, here the operator is consider still and the robot trajectory pass through different zones:

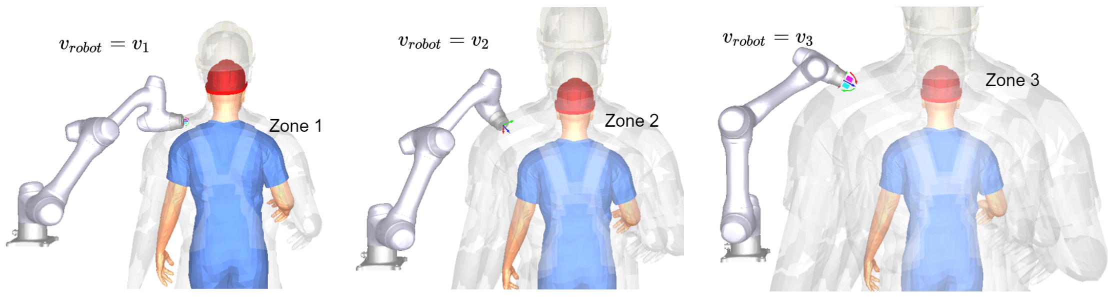


Figure 5.3: Visualization of the distance between robot and operator as concentric volumes: the closer the robot, the slower it must go

In the algorithm used, these robot speeds will not be discretized in zones, but will follow a continuous function, as shown in Figure 2.9.
The optimization algorithm will therefore automatically choose the paths that are furthest from the person, because they will allow a higher speed and therefore a shorter task time.

The advantages of this static human approach are multiple. The optimization will consist of only one point for each pick and place: start and end will be constant, while the intermediate configuration to be optimized are $x \in \mathbb{R}^6$. This is because having only obstacles and a static operator there is no reason why the optimal path in one direction is not also optimal in the return.
Another advantage is that at any time the cobot moves it will follow the most prudent trajectory and will slow down approaching the area occupied by the person

(even if the operator is far away).
In addition, the number of volumes considered can be reduced to the ones close to the robot, as the ones far away do not influence the process in any way. This reduces the computational effort to compute the distances.
Finally, the disadvantage is that having a very conservative movement does not fully exploit the times when the person is actually very far from the cobot. This is because this first algorithm does not take into account the *temporality* of movement, but only *spatiality*.
For this reason a second approach has been studied

5.3 Dynamic human model

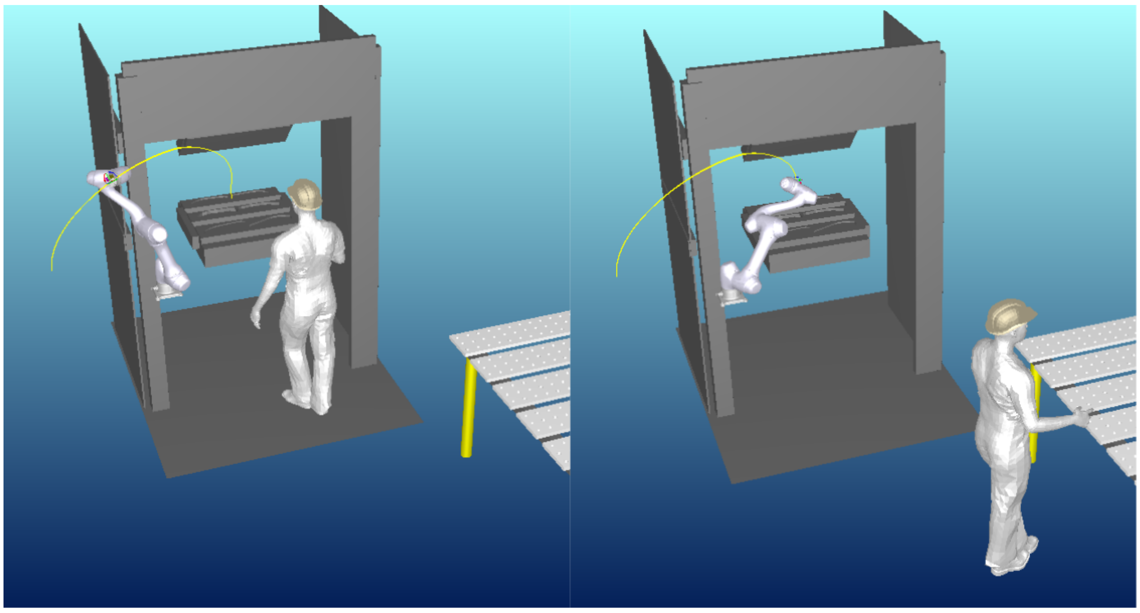


Figure 5.4: At each segment of the cobot's trajectory, the algorithm computes the distance between the robot and the operator's position at that second. When the operator is far the robot will move at a higher velocity.

If it is assumed that the task of the worker is repetitive and predictable with respect to the measurable times of the machines present in the work cell then one can exploit this assumption to further improve the performance of the cobot. Instead of considering a swept volume where it is very likely to find the operator,

the human can be considered as a moving obstacle that follows a known trajectory.
It is useful to consider separately the movements the man makes in the time of the first pick and place and those he makes in the second.

Unlike the static method, now it will be necessary to consider as variables to be optimized both the intermediate point of inward and the backward movement $x \in \mathbb{R}^{12}$. This is because during the time elapsed between one and the other the person will have moved and therefore the optimal path will be different.
With this different modelization of the human, there will be a different procedure for the distance calculations with respect to the first algorithm.
In order to know the distance between each part of the robot's trajectory and the man, it is first necessary to synchronise the times. We need to know what position the person is standing when the robot performs the sub-segment i. Thus for each sub-segment $i = 1, 2, ..n$ the time t_i should be calculated. This time t_i is the absolute time from the beginning of the process (considered as the time of opening the press) to the robot moving on the i^{th} sub-segment. To calculate this value consider $t_i = t_s + t_r$ where t_s is the time elapsed from the opening of the press to the start of the robot movement, while t_r is the time elapsed from the start of the robot movement to the start of the sub-segment undertaken. This time t_i is then rounded to the seconds. Each second corresponds to a different human configuration. Now we have the cobot position at the i segment and the corresponding human position. Finally, the distance between the robot at each configuration of the i^{th} sub-segment and the operator of time t_i is calculated using the method seen above [41]. The maximal allowed robot speed consistent with what we said in the previous chapter is calculated as in the Equation 4.10.

The advantage of this method is that the robot will work faster when the operator is likely to be further away and slower when he is closer. This improves performance and reduces the total task time.
One problem is that the operator will hardly follow exactly the same movements and timing of the one taken in the data acquisition, so the data will be only approximated.

5.4 Time-window human model

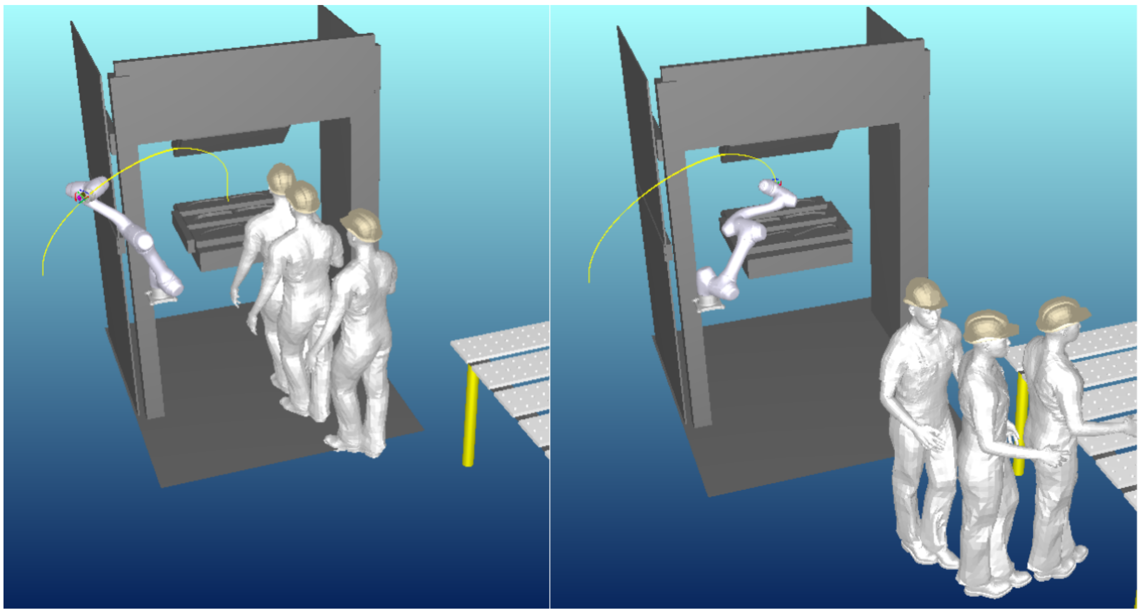


Figure 5.5: At each segment of the cobot's trajectory, the algorithm calculates the minimum distance between the robot and three operator positions: at that second, one second before and one second after.

A third approach born from the merge of the two previous ones can mitigate the disadvantages of both. Instead of considering one man at a time in the distance calculation and velocity definition, one can consider a sliding window of time.

During each iteration of the velocity evaluation, the distance between the robot configurations in an i^{th} sub-segment and the operator is measured. Instead of considering the operator as unique volume of all reached positions as in the first method, and instead of considering only the person of time t_i as in the second one, this third approach considers the configurations of the person at the times $t_i - w, \dots, t_i, \dots, t_i + w$ as shown in Figure 5.6.

A reasonable window considering the robot's timing is $w = 1$ (Figure 5.5).

This has the advantage of having a certain clearance in the movement of the person, in fact there are $2w + 1$ seconds of window, so it is also taken into account the possibility that the human moves a little slower or a little faster than the basic measurement.

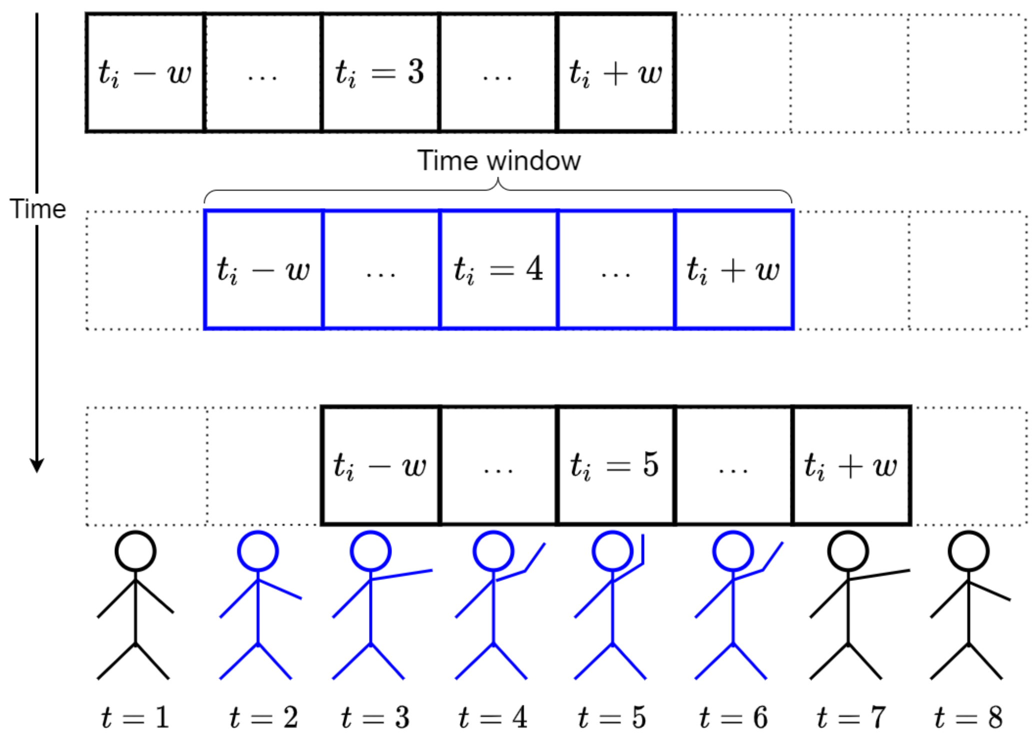


Figure 5.6: Visualization of the time window sliding and considering different operator configurations

At the same time, however, there is a dynamism that allows the cobot to work faster when the operator is far away than when he is closer.

