UNIVERSITÀ DEGLI STUDI DI PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

MASTER DEGREE IN
CONTROL SYSTEMS ENGINEERING

An Algorithm for Joints' Boundaries in a Six Degrees Of Freedom Robot

Supervisor:
PROF. EMANUELE MENEGATTI

Co-Supervisors:
ALBERTO GOTTARDI
NICOLA CASTAMAN

Student:
NICOLA MIOTTO
2019292

Accademic Year 2022/2023
Graduation Date: 6 March 2023

Abstract

In the context of Human-Robot Cooperation, the interaction between humans and robots is a crucial point. In addition, the safety of the operator during the collaborative transport of objects is fundamental. Therefore, in this thesis, an algorithm to enhance human-robot cooperative transport has been developed. The aim is to let the robot better adapt to the human operator's motion during the transport in the workcell. This will be achieved by taking advantage of some properly instantiated virtual obstacles and a fast approach to check the robot manipulator collisions with such obstacles, in order to obtain boundaries around the nominal path inside which the robot will accept deviations introduced by the human operator motion. The nominal path will be made available by a global planner such as RRT-Connect while respecting the boundaries will be managed by an MPC. The algorithm has been tested in three use cases for DrapeBot, a European Project for the collaborative draping of carbon fiber patches. Tests were performed instantiating 100 to 100 000 virtual obstacles. With 12148 obstacles instantiated the algorithm creates the boundaries joint limits in 1.527 s with 99.9774% confidence.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this work, we will describe a new algorithm to obtain collision-free regions for a robot, expressed as subspaces of the jointspace. The algorithm has been developed for a 6-DOF robot arm system and then generalized for an n-axis robot. This work has been developed and tested for the European Project DrapeBot, a project developing collaborative draping of carbon fiber parts[1], [2].
More in detail, *draping* is the process of placing soft and flexible patches of textile material on a 3D shape. Despite the development of automated tape laying and fiber placement, draping remains very important due to its flexibility and its compatibility with a wide variety of fabric materials. As a result, 30% of aerospace composite parts are still produced through draping, while almost all marine and wind energy structures are made this way. At present, most draping is done manually, with only 5% of aerospace composite parts using some automated draping process[3]. In Figure 1.1 we can see a robot and a human holding together a carbon fiber patch.

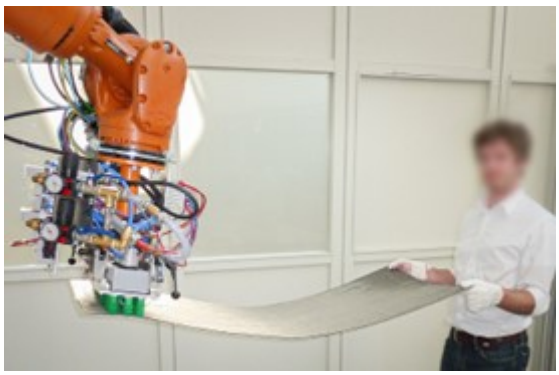


Figure 1.1: Picture of a robot and a human performing a collaborative task related to draping

Robotic draping has made significant progress with respect to the handling large patches of material and accuracy of the draping, where it has reached TLR6-7, but there are a number of challenges that still remain to be addressed, such as:

- Many parts include surface elements of high curvature, whose automatic draping is beyond the capabilities of a robot draping system and will likely remain out of reach (technically and economically) for the foreseeable future. This creates a need for

human-robot collaboration

- The robotic systems involved in draping are usually large-scale robots and interaction methods are needed in which such robots can safely and efficiently cooperate with humans

The DrapeBot project aims at increasing the range of parts for which (semi-)automated robotic draping can be applied. This will be achieved by developing a collaborative approach, where the robot deals with larger areas of lower curvature and the human is draping the geometrically complex features.
The focus of the work presented in this document is to enhance and simplify human-robot collaboration during the motion of the robot. As an example, the typical operational cycle is, starting with the robot in its home pose, going to pick up the patch, transferring the patch to the mould and, after having performed the draping process, going back to the home pose. Ideally, the robot performs such motion following a nominal path computed by a global planner. But, since there's a human in the loop, the real path will deviate from the nominal one, as the human operator will introduce unpredictable forces which will inevitably alternate it. Therefore we need to let the robot follow the human movements as much as possible without colliding with the objects in the scene and while maintaining a path resembling the nominal one. Aiming this, the algorithm presented computes reasonable boundaries allowing this kind of deviation, while respecting these boundaries will be guaranteed by the low-level controller.

We stated above that draping has applications in many different industries, and we will now define some applications related to our specific system. One use-case for DrapeBot is to be part in the production of "Stradale" (designed and produced by Dallara as a car manufacturer since 2019), where it will be used to manufacture complex components such as the front hood (size: 2x1.5m) or the side panel (size: 3 x 0.7m). Another use-case will be in shipbuilding, where it will be used for manufacturing the outer skin of the hull. Baltico has developed the strand laying technique to automatically place carbon fibers on ribs to produce the inner structure of the hull with an FRP (fiber-reinforced plastic) skin on top. This skin is currently draped manually and some degree of automation will also be beneficial for this production step. Typically, the hull is about 10 m long and 1.5 m high. Most of the surface is relatively flat and suitable for automatic draping, but there are areas with high curvature.
A third use-case is in the aerospace industry, where major parts of the structural framework of airplane fuselage are skin-stiffening elements such as stringers and ribs, and the draping strategy is defined in the part design and dictates areas where the material has to remain steady (seed points) and areas where the material has to be deformed (draped) into proper fiber angles. The partner related to the such application is DLR.
The algorithm developed in the current work will be tested in models of the workspaces of the use-cases described. The results will be presented in Chapter 4, where the use cases will be referred to as Dallara, Baltico and DLR, respectively. Its purpose will be to enhance human-

robot interaction by computing boundaries around the nominal path in the jointspace, in order to let the human operator deviate from the nominal path inside such boundaries without entering into a collision with the objects in the workspace.

1.1 Objectives and Structure

In order to stay aligned with one of the main objectives of DrapeBot, which is to enhance human-robot collaboration in the draping process, in the current work we focus on increasing the adaptability of the robot to the human operator. More specifically there will be developed an algorithm at the interface between the global planning module and the low-level controller. As the global planner will produce the nominal trajectory for the robot arm, we cannot expect the such path to be coherent with human movements, as they're inherently unpredictable. The aim of the algorithm is to compute, starting from the nominal trajectory, a set of boundaries expressed as joint values which will represent the interval inside which the joints value can be moved during the movement, in order to accommodate deviations from the nominal trajectory which are to be realistically expected as we have a human in the loop. Such boundaries should also reasonably ensure to change the trajectory without giving raise to collision within the workcell environment, as well as allowing paths resembling the nominal one. The low-level controller will have the duty to respect such boundaries. The aim can be stated as to enhance human-robot collaboration during movements by defining a subspace of the jointspace allowing deviations from the nominal trajectory.

This thesis will present some related works in Chapter 2, mainly related to Artificial Potential Fields (APF) and Model Predictive Controllers (MPC) and some techniques to analyze the workspace and the jointspace. In Chapter 3 our solution will be described mathematically; it will be based on checking the collisions of particular robot configurations with some virtual obstacles. In Chapter 4 the proposed solution will be tested and evaluated, with some specified metrics, over the three use-cases previously described, Dallara, Baltico and DLR. The results will also be discussed. Finally, in Chapter 5, the entire work will be summed up and some considerations regarding future works will be observed.

Chapter 2

Related Works

The problem we want to solve is to find a way to enhance the robot's adaptability to the human operator during collaborative tasks while maintaining a resemblance to the nominal path computed for such a task. The critical point is that the robot must adapt to the human, not the opposite. To the best of our knowledge, the problem at hand doesn't have a canonical solution in the current literature. There are though many related problems that are fully available and which can be useful as inspiration and comparison, which will gradually lead us to the decision of developing an algorithm to bound the nominal path in the jointspace. Model Predictive Control and Artificial Potential Fields are good starting points, as they allow us to update and modify the trajectory in real-time.