5.5 Warm starting the complete trajectory optimization

The complete trajectory for the industrial task requires two pick and place. In the first algorithm optimizing one pick and place corresponds to optimizing both, since the conditions do not change (the full volume is always considered) the optimum also remains unchanged. It is different for the second and third methods: depending on the time at which the pick and place should be started the distances to the operator will be different. therefore it is necessary to separately optimize both pick and places, defining the start time t_s properly.

Since the second pick and place is actually conditioned by the end time of the

first one, it is appropriate to make a further collective optimisation of the pick and place in sequence. This optimisation will have as its trajectory the sum of the two activities:
start→ **middle1**→ end→ **middle2**→ start→ **middle3**→ end→ **middle4**→ start
in this case the optimization involves 6x4 values (middle1 middle2 middle3 middle4), so $x \in \mathbb{R}^{24}$. This greatly lowers the efficiency of the algorithm and its ability to arrive to a solution, in fact with a random initialization it is very likely that it will not converge. For this reason it is better to use a technique called "warm starting"[42]. This technique consists of first optimizing separately the two pick and places and then initializing the overall problem with the results of the single pick and place optimisations. The results of this collective optimization are discussed in Chapter 7.

Chapter 6

Data acquisition: Virtual Reality and Kinect sensor

In this Chapter we are going to analyse the data acquisition process, which have the goal of obtaining the operator occupancy area in the workplace in the form of a point-cloud.
Since one of the aims of this thesis is to prove the concept of virtual commissioning, the real plant and press were not used (since they are hypothetically not yet in place) but virtual reality was implemented.
This type of approach has multiple benefits:

- it allows a preliminary study of the tasks the operators movement before the actual implementation of the cell;
- it does not require stopping or slowing down industry production, which is always economically problematic;
- it can be performed in any given location, and requires only the operator and two devices (a virtual reality visor and a depth camera).

6.1 Virtual Reality

One of the key areas of Industry 4.0 growth and development is virtual reality. Through VR, exact simulations of products, processes or production plant can be

(a) Virtual reality device: Facebook Oculus Quest 2

(b) Data acquisition device: Microsoft Kinect 2

Figure 6.1: Devices used in the data acquisition process

built to enable a different way to see their operation in person and in an immersive mode. Engineers can monitor progress in a more visual and interactive way through virtual simulation for product or process design and prototype validation. In return, errors at this stage can be reduced and productivity improved. Augmented and virtual realities can be adopted to help integrate the human workforce in the manufacturing system [43].

In our use case it was necessary to virtually simulate the work cell with the press and the surrounding environment. The first step was to design the press for moulding the soles (Figure 6.2) using a 3D modelling software (AutoCad®). Then using Unity we developed an executable Android application. Unity is a cross-platform graphics engine that is developed by Unity Technologies®. This software enables the development of video games and other interactive content, such as architectural visualisations or real-time 3D animations [44].
The procedure involves drawing the scene of the industry as if it were a video game, inserting the press and other necessary 3D elements (Figure 6.3). Then an android application is generated from Unity and directly loaded into the visor

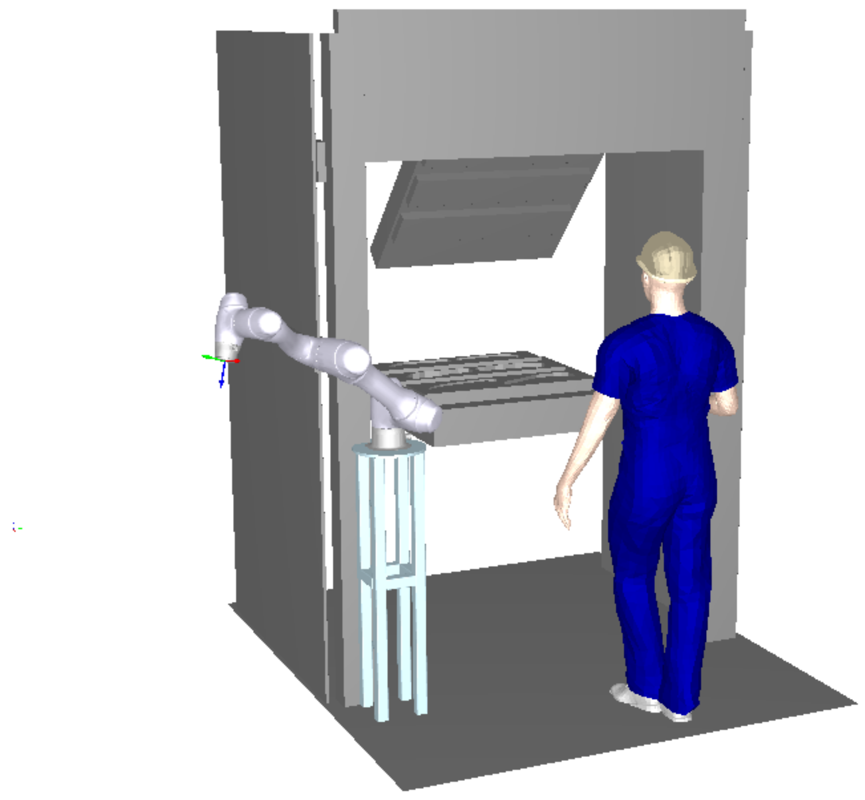


Figure 6.2: 3D representation of the press, the cobot and the operator

via USB-C cable. The visor used in this project is the Facebook® Oculus Quest 2 (Figure 6.1a). Once this process has been carried out it is possible to use the Oculus Quest 2 as a stand alone device. This allows the operator to simulate a complete working scene without being restricted by a cable. In addition, the visor can be programmed to display a reproduction of the user's hands in the scene (Figure 6.4). This is possible thanks to high fidelity tracking functions utilizing four small cameras positioned directly on the visor [45]. This allows you to have a much more immersive experience, seeing your hands directly on the scene and thus being able to simulate with greater fidelity the task that the operator is required to perform.

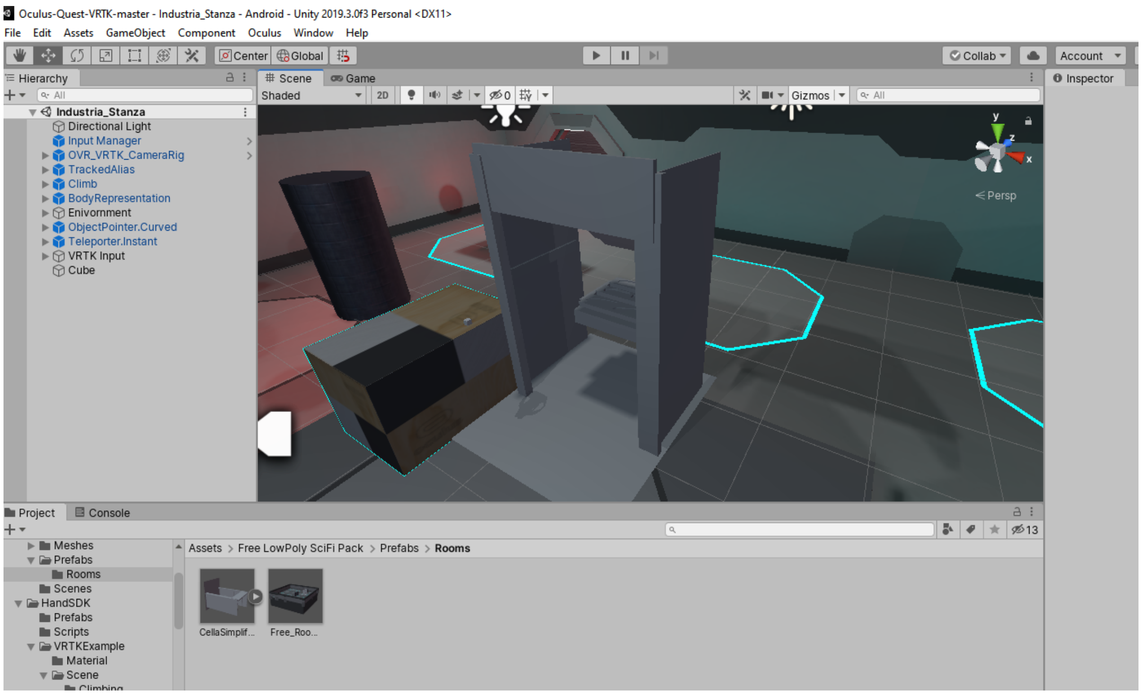


Figure 6.3: Unity editor with the current working cell and press

6.2 Environment setup and data acquisition

A human-tracking device is required to record the data on the movement of the worker. The Kinect is a device developed by Microsoft® for motion detection (Figure 6.1b). This device integrates an RGB camera with an infrared projectors and detectors that allow to acquire a depth map of the environment.
The sensor measures the amount of time the light takes between being emitted and being picked up by the camera after hitting an object. This technique provides a point-cloud of the environment. Then, through the random forest algorithm already integrated in the Kinect SDK, the recognition of gestures and the detection of the body skeleton is carried out. In particular it is possible to obtain the recognition of the 3D position, with respect to the reference system of the camera, of different parts of the body [46]. We are interested in: head, shoulders, elbows, hands and torso (Figure 6.5).

The Kinect camera is positioned 1.20 metres above the ground in a horizontal position. Considering the speed of the operator's movements, the precision of the Kinect and the overall time of the tasks it was decided to set a position

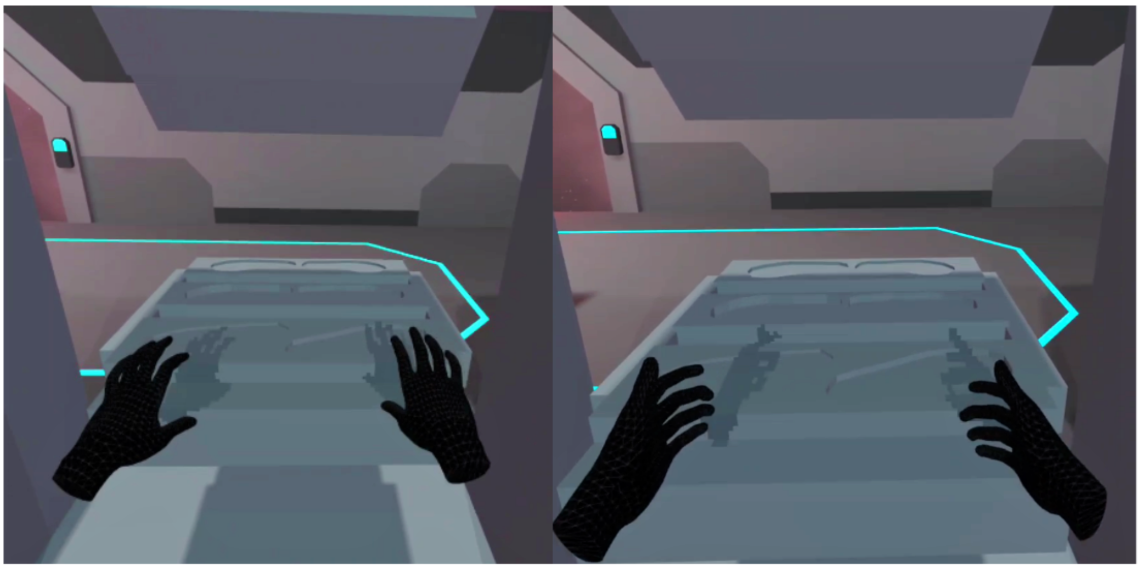


Figure 6.4: Actual screenshot of what the operator is seeing when wearing the visor.

acquisition every 1 second. The operator simulates the work in the industry by seeing himself in front of the press and completing the tasks. The first task is to approach the press and remove the soles, the second requires to bring the soles to the workstation and refine the edges, return to the press, and finally the last task is to place the rubber sheet and start the machine.

Once the acquisition was completed, a post processing operation was needed. The distance between the hand and the elbow was wrong in two operator model acquired, so a correction of the outliers was carried out. In particular, the defects were found to be greater when the operator was facing backwards, because the Kinect is not designed for the acquisition of people turned.

Once this phase was completed and the cloud-point of the operator's positions were obtained at one-second intervals, it was possible to start the optimization algorithms.

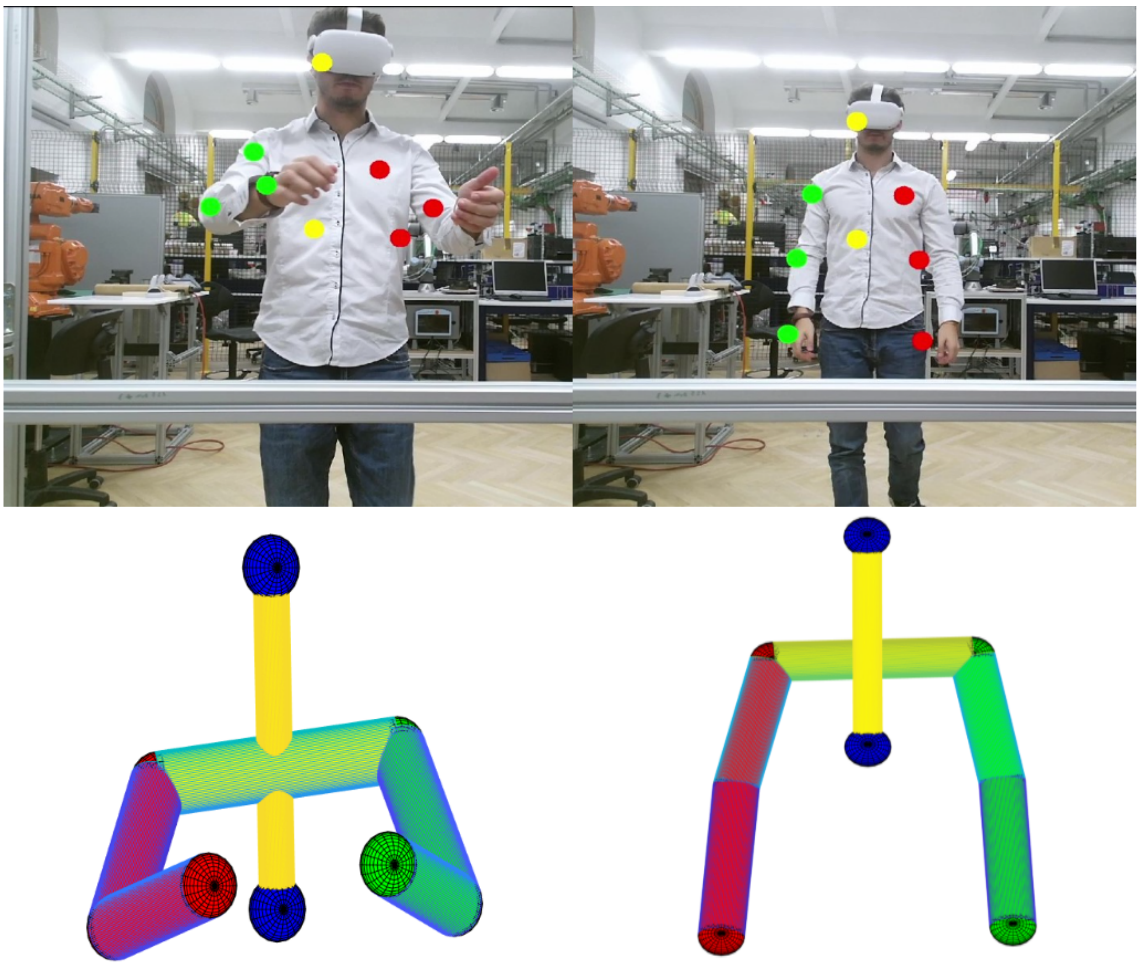


Figure 6.5: Kinect acquisition output and Matlab representation of the operator

Chapter 7

Experimental Results

In this chapter we are going to examine the results of the optimization, first by analysing the output of the algorithm and then by comparing the performance obtained in a series of experimental tests. These experiments are carried out with the co-presence of the operator and the cobot in an environment that emulates the final cell. To better appreciate the differences between these methods three different indexes are defined.

7.1 Initialization

Since there are several local minima, as explained in Chapter 4, several optimization attempts must be made. The optimization result that ended up in the lowest cost configuration is finally chosen.
Regarding the initialization of the decision variable, i.e. the choice of the intermediate configurations (Figure 4.1), we proceeded with a random choice. This is a very common strategy in the optimization problems since it allows to have very different outcomes and it is not biased by the human decision. Each joint has been chosen as a random number between the initial and the final configuration angle.
The second and third optimization algorithm were carried out using the warm starting technique (Section 5.5) . More precisely, first the two pick and place problems are optimized separately and randomly initialized, then the results are

merged to form the initialization set for the general problem. Finally the complete optimization is performed (Figure 7.1).

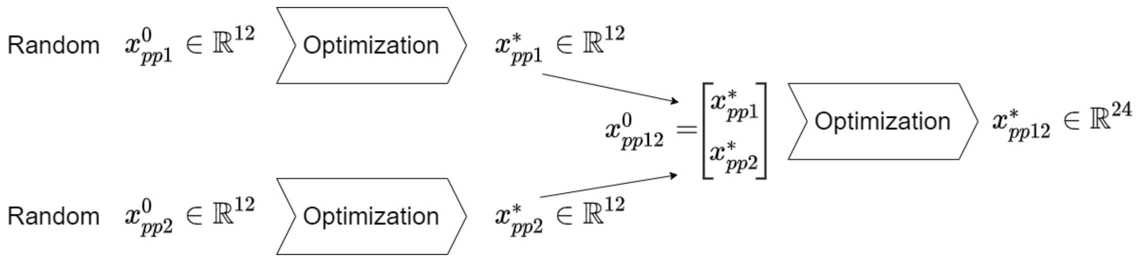

Random $x^0_{pp1} \in \mathbb{R}^{12}$
Optimization
$x^*_{pp1} \in \mathbb{R}^{12}$
$x^0_{pp12} = \begin{bmatrix} x^*_{pp1} \\ x^*_{pp2} \end{bmatrix}$
Optimization
$x^*_{pp12} \in \mathbb{R}^{24}$
Random $x^0_{pp2} \in \mathbb{R}^{12}$
Optimization
$x^*_{pp2} \in \mathbb{R}^{12}$

Figure 7.1: Visualization of the warm starting procedure

7.2 Optimization result

7.2.1 Static human volumetric sweep

This algorithm considers the distance between the robot and every operator position.

Therefore it is expected that the robot will go faster in the initial part of the trajectory, i.e. when it is further away from the volume spaced by the operator and slows down as it approaches the inside of the press. In the Figure 7.4 we can see that the highest speeds of the end effector occur in the first 2 and last 2 seconds. The trajectory of the cobot is avoiding the edge of the press while passing as far away as possible from the person (Figure 7.2). This is the most conservative method and results in one pick and place time of 22.5s. The time improvement from the initial trajectory time to the minima found by the algorithm is 22.95% (Figure 7.3), considering as initialization a constant speed of 20 deg/s.

There is no need for warm starting since the problem does not change between the first and second pick and place (they consider always all the human positions), so it is sufficient to make one representative pick and place optimization. The overall process requires two pick and place operations in sequence so the same trajectory is executed twice. The trajectory time is about $T = 2 \cdot 22.5 = 45$s.

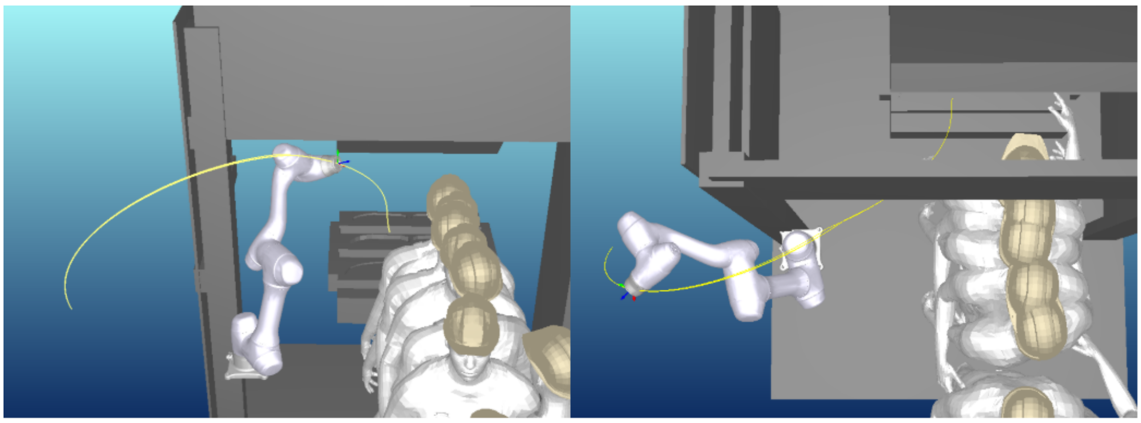


Figure 7.2: Two different view of the path of the first algorithm, Static human volumetric sweep