2.1 Model Predictive Control

In robotics technology, recent research trends focus on performing particularly critical tasks optimally, while fulfilling some plant constraints to avoid failures, wear of the electro-mechanical parts, or to guarantee safe and close human-robot interactions [4]. In the last decades, among the control algorithms published in the literature, Model Predictive Control (MPC) represents an appropriate and effective solution to solve this kind of problem, providing an optimal control strategy in case of even complex constrained dynamical systems [5], [6].
Essentially a MPC is constituted by:

- a prediction model
- an objective or cost function
- constraints on the system states and inputs

The aim is to obtain a control law minimizing the objective function [7]. In simple cases there could be no constraints, but they're typically present in practice as well as in the cases we're going to describe. Formally, considering the example of a system with m inputs and p outputs

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ x(k) &= x_k \end{aligned} \tag{2.1}$$

for the time horizon $[k, k+N]$, we want to minimize a cost function such as

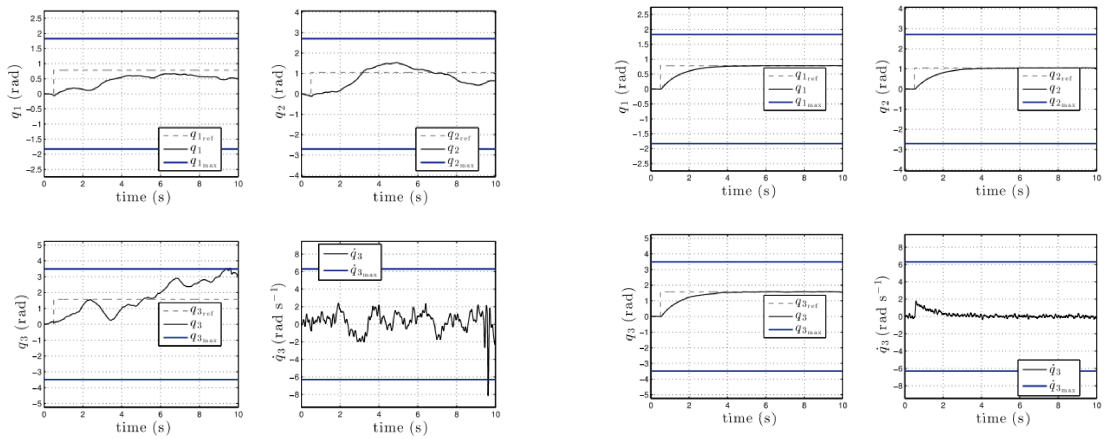
$$\min_u \; V(x(k), u_k, k) = \sum_{i=0}^{N-1} [x^T(k+i)Qx(k+i) + u^T(k+i)Ru(k+i)] + x^T(k+N)P_f x(k+N)$$

$$s.t. \quad x(k) \in \mathbb{X}_k, u(k) \in \mathbb{U}_k \tag{2.2}$$

where $Q \geq 0, R > 0, P_f > 0$ are the matrices setting the cost for the state vector $x(k)$, the input u and the final state $x(k+N)$ of the time horizon respectively.
The result is an optimal control sequence $u_k = (u(k), u(k+1), ..., u(k+N-1))$. The sets $\mathbb{X}_k$ and $\mathbb{U}_k$ abstractly represent the possible constraints acting at each time step k. To better understand how MPC can be applied to robotics, let's first have a clearer depiction of the kind of systems we are interested in.

A robot manipulator consists of a series of rigid bodies (*links*) connected by means of kinematic pairs or joints. Joints can be essentially of two types: *revolute* and *prismatic*. The joints' reference systems are called *frames*. The whole structure forms a kinematic chain. One end of the chain is constrained to a base. An *end-effector* (gripper, tool) is connected to the other end allowing manipulation of objects in space [8].
From a topological viewpoint, the kinematic chain is termed *open* when there is only one sequence of links connecting the two ends of the chain; it is termed *closed* when the start and the end of the chain coincide. The robot belonging to our system is open chain and all its joints are revolute.
The mechanical structure of a manipulator is characterized by a number of degrees of freedom (DOFs) which uniquely determine its posture. Each DOF is typically associated with a joint articulation and constitutes a joint variable.

As a first application of MPC to manipulators we are going to consider the robust multiloop hierarchical control scheme designed by Incremona[9] to solve motion control problems. The control scheme consists of three loops: an inner loop based on the so-called inverse dynamics approach, aimed at transforming the nonlinear multi-input-multi-output (MIMO) robotic system into a set of perturbed linearized decoupled single-input single-output (SISO) systems (the number of systems is equal to the number of the joints of the robot manipulator); a second loop including a controller designed according to the so-called Integral Sliding Mode (ISM) control approach, which has the role of rejecting at a higher rate all the matched uncertainties in each frame; finally, an external loop involving a controller of MPC type with the role of guaranteeing the optimal evolution of the controlled system in the respect of state



(a) (b)

Figure 2.1: Evolution of joint variables using a nominal MPC (a) and an ISM MPC (b). Notice the joints' boundaries (blue lines)[9]

and input constraints.
Such a scheme was successfully adopted in a robot manipulator to control the first three joints, showing a very precise tracking of the joints' nominal values with constraint satisfaction. It was tested in a collision-free workspace though, and clearly no component of its design would significantly facilitate human-robot interactions. At best, the cleaner reference following, as reported in figure 2.1, could provide a more quiet execution from the human operator point of view, but this would be a pretty slim improvement and wouldn't be useful in adapting the robot motion to the presence of the human user. It would turn out to be useful for the solution we will later propose to notice that the MPC is able to handle constraints in terms of joints' boundaries. This is true for MPC in general, not necessarily employing the ISM control approach.

The issue of obtaining a less stressful execution for the human operator is better addressed by Faroni[10] which implemented trajectory scaling in the context of MPC. This MPC tries to maximize the distance between the robot and the human operator and to slow down the movement when the distance is below a given threshold, introducing a velocity scaling factor to the nominal trajectory which changes based on the distance-dependent safety function in order to achieve this. This approach appeared to increase the distance from the operator in a pick-and-place task performed with a 6-DOF manipulator over a linear axis. Although improving the interaction with the human operator, this happens to do so in a task without a great degree of collaboration between the two. This would actually still be a useful implementation in our scenario, but it wouldn't address the need to increase the robot's adaptability to humans deviating from the nominal trajectory. In some specific tasks, where the human and the robot have to work in proximity, this approach could even be detrimental, as it would pull the robot away from the operator.
The previous work[10] has been further extended in order to implement in a unified way the current safety measures existing in the literature[11]. The end result is a human-aware

motion planner, embedding psychologically-grounded distance-velocity mapping and a real-time safe motion unit. Despite these improvements, the system described by Eckhoff[11] still suffers from the already mentioned limitations regarding our objective. Indeed it achieves state-of-the-art safety but doesn't significantly address adaptability. Another related issue is indeed that collision avoidance in these last two works is left to the global planner, which by definition outputs a collision-free path. While this is true, this implicitly means that the authors are taking into account only small modifications of the nominal trajectory by the operator's actions. This assumption is quite narrow, as the operator could be the origin of more pronounced deviations, and we want to take into account this more general scenario. Therefore we now start to evaluate collision-avoiding techniques.

2.2 Artificial Potential Fields

An effective paradigm for online planning relies on the use of artificial potential fields (APF). Essentially, the point that represents the robot in configuration space moves under the influence of a potential field U obtained as the superposition of an attractive potential to the goal and a repulsive potential from the obstacle region. Planning takes place in an incremental fashion: at each robot configuration $\mathbf{q}$, the artificial force generated by the potential is defined as the negative gradient $\nabla U(\mathbf{q})$ of the potential, which indicates the most promising direction of local motion [8]. One ever present complication with APF is the presence of states where the total potential is null, called local minima. In such states, no force is exercised on the robot, and therefore it's necessary to find a way to escape from such conditions. In the following paragraphs, we will give a description of the state of the art on APF in order to be able to analyze their limitations and compare these approaches with the one proposed in this thesis.