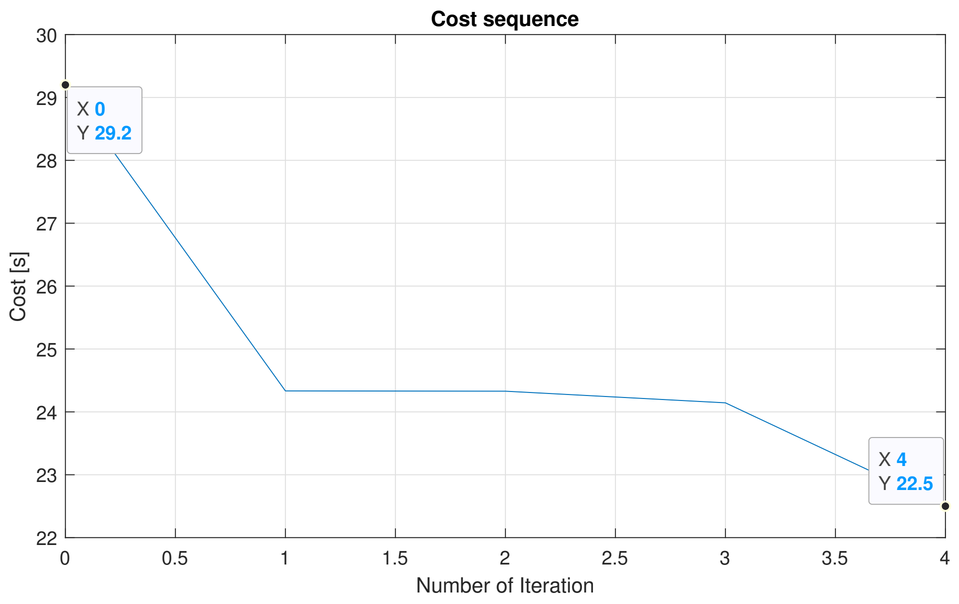


Figure 7.3: Cost sequence for the first algorithm: initialization at 29.2s and optimum at 22.5s

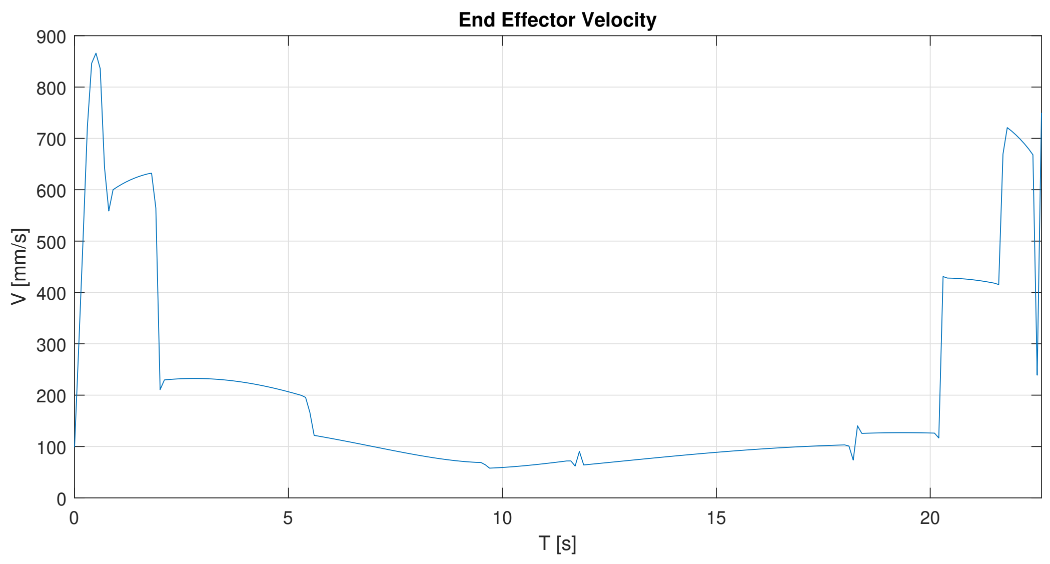


Figure 7.4: Norm of the end effector velocity *mm/s* during the first algorithm trajectory

7.2.2 Dynamic human model

This algorithm considers the distance between the robot and one different position of the operator for each second. The trajectory presented is constituted by the two consecutive pick and place optimized using the warm starting technique. The robot starts when the operator is finishing the process inside the press Figure 7.5 (1). At the beginning the algorithm knows that the worker position is near the press, so it will calculate a reduced speed. After a few seconds the man starts to move away from the press and the robot Figure7.5 (2). The movement of the cobot during this time frame will be in acceleration phase. When the worker stays away from the press for the time needed to refine the soles of the boots the robot will be able to go at a higher speed Figure 7.5 (3). Finally when the operator is returning to the press to insert the rubber sheet the robot will decrease gradually its speed Figure 7.5 (4). These speed variations (Figure 7.6) are made by measuring the minimum distance between the robot and the recording of the operator's movement during the data acquisition phase.
Some interesting results of the automatic path planning are shown in Figure 7.5 (3): the robot first moves horizontally away from the press and then back to the starting configuration, this should be indeed the fastest path if the operator does not occupy the press position, but it would be forbidden if the operator was close to it.
The improvement between the initialization trajectory time to the algorithm output trajectory time is 43.8% (Figure 7.7). This improvement are greater than those of the first algorithm because this technique is less restrictive: the robot can go faster for the whole time the operator is away in fact this is the less conservative method; it assumes that the operator moves in a very predictable way. The algorithm exploits every second that the operator is engaged in the other distant task to move the robot faster. The total trajectory time is $T = 27.09$s.

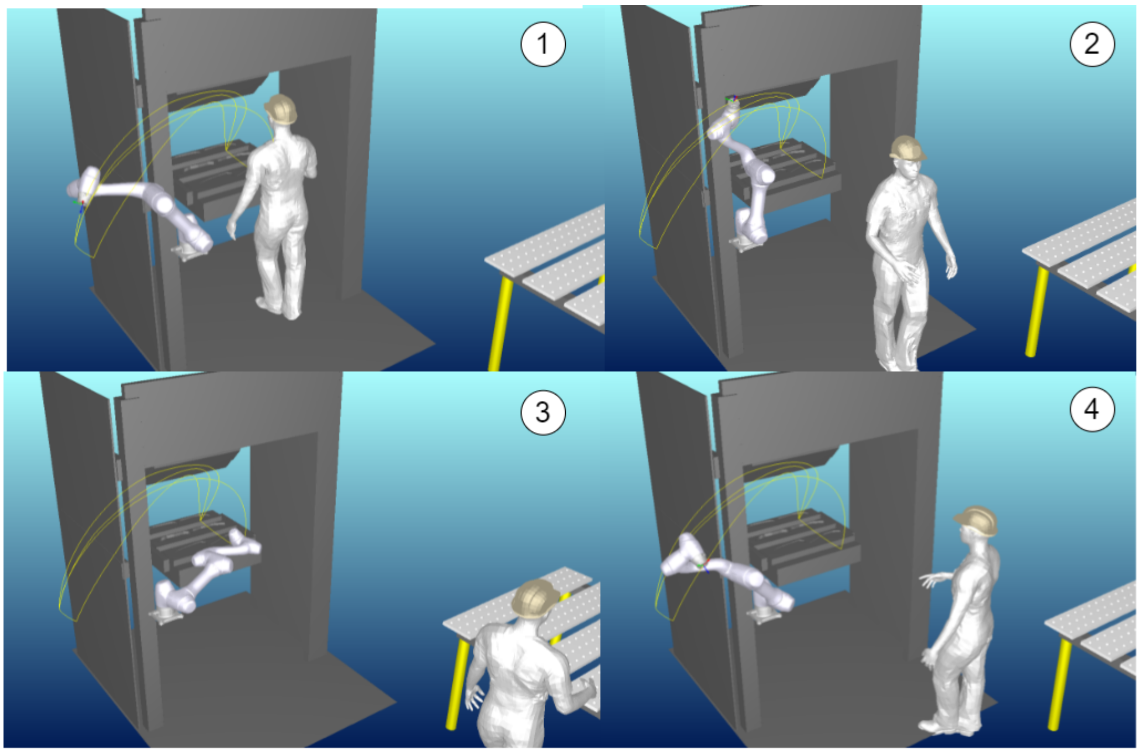


Figure 7.5: Four different view of the robot path and the operator position considered, Dynamic human model.

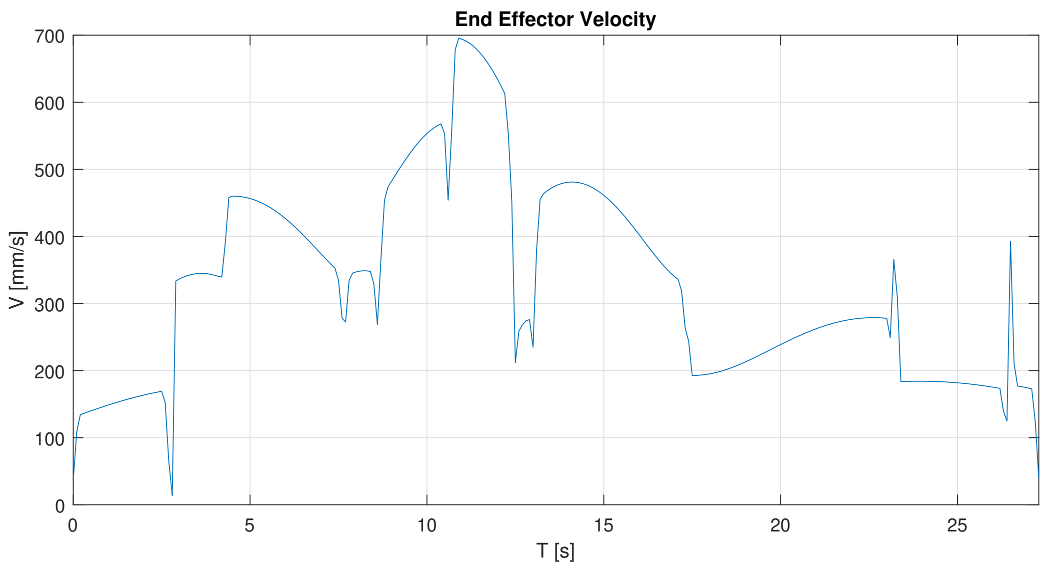
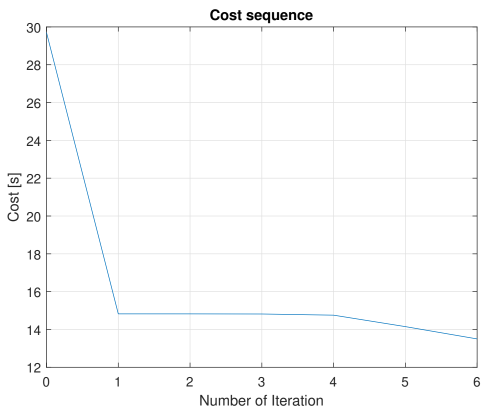
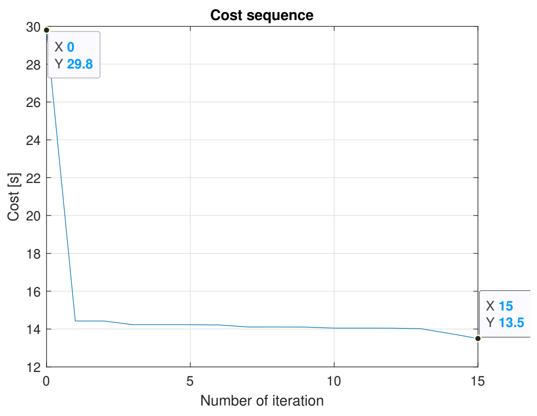


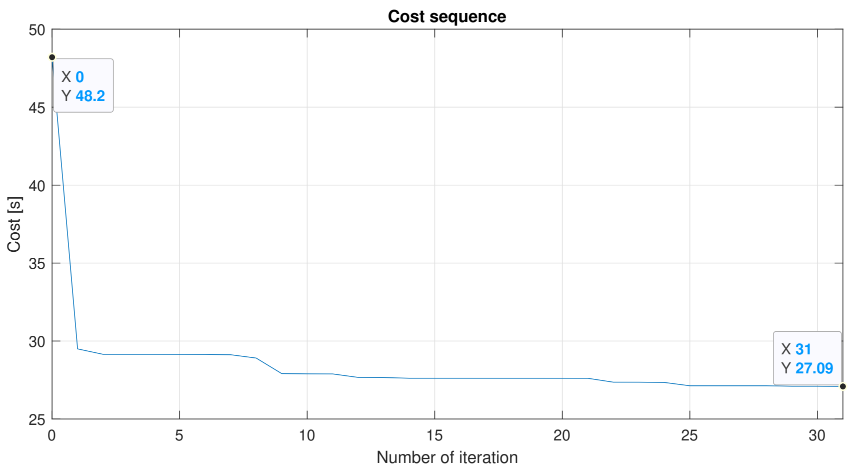
Figure 7.6: Norm of the end effector velocity *mm/s* during the second algorithm trajectory



(a) Cost sequence for the first pick-and-place



(b) Cost sequence for the second pick-and-place



(c) Cost sequence of the overall trajectory

Figure 7.7: Cost sequences for the second method

7.2.3 Time-window human model

The last method is the one that considers three operator's positions each second, more precisely, it also evaluates the distance of the operator one second before and one second after with respect to the previous method.
The movement of the robot, with its accelerations and decelerations will be similar to the second method, but in general more conservative. For example, the algorithm will take into account that the man could stay one second longer near the press, so it will make the robot go slower in the initial part Figure 7.8 (1). Similarly, this method will predict the man to return to his position shortly before he actually returns, thus starting to slow down in advance Figure 7.8 (4). When the worker is performing his tasks far away there is no significant difference with the previous algorithm so the robot will go at a high speed Figure 7.8 (2)(3).

The improvement between the initialization to the algorithm output is 35.74% (Figure 7.10). This improvement is in between the first and the second algorithm, considering that even if this method is more efficient than the first one it is also more conservative than the second. The total trajectory time is about $T = 34.7s$.

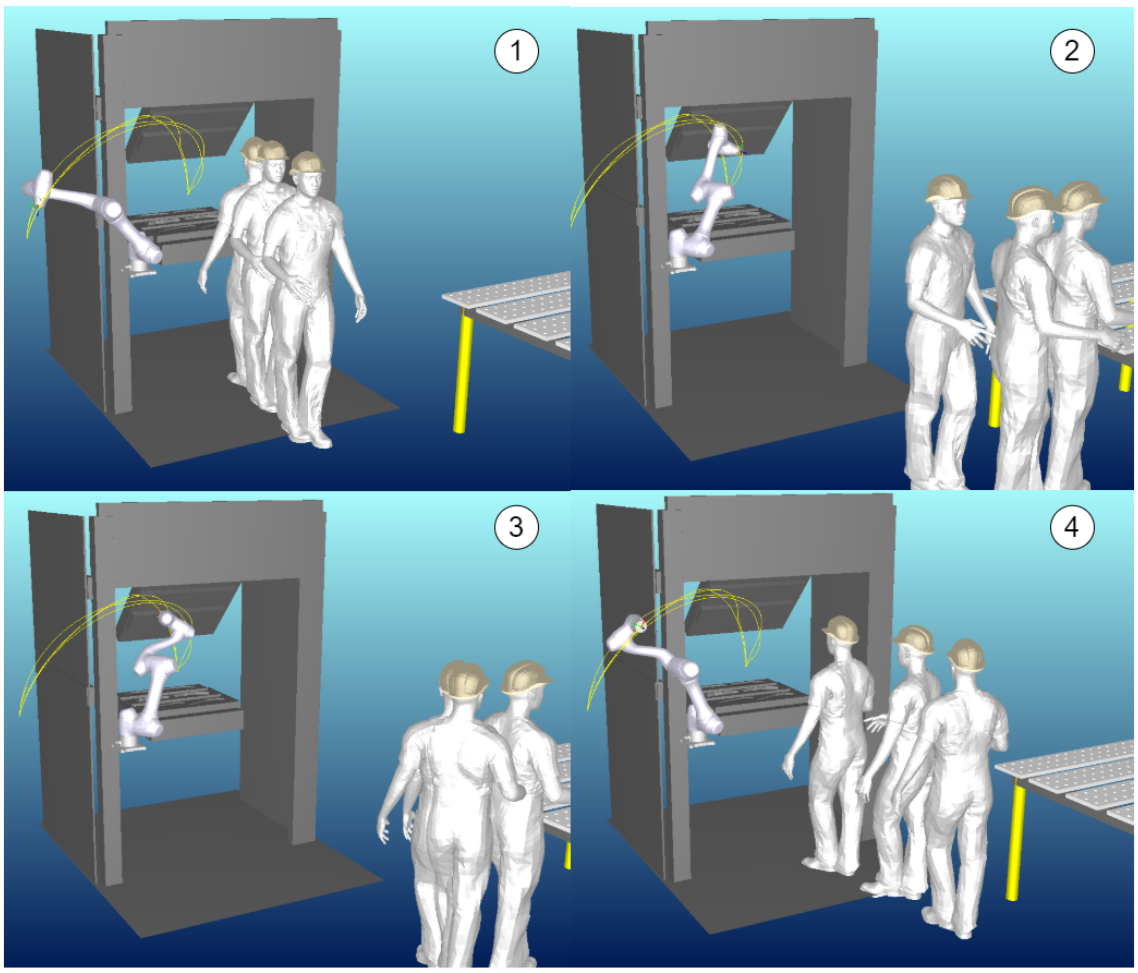


Figure 7.8: Four different view of the robot path and the operator position considered, Time-window human model.

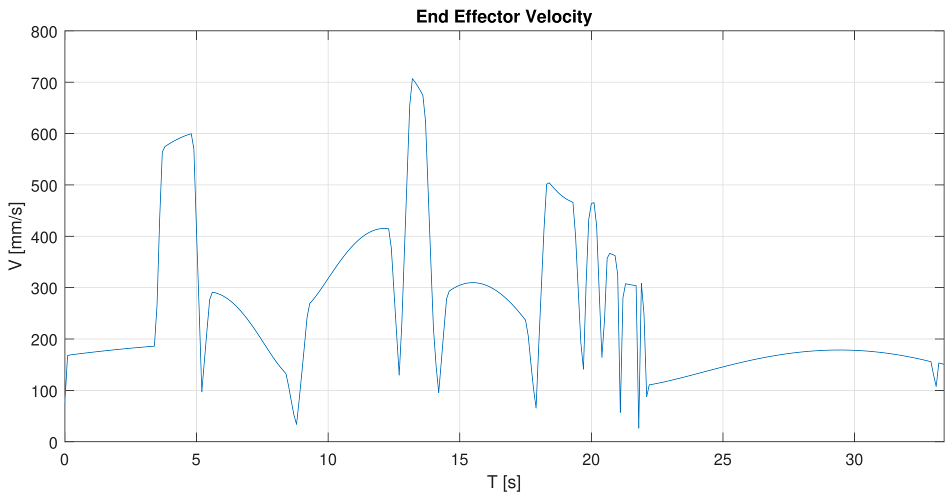
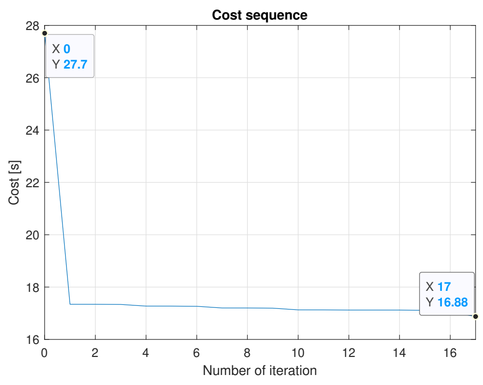
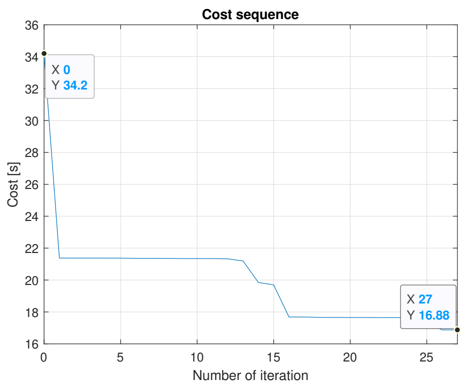


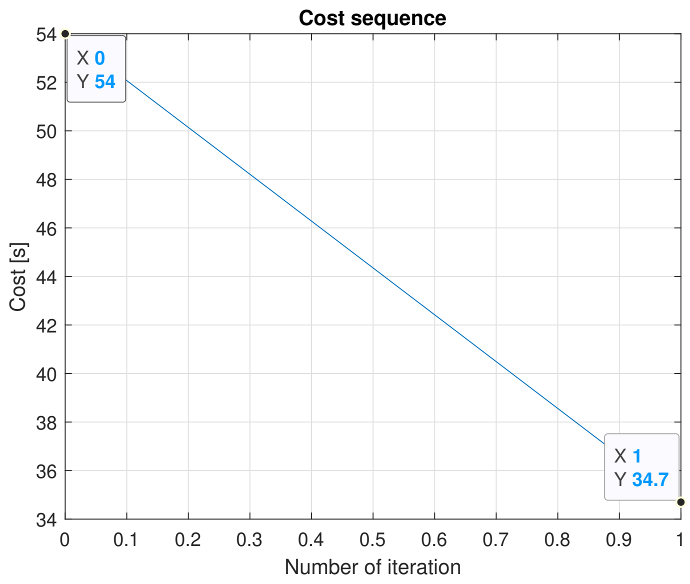
Figure 7.9: Norm of the end effector velocity *mm/s* during the third algorithm trajectory



(a) Cost sequence for the first pick-and-place



(b) Cost sequence for the second pick and place



(c) Cost sequence of the overall trajectory

Figure 7.10: Cost sequences for the third method

7.3 Experiment with cobot and operator

These experiments were carried out to verify in real life the quality of the trajectories generated by the three algorithms.
The challenge of this experiment is to faithfully represent the behaviour of an operator in the workplace. In reality, the worker cannot perform all the tasks precisely at the same time, but depending on various factors such as fatigue, boredom, etc., his speed may vary. Trying to reproduce this variability, the experiment was carried out with different operator executions.
From the data acquired it was found that the execution time of the first task (removing the soles) takes 12 seconds.
However, the experiments will also evaluate slower or faster movements of the operator to see how they interact with the robot's trajectory previously calculated.
The same thing is done for the second task (moving away and finishing the soles) which takes 35s.
To obtain this variance in speed, the operator's speed is increased by 10% and 20% and slowed down by the same proportions. In the experiments the operator's work was marked by a timer so that the time of each task could be varied accordingly to the speed that had to be tested.