One first simple application of APF is as a whole-arm path planning algorithm for a 6-DOF industrial robot[12]. Here the obstacles are modeled as spheres or cylinders, and a repulsive force is exercised from their centers to the frames' locations on the arm and on the end-effector, while an attractive force is exercised by the goal position. Thus, this APF uses just some points on the whole robot, not taking advantage of its entire structure. It is also applied in a stand-alone system, where the aim is just to control the arm, but the arm motions don't have to take into account the presence of a human being. It shares with our scenario the possibility of assuming a static environment during execution, which coupled with obstacles modeled as stated, allows for simplified obstacles detection using their projection in the Z-O-X plane in order to compute their distance relative to the arm. A fast way to manage objects' distances in the environment will also be relevant in our solution.
The issue of not making use of the whole robot body is addressed by Park[13], which employed a rigid body model in order to apply the forces on the whole arm. The rigid body model employs a dense amount of material points in order to approximate the structure of each robot's link. These two works[12], [13] also differ in how they detect obstacles, as in the

first one the check was performed on the frames to see whether they appeared to fall inside the spheres or cylinders, while in the second the check is performed over each material point through a discrimination matrix. The obstacles are modeled as spheres with two layers, in order to distinguish when the arm is effectively colliding with when they're in hazardous proximity. Also, instead of just using the end-effector goal pose as an attraction point, the position of each material point at the joints' goal values is used as an attraction basin. Such joints' values are obtained approximating the Jacobian with numerical methods to solve the inverse kinematics problem, as this robot has 7-DOF.
The problem of local minima was not addressed in the first work. Instead, in the second work, the authors solved this problem through the *dislocation* of the rigid body into many material points, as the force exercised on a single point can become null, but not the summation of all the forces.

Both of these works don't address any aspect of human-robot collaboration. One way to make use of APF to do so is, again, for increasing human operator safety[14]. This work differs from the previous one in the detection of the obstacle. Obstacles are mapped in the C-space, which was not the case before. For example, the mapping of a point obstacle for two-links systems of 3-DOF is often a curve with two or three sections. This paper reports results only for a two-links 3-DOF system but claims that the method can be extended into higher DOF. The main utility would be, as the human enters a body part in a hazardous region, this algorithm executes and impose a repulsive force on the robot, thus making it get away from and avoid harming the user. It is a different approach to obtain a similar result to what was described in the trajectory scaling MPC[10] and in the human aware planner[11]. Instead of trying to maximize the distance by an objective function, it uses the APF to pull the robot away. Still, this result doesn't come out to be useful for our aim, as improving operator safety doesn't turn out to help the robot to adapt to the user.

Other contexts of APF usage don't give rise to direct solutions too. We can observe how in Unmanned Aerial Vehicle (UAV) APF can be applied for collision avoidance[15] or for autonomous exploration[16]. For collision avoidance, the authors report a heuristically activated APF, where the repulsive potential dominates attractive potential for coplanar obstacles which are between the UAV and its goal, while attractive potential dominates repulsive potential where the obstacle is detected but doesn't lie in the path to the goal. The forces' direction is always such that the UAV moves in straight lines, and this perpendicular approach manages the local minimum problem. This is an advantage of the aerial environment which clearly doesn't hold in an industrial environment for a robotic arm.
Regarding UAV in exploration, the technique presented makes use of APF to update an entropy map when an obstacle is detected, in order to update an entropy function and then construct a 3D octomap of the environment. The UAV is moved towards regions with lower entropy, thus containing less information as they belong to less explored areas. The local minima problem is solved with a local repulsive potential applied when the vehicle stands on

a cell of the OctoMap for over a given amount of time. In DrapeBot we do already have a model of the environment, as should be expected in an industrial setting and so exploratory techniques are not necessary. This example though suggests us a way to build up an environment from collision data, and this fundamental approach will be exploited in the final solution in order to add virtual obstacles which will be used to influence the mobility of the joints.
MPC and APF have been used together in the motion control of autonomous vehicles[17]. Here the road borders' repulsive potential and the obstacles' one are modeled as in Figure 2.2. The MPC allows controlling the vehicle to be held close to the nominal path, as otherwise, the road APF would attempt to push the vehicle toward the center of the road. This is again a different context than our case, but the APF modeling will be influential to the addition of virtual obstacles in the final solution just cited above.

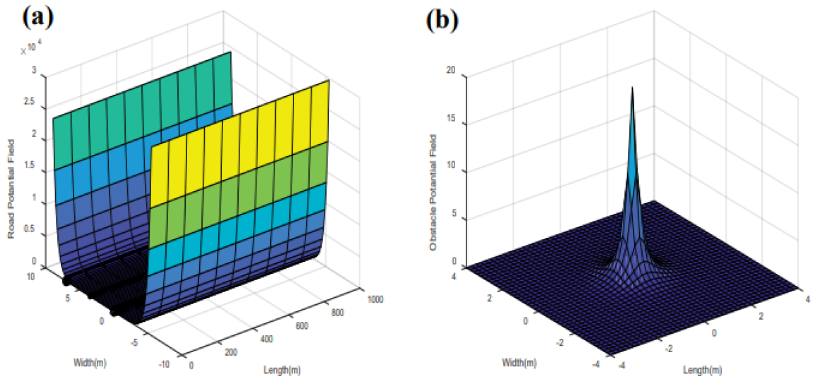


Figure 2.2: Schematic model of the APF used in the autonomous vehicle, for the road's borders (a) and the obstacles (b) [17]

2.3 Collision-Free Jointspace Subset

Considering the limitations relative to the aim of increasing the adaptability to the human operator of the APF solutions proposed in the previous section, we now shift to the research of a way to define a suitable collision-free subset of the jointspace. If we manage to find one in an acceptable amount of time we could let the robot arm accept deviation when the user's presence causes the robot to move inside this subset while leaving to the low-level controller the job of not allowing to exit such subspace. Although realized by a different entity than where the proposed solution was developed, the low-level controller will indeed be based on an MPC and, referring to equation 2.2, we could pass this subset as a constraint in $\mathbb{X}_k$.

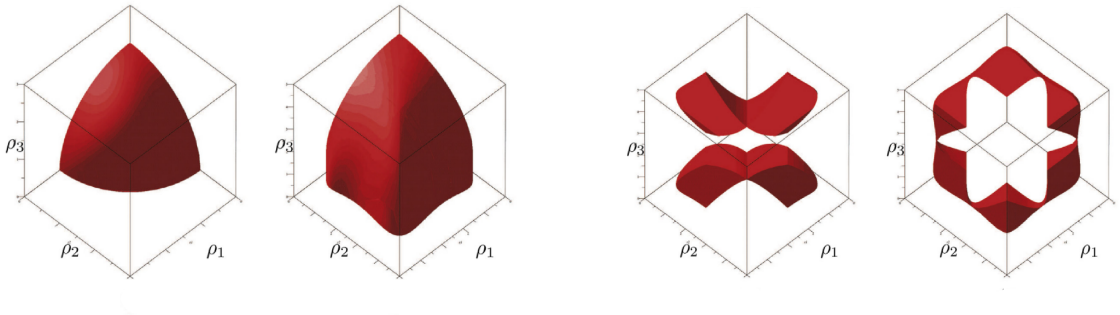
It is usually possible to develop closed-form laws for the joints by studying the geometrical and topological properties of a given mechanism, and such laws would be available also for our robot, which has a well-known structure. For example, in the case of a 3-DOF translational parallel manipulator[18], the workspace has been defined by using a topological design method of parallel mechanism based on position and orientation characteristic

Manipulators	**Plotting Time** [s]
Orthoglide	7.650
Hybridglide	23.536
Triaglide	13.682
UraneSX	25.100

Table 2.1: Plotting times for the delta-like robots using the CAD method[20].