In order to understand and quantify the improvement of the three algorithms a basic trajectory was needed. This trajectory provides a benchmark against which the algorithms can be compared. This trajectory is the simplest way to get the robot to move from the starting point to the ending point. For programming this operation the servo-assist training method was used (Section 2.1.1), i.e. the simplest and most immediate way to program the task. The speed was maintained at 20deg/s, i.e. the default speed set on the cobot. This trajectory will be named "Base".

In order to compare these 4 trajectories (Base, Algorithm1, Algorithm2, Algorithm3) a standard index must be defined.
The first one will be the cycle time, so the total time in between the opening and the closing of the press.

The second index is the energy transferred in a transient contact with the operator shoulder.
The last comparing index is the velocity variance admitted by the algorithm, i.e. how much the operator can vary its task time without colliding with the cobot.

7.3.1 Cycle time

Considering the operator at a standard speed, we will have the following cycle times:

Method	**Robot trajectory Time**	**Cycle Time**
Base	$54s$	$68s$
Algorithm1	$45s$	$60s$
Algorithm2	$27.1s$	$52s$
Algorithm3	$34.7s$	$52s$

Table 7.1: Comparison between Trajectory times and cycle times, i.e. from opening until closing the press

Those cycle times are directly taken from the experiments. The base trajectory and the first algorithm are particularly slow, so the operator has to wait for the robot to finish its task before inserting the rubber sheet and starting the press. This will increase the cycle time and add a downtime in the process. This is to be always avoided. The second and third algorithms have the same cycle time because when the operator returns he can directly start the press without waiting any longer. So in this comparison index the second and third algorithm are equally well performing.

7.3.2 Transient collision

To define this index, which is correlated with the safety, the energy dissipated due to transient contact is considered. This value is usually calculated as part of

the Power Force Limiting strategy. This criterion was used because there are no distance sensors as required by the SSM so in theory this task is subject to the PFL regulation, as contact could occur.
By making a simple risk assessment the worst case scenario was defined: the robot (with a payload of one and a half kilograms) hits the person on the shoulder. The shoulder is a relatively hard body part, it cannot distribute energy very well, therefore it has a low pain threshold. In our setup we will assume the person to be still, and we will calculate the robot speed relative to the shoulder in the following way: for each instant we will find the distance between the shoulder and the end effector, from this variation we will obtain the speed of displacement from the shoulder. Note this speed is different from the absolute speed of the end effector in Figure 7.6-7.9.
The index is calculated from the formula:

$$E = \frac{1}{2}\mu v_{\mathrm{rel}}^2 \tag{7.1}$$

where v_{rel} is the relative velocity between the end effector and the operator's shoulder when he is standing in front of the press, k is the elastic constant of the body part, μ is the reduced mass of the two-body system, obtained by $\mu = \left(\frac{1}{m_{\mathrm{H}}} + \frac{1}{m_{\mathrm{R}}}\right)^{-1}$ where m_H is the effective mass of the human body region and m_{R} is the effective mass of the robot. Both k and m_H can be found in the Table A.3 of the ISO-TS-5066 [1].
Finally it was possible to produce the graphs shown here:

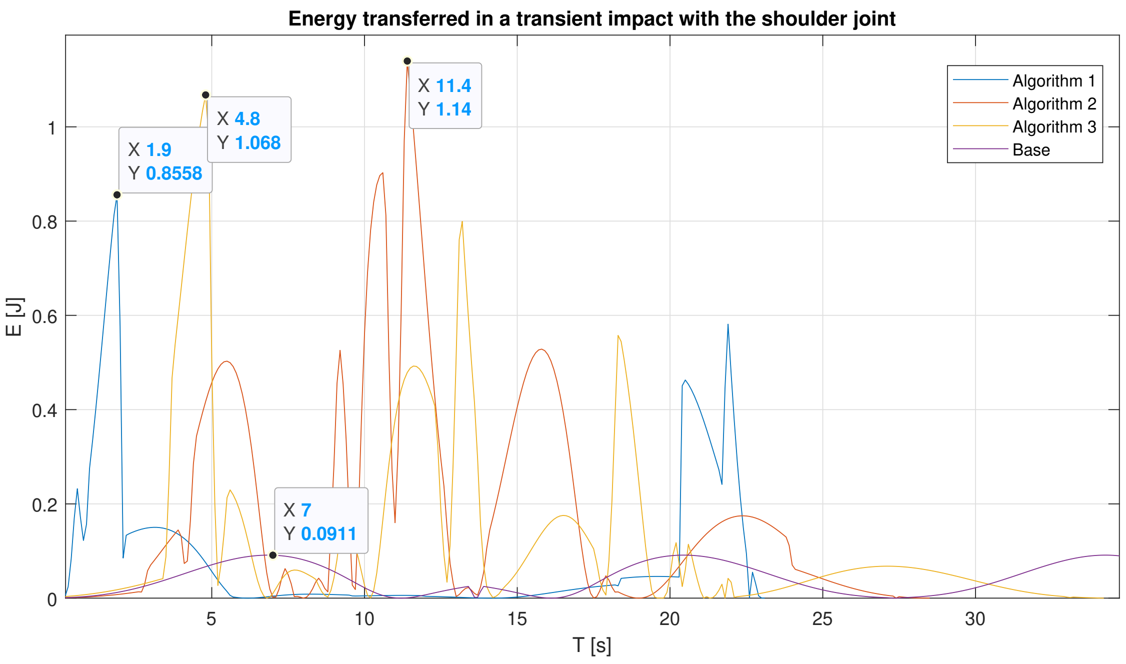


Figure 7.11: Visualization of the energy that can be transmitted in a transient contact between cobot and operator's shoulder. Note: the Base graph is truncated.

It can be observed that the worst case is represented by the values:

Method	**Maximum impact energy**
Base	0.091J
Algorithm1	0.856J
Algorithm2	1.14 J
Algorithm3	1.068J

Table 7.2: Maximum energy transmitted in the worst case scenario from the cobot to the operator shoulder in a transient collision

Clearly, the first algorithm, which is the most conservative one, has a low energy along with an equally low speed. On the other hand, Algorithm 2 and 3, have an higher energy transmission due to the faster movements. However, this increase on the danger is necessary in order to have reasonable execution times. In all cases, even if the operator were to find himself close to the robot at the moment of maximum energy exchange, this would represent an acceptable risk according to the FPL criterion. Indeed all the 4 trajectories are well below the ISO threshold of pain $E_{shoulder} = 1.46$ J calculated using the pressure limit associated with the shoulder joint (Table A.2 of the ISO-TS-15066 [1]) in a 1 centimeter square area.

7.3.3 Velocity variance

The last index considers the synchrony of movements, i.e. what is the speed variation that the operator can have before coming into contact with the cobot. Allowing a greater variance is preferred since it results in a more versatile trajectory that can perform well at a range of operator speeds.

For this index it is worthwhile to examine what is the maximum permissible decrease in speed. This is because if the operator slows down too much, the cobot could enter the press while the operator is still finishing the task. On the contrary, whenever the operator is faster than average, the only risk is that he will have to wait a few seconds before he can insert the rubber and switch on the press, but

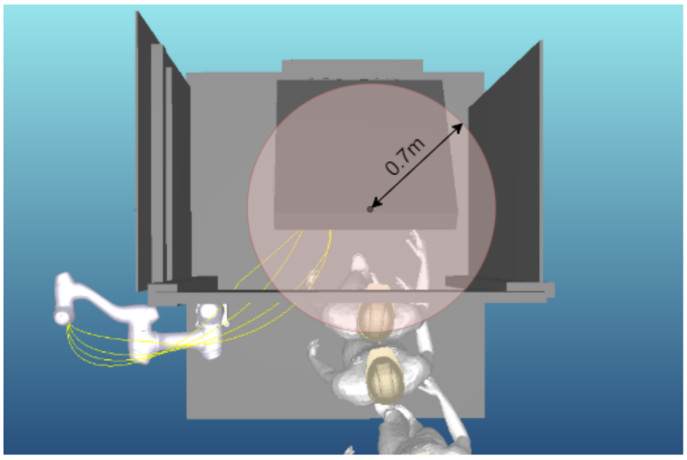


Figure 7.12: Visualization of the collaborative area, its center is inside the press.

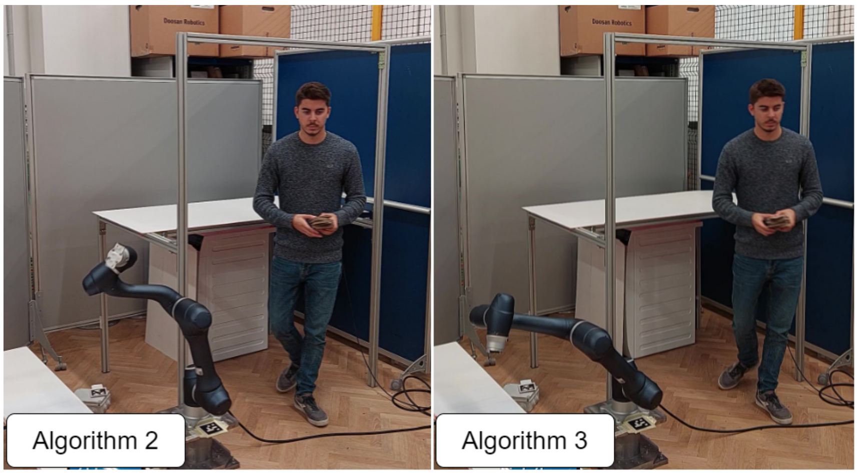
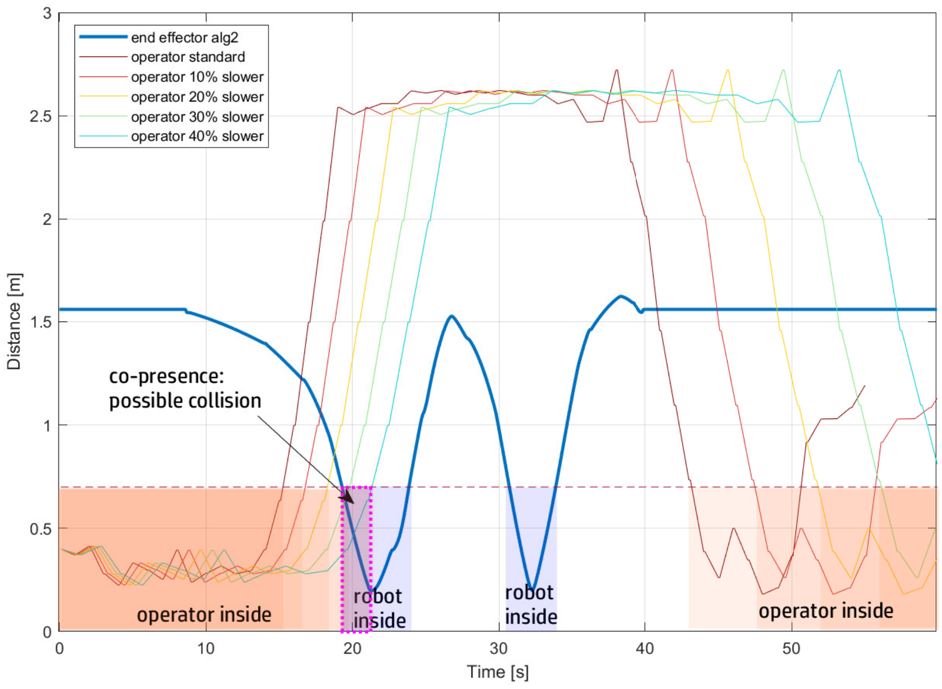