equations. Once the workspace has been defined, it is possible to access the jointspace by inverse kinematics, such as it was done with the APF with material points[13], although this would add a computational burden if performed online. The most critical issue though is that, once a significant amount of obstacles are present in the environment, obtaining a closed form becomes impossible or highly impractical.
To overcome this issue Xiangdong[19] has proposed a method for mapping the boundaries of collision-free reachable workspaces. The obstacle and the end-effector are modeled as intersections of convex implicit surfaces, specifically convex superquadrics in the cited work. By employing an interior-point algorithm based on the Newton-Rampson method, the collision-free conditions are obtained and integrated with the continuation method, creating an algorithm that enables the boundary mapping of collision-free reachable workspaces. Clearly, as these are boundaries on the workspace, they are computed from the end-effector perspective. Therefore, the inverse kinematics issue remains, and for a 6-DOF system such as the one we are considering there could be in general up to 16 admittable solutions[8].
Another interesting approach would be to use Cylindrical Algebraic Decomposition (CAD)[20]. Although this has again been tested in an obstacle-free environment, it would allow us to obtain the jointspace directly differently from the previous two works cited, bypassing the inverse kinematics approach. Some examples of the jointspace plots for a family of delta-like robots computed using CAD are presented in figure 2.3. In table 2.1 the computational times for various robots (Orthoglide, Hybridglide, Triaglide, and UraneSX) are reported. CAD has two main issues though. The first one is its computational complexity being double exponential. This implies the computational time can explode pretty soon increasing the input by a small amount. Also, the absolute values as reported in table 2.1 aren't very exciting, as we would like to get them in a couple of seconds at most in the real use-case. The second issue would be that a jointspace obtained in an environment with obstacles would not likely be expressable with a closed form, and thus it should be stored and, at each step, the current joints values should be checked to be inside the storing structure or close enough, adding again online computational burden and likely causing snappy movements. This last criticality would also be present if we decided to take advantage of the previous work using superquadrics and then "translate" the bounded workspace in a finely sampled subspace of the jointspace.

To overcome the issues and limitations presented, in the next chapter, we will develop an algorithm to obtain boundaries in the jointspace defining interval in which the nominal value



(a) (b)

Figure 2.3: Joint space plot for Orthoglide and Hybridglide (a), Triaglide and UraneSX (b) robot[20].

of the global planner will lie inside. Indeed, the main advantage we have over the methods described in this section is that we don't need to obtain a completely generalized depiction of the spaces, but we can use the nominal path as a guide and build a subspace around it.

Chapter 3

Proposed Solution

As we have seen in the previous chapter 2, we cannot provide a suitable way of allowing collision-less deviations from the nominal path using only MPC or APF. As well, the techniques [19], [20] for deriving valid subsets of the jointspace have their own limitations, mainly in terms of computational complexity. In this chapter, we are going to explain and discuss the solution that has been developed in order to find collision-free intervals in the robot's jointspace around the nominal values generated by the global planner. The solution will present a linear computational complexity and will be entirely performed *offline* in the jointspace, thus without occupying computational time for low-level controllers during execution.
The overall concept is to generate a volume that synthesizes the whole workspace but is left free in the space surrounding the nominal path and near the goal, then the boundaries defining the final intervals are found by observing for which values of the joints the robot collides with such volume. The algorithm consists of the following main steps:

- obtain the nominal path from the global planner;
- separate the obstacles into different categories, namely *goal-obstacle*, *non-goal obstacles* and *trail*;
- generate the volume;
- observe how the robot interacts with the volume.

The first step isn't strictly part of the developed solution, but its output is a necessary starting point. The development and testing of the algorithm were done using RRT-connect as a global planner[21], but any kind of planner outputting the nominal path expressed by joint values would be fine. In this work, the global planner takes as inputs the end-effector start and goal positions in 3D Euclidean space.
Concerning the second step, the *trail* is a set that encapsulates the information about the nominal path. As the set of obstacles, it will be composed of collision boxes, as we'll better describe in the next section.
In the third step, we use the sets defined in the step before to grow a volume from a height-map overlapping the floor of the workspace. More explicitly, we will derive a height for

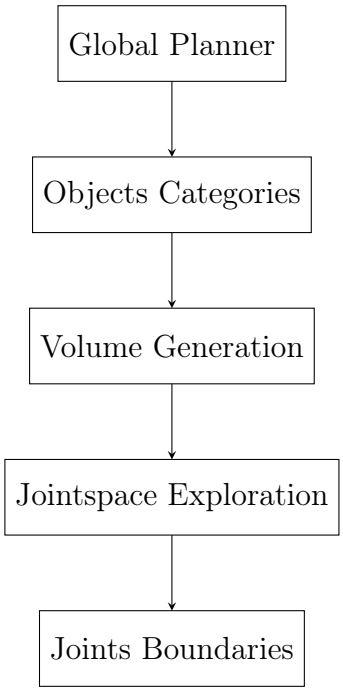


Figure 3.1: Flowchart summarizing the proposed algorithm

each cell of the height-map and instantiate a prism (again a collision box) of such computed height on that specific cell. The final volume will be the set of these prisms.
Finally, in the last step, we observe how the robot interacts with the previous volume in order to define the boundaries. This will consist in observing if the robot collides with the volume at some specific points in the jointspace. In the flowchart in figure 3.1 we can visualize the steps comprising the algorithm.

3.1 Objects Categorization

This preliminary step is needed as different objects should influence the subsequent volume, and thus the joints, differently. The objects will be divided into the three categories previously cited: *goal-obstacle*, *non-goal obstacles*, and *trail*. The *goal-obstacle* is just the obstacle assumed to be below the goal position of the end-effector, the *trail* is the set of robot links as they are oriented through the nominal path and *non-goal obstacles* are the remaining obstacles. All the objects will be simplified through the use of collision boxes. We will now give a mathematical description to let the reader better understand how such objects are managed during the execution.

A collision box can be defined by a Roto-Translational Matrix $\mathbf{R} \in \mathbb{R}^{4\times4}$ containing the information about the position of the box's center and a vector $\mathbf{s} \in \mathbb{R}^3$ containing the lengths of the box's sides. Collision boxes give us the possibility of checking very efficiently whether a point is inside them and whether two collision boxes are colliding. These will be useful in various steps of the proposed algorithm.
In order to check whether a given point in 3D Euclidean space is inside a collision box we just

need to translate the point being checked into a system of reference centered in the box's position, and then check if the point's coordinates in this system are, in absolute value, smaller than the relative box's sides' half lengths.
To check if two boxes are colliding we take advantage of an algorithm developed two decades ago by Gottschalk[22], which performs some tests on two boxes returning 0 if they touch and a positive number if they don't.
From 3D models of the environments, we do have a collision box for each object, and thus we have a set of obstacles' collision boxes. It is undesirable for the volume to grow too close to the object corresponding to the goal pose; as we get closer to the goal we do expect the joints' intervals to become narrower, as the human operator should have less maneuver to deviate from the nominal path, but too much volume could lead to the collapse of the interval to the nominal values, for reasons which will be clearer in section 3.3. This would amount to leaving almost no freedom to the human operator.
Identifying the obstacle corresponding to the goal is therefore critical, as this object will be treated differently from the others for the reason just described. We will refer to this obstacle as *goal-obstacle*. Usually, in the use cases we're interested in, the goal position of the end-effector is above a table, as the robot should move from over one table to another in order to perform draping or to pickplace tasks. Assuming this is true in general, we can easily identify the collision box of the *goal-obstacle* with the simple procedure of Algorithm 1.

Algorithm 1 Identify Goal Obstacle

```
point ← endeffector 3d pose
while point.Z > 0 do
    for all collision box ∈ obstacles do
        if point is inside collision box then return collision box
    point.Z ← point.Z − Δ_Z
```

To find the *goal-obstacle* the algorithm simply starts at the goal position of the end-effector and then descends along the Z-axis until the point lies inside one of the obstacles' collision boxes. The check is performed using the algorithm for checking whether a point is inside a collision box, outlined in the previous section. Once the *goal-obstacle* has been found, the remaining obstacles are set to their own category.

It is necessary to take into account the nominal path as determined by the global planner. In order to do so another set of collision boxes is defined, called the *trail*. Given k sets of joints' values of the nominal path and m links, the *trail* is a set of $m \times k$ collision boxes, where each collision box corresponds to a given robot link at a given point (in the jointspace) of the nominal path.

3.2 Volume Generation

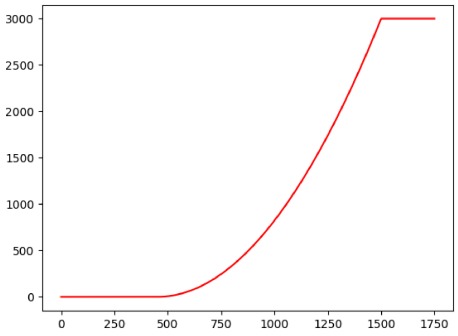
The aim of this volume is to synthesize the obstacles and the spatial areas unrelated to the nominal trajectory in one single object to be used to gauge the moving limits of the robot with the procedure described in the next section. To perform this step we need a height function and a height-map. The resulting volume will be again a set of collision boxes.

3.2.1 Height-Map

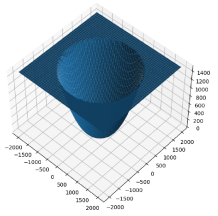
The volume is grown over a height-map which roughly approximates the workspace floor. A height-map was chosen over a 3D structure, such as an octree or octomap[16], because it allows us to save a significant amount of computational cycles, as we lose one spatial dimension. We also don't expect the human operator to deviate the trajectory significantly upward most of the time, and so adding more information in such a direction wouldn't be particularly useful.
A height-map was also preferred over a quadtree[23] since a possibility with an exploratory approach would be that of resulting in a map with large void sections corresponding to the free workspace. So, once the height of the cells gets computed, there could be significant discontinuities between two adjacent volumes, and this could be the origin of chaotic behaviors in the search for the final joints' boundaries. As seen in section 2.2, a heterogenous structure such as an octomap (or a quadtree) is better suited for exploratory purposes, but in DrapeBot the environment is defined by the workcell constructor and therefore can be considered known, as well as static during a given trajectory movement. Indeed height-maps are canonically used for reconstruction and mapping purposes, but here we are going to use them for generative purposes.
The height-map can be defined by an origin $O_h \in \mathbb{R}^3$, a pair (x_{dim}, y_{dim}) of dimension of its sides and a maximum number of cells MAX_{cells}. The first two parameters should be chosen such that the resulting height-map covers the floor of the workspace. The resulting side of a single cell would be $side = \sqrt{\frac{x_{dim} y_{dim}}{MAX_{cells}}}$, the number of rows $x_L = \frac{x_{dim}}{side}$ and the number of columns $y_L = \frac{y_{dim}}{side}$. These last two values must be rounded to an integer value if needed. In order to instantiate the volume we need a way to decide how much to grow up the volume in each cell of the height-map.

3.2.2 Height Function

The aim of the height function is to set, cell by cell, the heights of the final volume. There is ample freedom in the definition and tuning of the height function. For the current work, the choice was to define a function that would make the volume grow higher closer to the *non-goal obstacles* and lower when near the *trail* or the goal obstacle. By doing so we allow the joints' boundaries to exclude the regions of space around the obstacles but to not become too narrowed around the nominal values or around the goal region.
The mathematical definition of the height function follows:

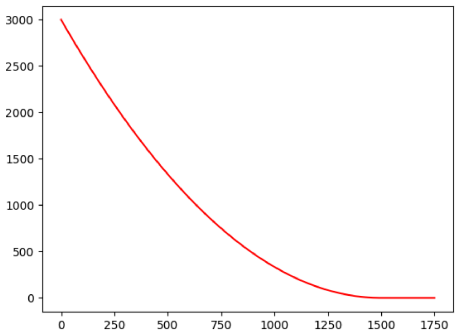


(a)

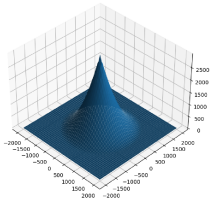


(b)

Figure 3.2: Representation of the height function components of *trail* and goal obstacle, plotted over distance (a) and depicted in 3D space(b)



(a)



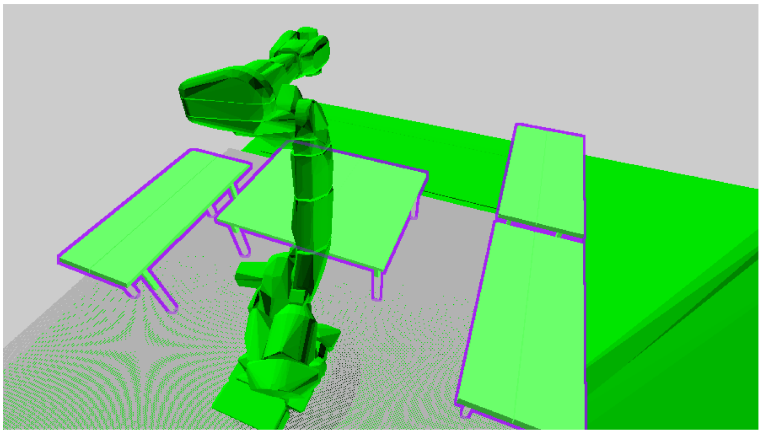
(b)

Figure 3.3: Representation of the height function component of the *non-goal obstacles*, plotted over distance (a) and depicted in 3D space(b)

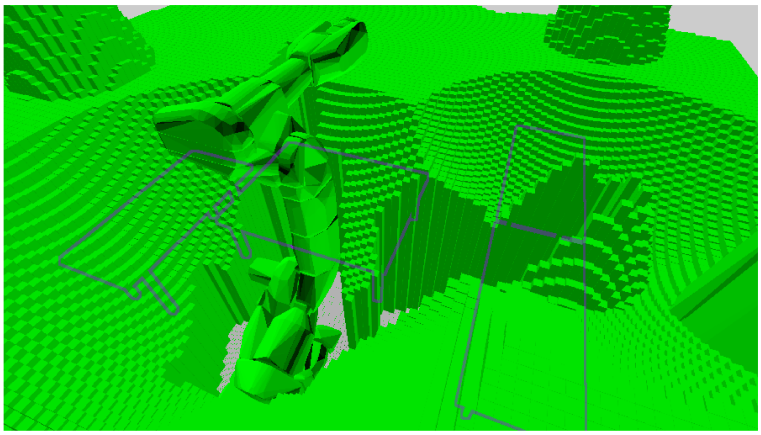
$$h(c) = A_t(c)\left(h_{max}\left(\frac{d_t - d_{t,LB}}{d_{t,UB} - d_{t,LB}}\right)^{\alpha_t} + h_{max}\left(\frac{d_g - d_{g,LB}}{d_{g,UB} - d_{g,LB}}\right)^{\alpha_g} + h_{max}\left(\frac{d_{ng,UB} - d_{ng}}{d_{ng,UB}}\right)^{\alpha_{ng}}\right) \tag{3.1}$$

where the suffice g refers to the *goal-obstacle*, ng to the *non-goal obstacles* and t to the *trail*. The suffices UB and LB are short for *Upper Bound* and *Lower Bound*. So, d_t is the distance from the current cell c center to the *trail*, while $d_{t,UB}$ and $d_{t,LB}$ are respectively the upper and lower bounds for the function presented in figure 3.2, where $\alpha_t, \alpha_g = 2$. Analogously for d_g and d_{ng}. See also figure 3.3. The *non-goal obstacles*' distance d_{ng} used is the one of the closest obstacle to the cell being evaluated. Notice that the volume near the obstacles is modeled similarly to the obstacles' APF in [17]. The parameter h_{max} sets the highest possible increment a single component of the function can contribute. $A_t(c)$ is a generic activation function that sets the volume corresponding to a cell c too close to the *trail* to zero.