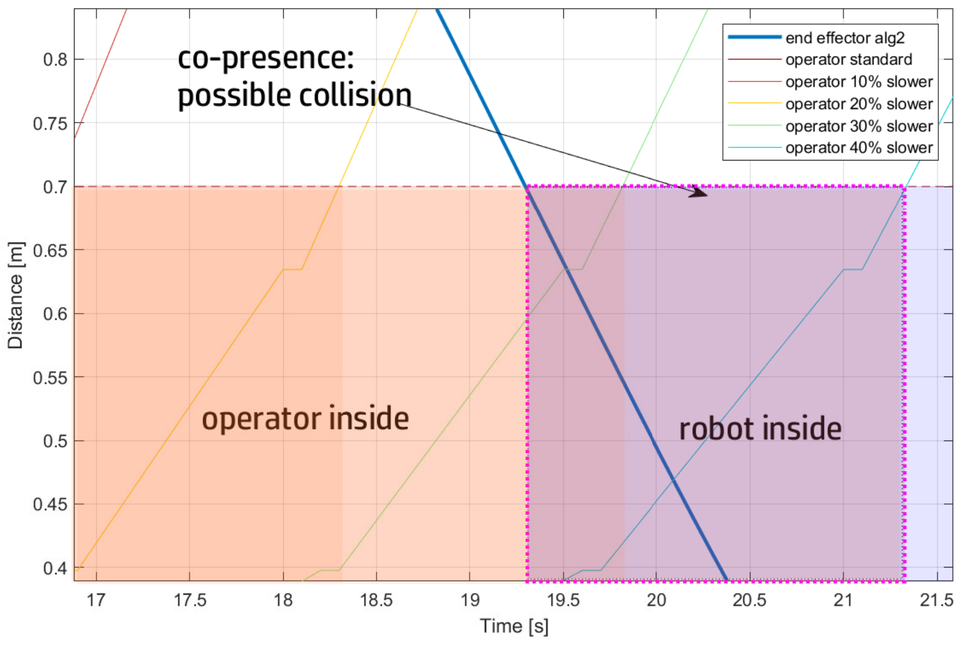
Figure 7.13: Time frame of the moment when the operator has finished the first task and is moving away from the press. It can be observed the cobot to be nearer the collaborative space in the second algorithm.

an accident can hardly happen.

In order to validate this index, it is useful to study slowed-down experiments (Figure 7.13). To do so, the distance between the operator and the press has been graphically represented together with the distance between robot and press (Figure 7.14, 7.15). Furthermore a collaborative area has been defined (Figure 7.12), such that if the cobot and the person are inside this area simultaneously there is a risk of collision.

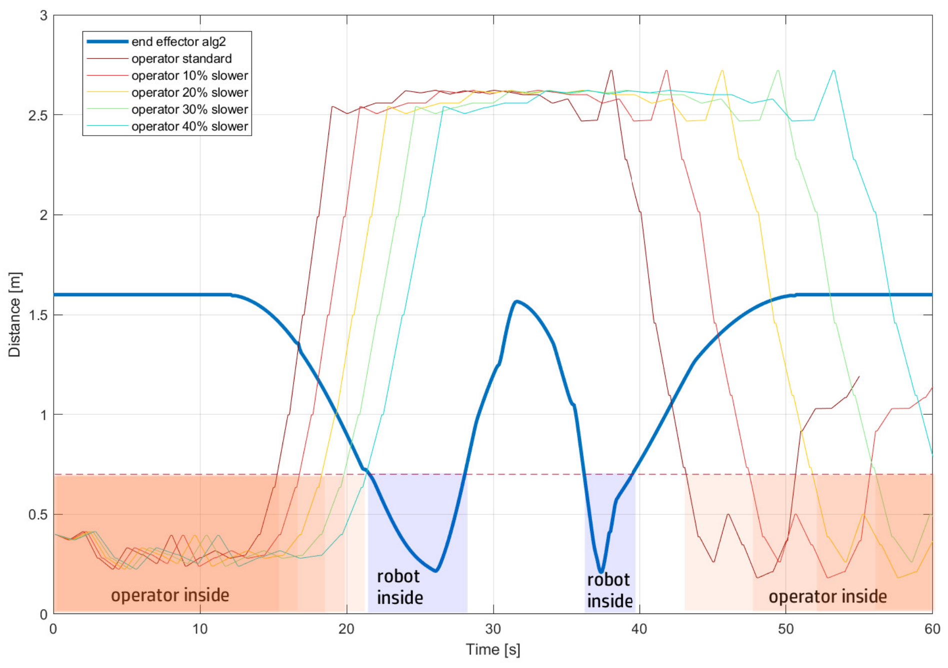


(a)

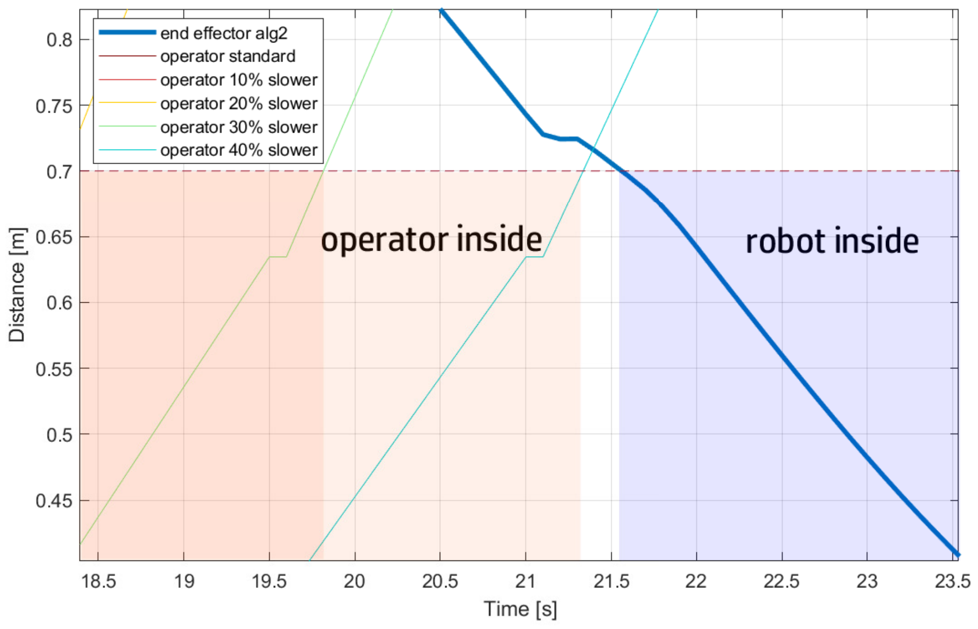


(b) zoom

Figure 7.14: This graph visualises the movements of the operator at different speeds and of the robot according to the Algorithm 2. The distance is calculated from the centre of the collaborative area. When a plot goes below the dashed line it has to be considered inside the collaborative area. If both cobot and operator are inside the area at the same time there is a risk of collision.



(a)



(b) zoom

Figure 7.15: This graph visualises the movements of the operator at different speeds and of the robot according to the Algorithm 3. The distance is calculated from the centre of the collaborative area. When a plot goes below the dashed line it has to be considered inside the collaborative area. If both cobot and operator are inside the area at the same time there is a risk of collision.

The comparison is made between only the second and third algorithms, as the first and base algorithms have already proved to be irrelevant due to excessive slow motion.

Method	**Maximum speed variance**
Algorithm2	27%
Algorithm3	42%

Table 7.3: Maximum speed variance admitted before the Human and robot presence overlap in front of the press

From the experiment it can be seen that the case of the 20% speed reduction is very close to the limit of Algorithm 2. If we study this algebraically we can see that the precise limit is 27%, while for Algorithm 3 is 42 %.
This index reveals the main feature of the third algorithm. In fact, by taking into account a window of values of the operator's positions, it is able to better accommodate possible changes with respect to standard movements. Also, during the experiments it was found that this algorithm is slightly less aggressive than the second one, giving in some ways a more comfortable feeling to the operator.

Chapter 8

Conclusions

Today the collaborative robotics trend is experiencing significant growth. This is driven by the numerous advantages that the installation of a cobot brings to a manufacturing company. These advantages include production flexibility, which is far from the one of traditional robots, as well as a higher productivity than what human operators alone can manage. These and other qualities, as well as lower costs, have motivated and are still motivating many small and medium enterprises to adopt collaborative robots. Today these cobots work by sharing work areas with humans rather than cooperating directly with them. This growing demand for new installations has gone along with the increasing focus on the safety and comfort of the operator when working with the cobot.
With regards to the safety requirements of this type of robot, an ISO-TS-15066 standard has been developed, which defines a series of approaches such as speed and separation monitoring and power and force limiting. One of the goals of this work is to develop an algorithm that can help the programmer to automatically define offline the optimal trajectory for the cobot, while at the same time increasing the safety of the operation. The aim of this algorithm is to avoid or reduce the amount of trial-and-error that a programmer has to carry out for each new task. In particular, the algorithm is able to effectively prevent the robot from colliding with existing obstacles and to ensure that it does not pass through the area that the person occupies. All of this is done by trying to emulate the behaviour that a robot would have if controlled with an online approach such as SSM, which

would require additional ad-hoc sensors and installations.
This new approach not only works for cobots already installed and in operation, but it is also designed to work in those situations where the complete plant with cobots and other machines is not yet available, i.e. joining the category of virtual commissioning tools alongside other simulators software.
The first step was to simulate the task that the operator has to perform in the work area by wearing a virtual reality device that brings him/her into a simulated industry. After recording these movements as point-clouds it was possible to start the optimization algorithm. This optimization algorithm searches for the trajectory that results in the lowest cost function represented by the task time. This is done by simulating at each step the trajectory through RoboDK, measuring each time all the distances between robot and operator and adjusting the speed accordingly. Depending on the method of calculation of these distances, three distinct methods have been developed:

- static human volumetric sweep, where the algorithm considers all the positions the operator takes during the entire operation;
- dynamic human model, where the algorithm considers only the distance between the robot and the operator at the pose that depends on the given instant;
- time window human model, where the algorithm considers a window of positions at each considered instant.

After optimizing the best trajectory between the given start and end points of the task the program can be loaded directly on the cobot. At this point the cobot is able to execute the optimized trajectory without the need of additional sensors in the final setup.

By carrying out several experiments with the trajectories identified by the three different methods, it was possible to define three quality indices used to compare the results in a more straightforward and clear manner.