This function's values are evaluated cell by cell all over the height-map. For each cell c a collision box with square base, with $side = \sqrt{\frac{x_{dim} y_{dim}}{MAX_{cells}}}$ and height $h(c)$ is instantiated such



(a)



(b)

Figure 3.4: Visualization of a scene in PhysX before (a) and after (b) volume generation

that the cell corresponds with the box's base. When $h(c) = 0$ no box is instantiated.
In figure 3.4 we can see an example of the volume generation in one of the use cases where the solution has been evaluated, visualized using the software PhysX[24]. As we can see the volume is truncated across the robot's nominal poses and over the goal obstacle to the left. Volume peaks are present over the other obstacles, as expected. Notice that, in order to use the collision boxes forming the volume for the next step in the proposed algorithm, there is no need to add them to the physical model of the environment.
Once the volume has been generated, we finally need a way to obtain the jointspace boundaries around the nominal values. Notice that it's not mathematically possible to completely translate the volume into the jointspace, in the general case. Having 6 joints we end up with 12 parameters, but a generic 3D volume cannot be reduced to just 12 parameters. The volume will act therefore as a guide to bound the joints, exercising "pressure" onto the joints.

3.3 Jointspace Exploration

Once the cohesive volume has been obtained, then it has to be used in order to impose one bounding interval for each joint. Therefore, for each joint, it is necessary to find the upper and lower bound. We propose the Algorithm 2 to search and describe the boundaries. The algorithm reported founds the upper boundary. To find the lower boundary the algorithm is symmetrical.

We will now describe Algorithm 2 in order to facilitate its understanding. Suppose we are searching the upper bound for the first joint. The algorithm starts by setting the first joint at its absolute upper bound, while other joints are set at their input, and kept at these values in all the following steps of Algorithm 2. A collision check is performed between the robot and the volume generated in the previous section. If there's no collision, the algorithm halts and returns the absolute upper bound as a result. Otherwise, the first joint is set to the midpoint value between the input value and the absolute boundary. In such case, the algorithm enters a loop, which is depicted in figure 3.5. At each instance of this loop, a collision check is performed as before. If there's a collision, the joint is set to the midpoint between the highest collision-free value and the current joint value. If there's no collision, the joint is set to the midpoint between the current value and the lowest colliding value. The loop halts once the distance between the highest collision-free value and lowest colliding value is less than a given *range*, and returns the highest collision-free value as the resultant upper boundary.

Algorithm 2 Joint Upper Bounding

```
1: A ← jointToBound_nominal
2: B ← UB_max
3: C ← B
4: JointToBound ← B
5: if collision then
6:     B = (A + C)/2
7:     JointsToBound ← B
8: else
9:     return B
10: range ← B − A
11: while range > Δ do
12:     if collision then
13:         C ← B
14:         B ← (A + C)/2
15:         jointToBound ← B
16:     else
17:         A ← B
18:         B ← (A + C)/2
19:         jointToBound ← B
20:     range ← B − A
21:
22: return A
```

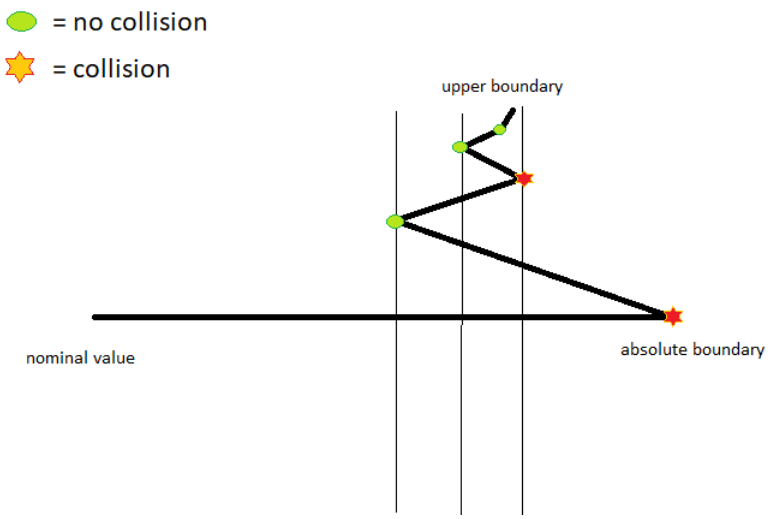


Figure 3.5: Depiction of the loop searching for the upper boundary in Algorithm 2

More in detail: let D be the distance between the input value of the joint we want to bound and r be the range parameter. At the beginning of the loop, D is the distance between the highest collision-free value (input value) and the lowest colliding value (the absolute boundary). At each step of the loop, D is cut in half. Therefore the exit condition can be written as $\frac{D}{2^k} < r$, where k is the number of steps inside the loop. So, the loop is exited when $k > log_2 \frac{D}{r}$. Therefore the whole algorithm 2 takes $k = 1$ steps if it doesn't enter the loop or $k = 1 + \lceil log_2 \frac{D}{r} \rceil$ if it does enter the loop.

In order to find the final boundaries this procedure has to be repeated for each joint with various input sets. In the following, a sequence will be deduced to carry out such evaluations, based on the magnitude of influence each individual joint has on the entire robot.

3.3.1 Joints Hierarchy

In order to evaluate how much each joint influence the position of the others and of the whole robot, the following test was performed in a free environment for a model of the robot system ABB IRB 6700 205-2.80, whose specifics are reported in Figure 3.6. One by one, for each joint a random angular value was added, and the variation in the position of other frames was observed and related to the angular variation. In the following table 3.1 the mean values obtained are reported. Each joint was randomly varied 10 000 times using a standard uniform distribution in the interval between each joint's absolute boundaries. The $cell_{i,j}$ can be read as *joint-i influence on joint-j can be expressed with the such value of the parameter*. Using a loose notation, the 1 on the diagonal tells us that each joint influence itself perfectly, while 0 below the diagonal tells us no joint influence the previous joints. The other values are reported in $\frac{mm}{rad}$.
In tables 3.2, 3.3 and 3.4 are respectively reported the maximum, minimum, and standard deviations obtained with the test described above. The standard deviation is $\sigma = \sqrt{\frac{\sum_i^N (v_i - v_m)^2}{N-1}}$, where v_m is the mean, v_i is one sampled value and the sample has N elements. In chapter 4

	j1	j2	j3	j4	j5	j6
j1	1	39.5225	322.792	235.623	673.823	1197.08
j2	0	1	596.778	1322.18	1496.61	1799.17
j3	0	0	1	96.6101	492.284	1048.88
j4	0	0	0	1	6.73508	4.23817
j5	0	0	0	0	1	6.44586
j6	0	0	0	0	0	1

Table 3.1: Mean of the joints' influence values in $\frac{mm}{rad}$

	j1	j2	j3	j4	j5	j6
j1	1	49.5526	404.711	295.42	844.828	1500.88
j2	0	1	626.153	1387.26	1570.27	1887.72
j3	0	0	1	110.208	561.572	1196.51
j4	0	0	0	1	11.5	7.23657
j5	0	0	0	0	1	7.41628
j6	0	0	0	0	0	1

Table 3.2: Maximum of the joints' influence values in $\frac{mm}{rad}$

	j1	j2	j3	j4	j5	j6
j1	1	4.03894	32.9872	24.0791	68.8603	122.334
j2	0	1	466.366	1033.25	1169.56	1406
j3	0	0	1	42.4665	216.391	461.052
j4	0	0	0	1	9.8511110^{-4}	6.1989810^{-4}
j5	0	0	0	0	1	2.55776
j6	0	0	0	0	0	1

Table 3.3: Minimum of the joints' influence values in $\frac{mm}{rad}$

	j1	j2	j3	j4	j5	j6
j1	0	10.777	88.019	64.2498	183.738	326.42
j2	0	0	33.4064	74.0127	83.7771	100.714
j3	0	0	0	15.1195	77.0426	164.15
j4	0	0	0	0	3.92307	2.46866
j5	0	0	0	0	0	1.08049
j6	0	0	0	0	0	0

Table 3.4: Standard deviation of the joints' influence values in $\frac{mm}{rad}$

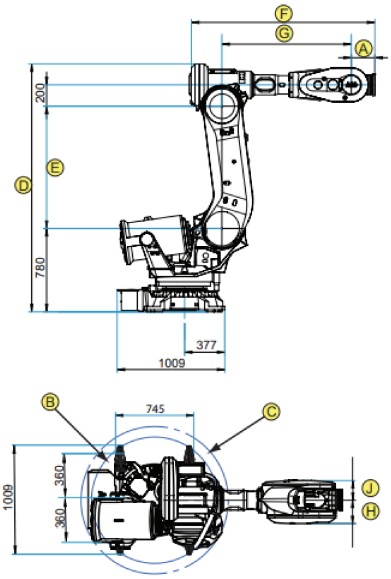


Figure 3.6: Main dimensions of IRB 6700: A=200 mm, B=532 mm, C=633 mm, D=2445 mm, E=1280 mm, F=1670 mm, G=1182.5 mm, H=186 mm, J=209 mm

we will report how the algorithm performs also in the DLR use-case, where the robot has a slightly different structure compared to the other use-cases. A 6-DOF robot arm, analogous to what is described so far, is mounted on a linear axis. As the robot arm maintains the same structure, it has been chosen to treat separately the linear axis and then apply the same Algorithms 3.5 and 3 to the 6-DOF structure. The linear axis does indeed just have the purpose of moving such structure across the workspace, without having any influence on the arm configuration.

The data found confirms that the first three joints are those influencing the overall robot position the most, as we would expect from a 6-DOF robot with revolute joints as the one we are considering. Indeed the rows' average values are:

$$\begin{aligned} I_{j1,avg} &= 600.2035\ mm/rad \\ I_{j2,avg} &= 1303.6845\ mm/rad \\ I_{j3,avg} &= 545.9247\ mm/rad \\ I_{j4,avg} &= 5.486625\ mm/rad \\ I_{j5,avg} &= 6.44586\ mm/rad \\ I_{j6,avg} &= 0\ mm/rad \end{aligned} \tag{3.2}$$

where with $I_{jk,avg}$ is indicated the average influence of the k-th joint. The last 3 joints only affect the roll and pitch of the end-effector and thus do not influence the collision of the whole robot with the obstacles in the workspace. Therefore, they can be bounded arbitrarily. This same choice was adopted in the analysis by Incremona[9] et al.

See Figure 3.7 to better visualize how each joint influence the whole robot.
Given such assumptions, we propose the following Algorithm 3 in order to find the boundaries around a given point in the jointspace. With $Bound(j_k)$ we refer to applying the Algorithm 2 to find the upper boundary of the k-th joint and its symmetrical version to find the lower boundary. To sum it up, the idea is to start from the first joint, which affects all the robot structure, and find its boundaries using algorithm 2 while the other joints are at their nominal values. Boundaries on the second joint are computed as the most restrictive ones found by setting the other joints at nominal values and the first joint at its upper boundary and then at its lower boundary. For the third joint all the combinations of lower and upper boundaries on the first and second joint are used.

Algorithm 3 Joints Bounding

1: $Bound(j_1) \rightarrow j_{1,LB}, j_{1,UB}$
2:
3: $j_1 \leftarrow j_{1,LB}$
4: $Bound(j_2) \rightarrow j_{2,LB}, j_{2,UB}$
5: $j_1 \leftarrow j_{1,UB}$
6: $Bound(j_2) \rightarrow j_{2,LB}, j_{2,UB}$
7:
8: $j_1 \leftarrow j_{1,LB}$
9: $j_2 \leftarrow j_{2,LB}$
10: $Bound(j_3) \rightarrow j_{3,LB}, j_{3,UB}$
11: $j_2 \leftarrow j_{2,UB}$
12: $Bound(j_3) \rightarrow j_{3,LB}, j_{3,UB}$
13: $j_1 \leftarrow j_{1,UB}$
14: $j_2 \leftarrow j_{2,LB}$
15: $Bound(j_3) \rightarrow j_{3,LB}, j_{3,UB}$
16: $j_2 \leftarrow j_{2,UB}$
17: $Bound(j_3) \rightarrow j_{3,LB}, j_{3,UB}$
18:
19: **return** $\{j_{1,LB}, j_{1,UB}, j_{2,LB}, j_{2,UB}, j_{3,LB}, j_{3,UB}\}$ =0

Previously we described in section 3.1 how the volume shouldn't be too much narrow, otherwise, it could restrict the boundaries too much, to the point that an interval could collapse to the sole nominal value. This scenario should be avoided by carefully defining the height function and doing proper testing, but as a secondary safe measure, it could be performed, after Algorithm 3, a check of the boundaries obtained. If there's an instance in which the lower boundary, the upper boundary, and the nominal value are the same, then this interval has collapsed. In order to go on, an arbitrary (but small) interval could be set around the nominal value.
In the previous section, we evaluated how many steps Algorithm 2 does take. We'll now evaluate the whole solution from a computational perspective.