In conclusion, the static human volumetric sweep algorithm is certainly the most conservative and the safest out of the three, but its overall execution time does not allow it to be considered as a solution applicable in the real world. The second algorithm, with the dynamic human model, is the best performing algorithm from the point of view of trajectory time. However, it has a low variance of the allowed operator's speed, i.e. it only synchronises well with the human if his/her speed does not deviate too much from the starting acquisition. This results in a higher risk of collision if the operator does not always work at the same speed. Combining this higher probability with a higher impact energy than the other algorithms (although only slightly), the lower safety of the solution becomes evident. Finally, the third algorithm, the time window human model, which combines the dynamic approach of the second algorithm with the more conservative approach of the first one, manages to achieve satisfactory results. In particular, the overall trajectory time is well matched to the cycle times of the process in consideration. Indeed, the improved speed variability allows for an improved collaboration with the operator.

8.1 Possible Improvements

Many things can be improved in further studies.
The process can be further automated through a more high-level management of the procedure, such as making the application autonomously perform multiple attempts and choose the best local minimum without human supervision.
Furthermore, in the case of the dynamic and time window model, information on the direction of movement can be added, as it is safer to go at high speed when the operator is moving away from the robot than when he is approaching it. Another possible improvement can be achieved by adding the accelerations as optimization parameters in order to smooth the speed changes, which are currently quite sharp. In addition to having smoother trajectories, this also benefits robot maintenance since more continuous speeds cause less wear on the joints.
Finally, the speed can be raised by increasing the safety scale factor, if the risk can be proved to be acceptable. One of the methods that can be used to reduce the risk is to take advantage of the sensors that are already present in the ma-

chines near the robot. Taking as an example the work cell studied, one could use the press safety sensors. This type of sensor, often made by a laser curtain, is needed to disable the machine when the operator is in the working area. In this way, when the operator is close to the machine and so to the cobot arm, the joints speed can be limited by lowering the k; once the operator leaves the press the cobot velocity can be raised using a higher k.

Bibliography

[1] Organization for Standardization: Switzerland. *ISO TC184/SC2, ISO/TS 15066 Robots and robotic devices - Safety requirements for industrial robots.* 2013.

[2] Luka Peternel et al. “Towards ergonomic control of human-robot co-manipulation and handover”. In: *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. 2017, pp. 55–60. DOI: 10.1109/HUMANOIDS.2017.8239537.

[3] Rainer Müller, Matthias Vette, and Ortwin Mailahn. “Process-oriented Task Assignment for Assembly Processes with Human-robot Interaction”. In: *Procedia CIRP* 44 (2016). 6th CIRP Conference on Assembly Technologies and Systems (CATS), pp. 210–215. ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2016.02.080. URL: https://www.sciencedirect.com/science/article/pii/S2212827116003620.

[4] Chui et al. “A future that works: automation, employment, and productivity.” In: (2017). [San Francisco]: McKinsey Global Institute. URL: http://www.mckinsey.com/global-themes/digital-disruption/harnessing-automation-for-a-future-that-works.

[5] M. P. Groover. “automation”. In: *Encyclopedia Britannica* (October 22, 2020).

[6] Matt Scanlan. *Manufacturing Simulation for Industry 4.0.* 2019. URL: https://www.engusa.com/en/posts/manufacturing-simulation-for-industry-4-0.

[7] *Virtual commissioning*. 2015. URL: https://team3d.it/virtual-commissioning-cose/.

[8] Ulrich Raschke and Christina Cort. "Chapter 3 - Siemens Jack". In: *DHM and Posturography*. Ed. by Sofia Scataglini and Gunther Paul. Academic Press, 2019, pp. 35–48. ISBN: 978-0-12-816713-7. DOI: https://doi.org/10.1016/B978-0-12-816713-7.00003-9. URL: https://www.sciencedirect.com/science/article/pii/B9780128167137000039.

[9] *3D manufacturing simulation software*. URL: https://www.visualcomponents.com/.

[10] *Operazioni globali: Delmia - Dassault Systèmes*. URL: https://www.3ds.com/it/prodotti-e-servizi/delmia/.

[11] *Fattore Umano Ed Ergonomia: Siemens Software*. URL: https://www.plm.automation.siemens.com/global/it/products/manufacturing-planning/human-factors-ergonomics.html.

[12] *RobotStudio: ABB Robotics*. URL: https://new.abb.com/products/robotics/robotstudio.

[13] Organization for Standardization: Switzerland. *ISO 13849-1:2015 Safety of machinery – safety related parts of control systems – Part 1: General principles for design*. 2015.

[14] K. Kaltsoukalas, S. Makris, and G. Chryssolouris. "On generating the motion of industrial robot manipulators". In: *Robotics and Computer-Integrated Manufacturing* 32 (2015), pp. 65–71. ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2014.10.002. URL: https://www.sciencedirect.com/science/article/pii/S0736584514000891.

[15] David Šišlák, Premysl Volf, and Michal Pechoucek. "Accelerated A* trajectory planning: Grid-based path planning comparison". In: *Proceedings of the 19th International Conference on Automated Planning & Scheduling (ICAPS)*. Citeseer. 2009, pp. 74–81.

[16] Dave Ferguson and Anthony Stentz. "Using interpolation to improve path planning: The Field D* algorithm". In: *Journal of Field Robotics* 23.2 (2006), pp. 79–101.

[17] L.E. Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–580. doi: 10.1109/70.508439.

[18] Xingchen Chen, Nanfeng Xiao, and Ya Chao. "Kinect Sensor-Based Trajectory Planning Method of Collision Avoidance for Industrial Manipulator with an Dexterous Hand". In: *Recent Trends in Intelligent Computing, Communication and Devices*. Springer, 2020, pp. 695–704.

[19] Steven M. LaValle. "Rapidly-Exploring Random Trees: A New Tool for Path Planning". In: *Cambridge University Press* (2006).

[20] Rodriguez et al. "An obstacle-based rapidly-exploring random tree". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* 2006, pp. 895–900. doi: 10.1109/ROBOT.2006.1641823.

[21] Olzhas Adiyatov and Huseyin Atakan Varol. "Rapidly-exploring random tree based memory efficient motion planning". In: *2013 IEEE International Conference on Mechatronics and Automation*. 2013, pp. 354–359. doi: 10.1109/ICMA.2013.6617944.

[22] John Schulman et al. "Motion planning with sequential convex optimization and convex collision checking". In: *The International Journal of Robotics Research* 33.9 (2014), pp. 1251–1270. doi: 10.1177/0278364914528132. eprint: https://doi.org/10.1177/0278364914528132. url: https://doi.org/10.1177/0278364914528132.

[23] O. Khatib. "Real-time obstacle avoidance for manipulators and mobile robots". In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. Vol. 2. 1985, pp. 500–505. doi: 10.1109/ROBOT.1985.1087247.

[24] R. Volpe and P. Khosla. “Manipulator control with superquadric artificial potential functions: theory and experiments”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 20.6 (1990), pp. 1423–1436. doi: 10.1109/21.61211.

[25] Wenrui Wang et al. “An improved artificial potential field method of trajectory planning and obstacle avoidance for redundant manipulators”. In: *International Journal of Advanced Robotic Systems* 15.5 (2018), p. 1729881418799562.

[26] Matthew P Kelly. “Transcription methods for trajectory optimization”. In: *Tutorial, Cornell University, Feb* (2015).

[27] T Chettibi et al. “Minimum cost trajectory planning for industrial robots”. In: *European Journal of Mechanics-A/Solids* 23.4 (2004), pp. 703–715.

[28] C.-H. Wang and J.-G. Horng. “Constrained minimum-time path planning for robot manipulators via virtual knots of the cubic B-spline functions”. In: *IEEE Transactions on Automatic Control* 35.5 (1990), pp. 573–577. doi: 10.1109/9.53526.

[29] Jiangyu Lan et al. “A Multi-Objective Trajectory Planning Method for Collaborative Robot”. In: *Electronics* 9.5 (2020). issn: 2079-9292. doi: 10.3390/electronics9050859. url: https://www.mdpi.com/2079-9292/9/5/859.

[30] Yu Zhao, Hsien-Chung Lin, and Masayoshi Tomizuka. “Efficient Trajectory Optimization for Robot Motion Planning”. In: (2018), pp. 260–265. doi: 10.1109/ICARCV.2018.8581059.

[31] Lu-Ping Luo et al. “Trajectory planning for energy minimization of industry robotic manipulators using the Lagrange interpolation method”. In: *International Journal of Precision Engineering and Manufacturing* 16.5 (2015), 911–917. doi: 10.1007/s12541-015-0119-9.

[32] Andrea Maria Zanchettin and Paolo Rocco. “Path-consistent safety in mixed human-robot collaborative manufacturing environments”. In: *2013*

IEEE/RSJ International Conference on Intelligent Robots and Systems. 2013, pp. 1131–1136. DOI: 10.1109/IROS.2013.6696492.

[33] Matteo Ragaglia, Andrea Maria Zanchettin, and Paolo Rocco. "Safety-aware trajectory scaling for Human-Robot Collaboration with prediction of human occupancy". In: *2015 International Conference on Advanced Robotics (ICAR)*. 2015, pp. 85–90. DOI: 10.1109/ICAR.2015.7251438.

[34] Bakir Lacevic, Andrea Maria Zanchettin, and Paolo Rocco. "Towards the Exact Solution for Speed and Separation Monitoring for Improved Human-Robot Collaboration". In: *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. 2020, pp. 1190–1195. DOI: 10.1109/RO-MAN47096.2020.9223342.

[35] Niccolò Lucci et al. "Combining Speed and Separation Monitoring With Power and Force Limiting for Safe Collaborative Robotics Applications". In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 6121–6128. DOI: 10.1109/LRA.2020.3010211.

[36] *Simulate robot applications with RoboDK*. URL: https://robodk.com/.

[37] *About robodk*. URL: https://robodk.com/about.

[38] *Broyden–Fletcher–Goldfarb–Shanno algorithm*. 2021. URL: https://en.wikipedia.org/wiki/Broyden-Fletcher-Goldfar-Shanno_algorithm.

[39] Larry Armijo. "Minimization of functions having Lipschitz continuous first partial derivatives." In: *Pacific Journal of Mathematics* 16.1 (1966), pp. 1 –3. DOI: pjm/1102995080. URL: https://doi.org/.

[40] Richard Scheunemann Denavit Jacques; Hartenberg. ""A kinematic notation for lower-pair mechanisms based on matrices"". In: *Trans ASME J. Appl. Mech. 23* (1955), pp. 215–221.

[41] Dan Sunday's Geometry Algorithms. *Minimum distance between two line segments) in 3D*. URL: https://www.it-swarm.it/it/algorithm/calcolo-della-distanza-piu-breve-tra-due-linee-segmenti-di-linea-3d/957789635/.

[42] AlainChabrier. *Warm start optimization*. 2021. URL: https://medium.com/@AlainChabrier/warm-start-optimization-ac73eef189e9.

[43] Lorenzo Damiani et al. “Augmented and virtual reality applications in industrial systems: A qualitative review towards the industry 4.0 era”. In: *IFAC-PapersOnLine* 51.11 (2018). 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018, pp. 624–630. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.08.388. URL: https://www.sciencedirect.com/science/article/pii/S2405896318315131.

[44] *Unity (Motore Grafico)*. 2021. URL: https://unity.com/.

[45] *Controllers and Hand Tracking - Oculus Quest2*. 2021. URL: https://support.oculus.com/290147772643252.

[46] *Kinect*. 2021. URL: https://developer.microsoft.com/it-it/windows/kinect/.