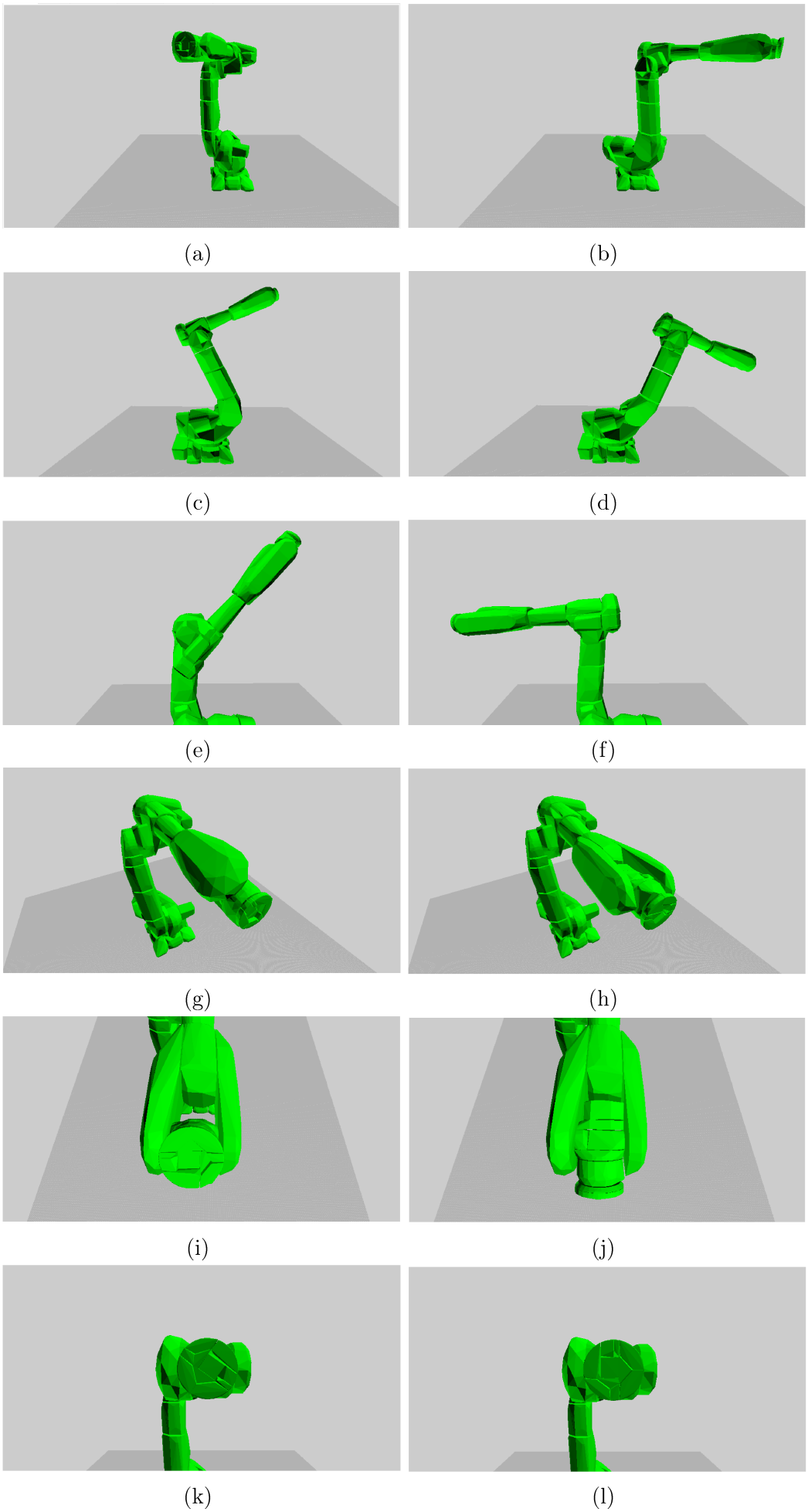


Figure 3.7: Moving the various joints. Joint1 to Joint6, from top to bottom

3.4 Theoretical Time Analysis

In this section, the proposed algorithm will be analyzed from a computational perspective. In particular, we want to observe how the dimensions of the height-map, and so the final amount of collision boxes generated, influence the computing time.
Let the global planner output a set of N points in the jointspace, which represent the nominal path. Let the height-map be composed of $m \times n$ cells. Let the goal-position of the end-effector be $\mathbf{p_g} = [x_g, y_g, z_g]^T \in \mathbb{R}^3$. Let the robot have L joints. Suppose there are N_O obstacles in the environment.

The Object Categorization step requires first identifying the goal obstacles, which can be completed in a finite amount of steps depending on the parameter Δ_z (see algorithm 1). More explicitly the steps required are $\frac{z_g}{\Delta_z}$. To define the *trail* we just need to take all the links' collision boxes setting the robot at each point in the nominal path. Therefore getting the *trail* requires NL steps. We call this finite amount of time T_1.

In order to generate the volume, it's just needed to evaluate the height function cell by cell. Notice that at each evaluation just one distance needed to be evaluated for the goal and the *trail*, while the shortest distance of the obstacles must be found. So $2 + N_O$ distances have to be evaluated for each cell in order to compute the height function of that cell, and so $T_2 = T_2(N_O)$.
Still, we end up with $C \leq m \times n$ cells, as cells with null height function value do not instantiate collision boxes.

As already stated in section 3.3 algorithm 2 takes 1 or $k = 1 + \lceil log_2 \frac{D}{r} \rceil$. Here the most expensive operation in terms of computational time, which is checking if the robot is colliding with the volume, is executed. This single operation takes LC steps, as each link is confronted with each component of the whole volume. Algorithm 3 performs such operations $(1 + 2 + 2^2) = 7$ times for both the upper and the lower boundaries.
Therefore the total time for this step T_3 is bounded:

$$14CLN \leq T_3 \leq 14CLN(1 + \left\lceil log_2 \frac{D}{r} \right\rceil) \tag{3.3}$$

This tells us that the discretization of the height-map is the main parameter under our control to manage the computational time of the proposed method. This relation, represented by C, appears to be linear, as well as the other relations with the not controllable parameters L (since the robot is given), N_O (since also the environment is given), and N. Based on this analysis, the expected computational time is therefore smaller than what could have been expected with the method described in chapter 2, particularly the one based on cylindrical algebraic decomposition [20], which has a double exponential complexity.

Notice that N also has a linear impact on the computational time.

Chapter 4

Experimental Results

In this chapter, we are going to describe the methodologies used to test the proposed solution and report and comment on the results achieved. The test was performed in the 3 use-cases of the European project DrapeBot presented in Chapter 1: Dallara, Baltico and DLR. Dallara belongs to the automotive sector, Baltico to shipbuilding, and DLR to the aerospace industry. Each workcell is represented using a tree-like structure, where each object is instantiated as a node from the root. Each object then has its own branches and leaves, containing the transform and the 3D models. The structure is depicted in Figure 4.1, with a simplified example of a workcell with two objects and a 3-DOF robot. TF stands for *Transform*.
Notice that the algorithm proposed in Chapter 3 was developed through Dallara and Baltico and later adapted, as it will be explained in later sections in this chapter, to the third use-case DLR.

4.1 Methodology

The typical operational sequence for the process of draping would be:

- *home-patch*: the robot goes from the home pose to the carbon-fiber patch and picks it up
- *patch-mould*: the carbon fiber patch is transferred to the mould and then draped
- *mould-home*: the robot comes back to the home pose, and a new patch is prepared

Figure 4.2 depicts the process workflow.
Given a workspace, the algorithm has been tested for the paths defined above and summarized in Figure 4.2. In order to analyze the performance of our approach, we collected the following data:

- the time to generate the volume, the time to find the start boundaries, and the time to find the end boundaries
- the numerical values of the boundaries

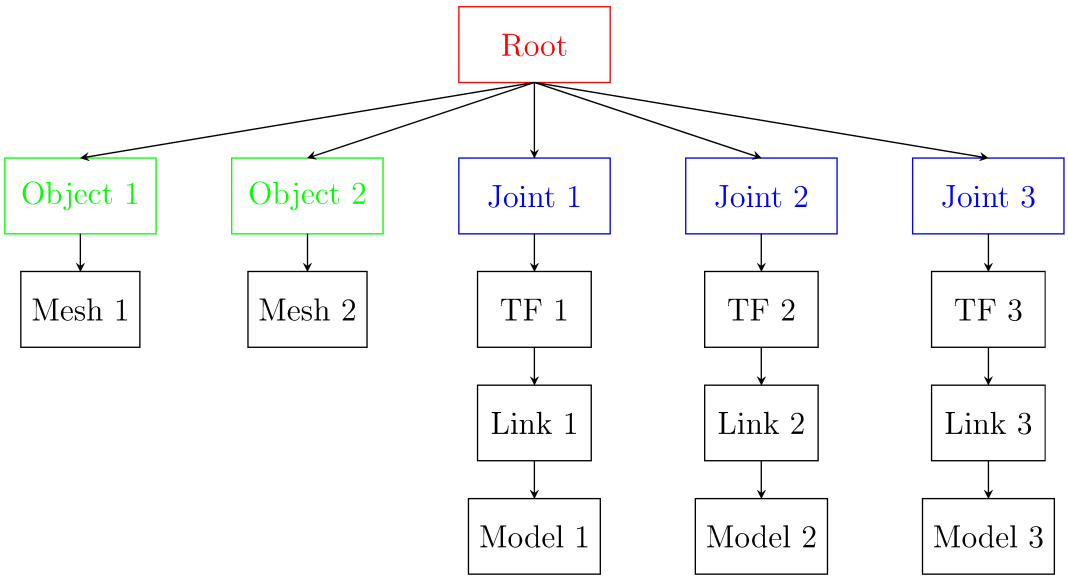


Figure 4.1: Abstract Tree-like structure representing the workcell

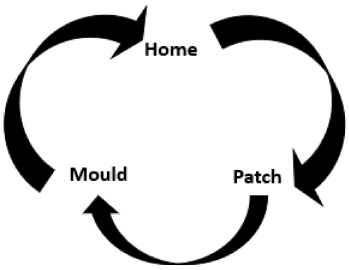


Figure 4.2: the home-path-mould operating cycle

- the number of instantiated collision boxes for the volume
- the percentage of the absolute joints' boundaries that we have restricted to
- the Confidence Factor (CF) of the boundaries providing collision-less deformations of the path

Each path was tested by setting the maximum number of collision boxes (i.e. the number of cells of the height-map) over the following values: 100, 150, 200, 500, 750, 1000, 1500, 2000, 3000, 5000, 7500, 10000, 12000, 15000, 20000, 25000, 30000, 40000, 50000, 100000.
The CF has been computed as the average number of collision-less random joints configuration encountered in a time window. As the nominal path can be composed of linearly connected configurations, we arbitrarily divided the path into 30 windows within this linear law and did the same with the boundaries. In each time window 10000 random joints' configurations, with values within the time window's boundaries, are tested for collision. The aim of the CF is to express the goodness of the boundaries produced. A value of CF equal to 100% would mean that there's no possibility to let the robot collide with the obstacles while its joints take values inside the boundaries.
Taking equation 3.1 as a reference and after some fine-tuning, the following values were used for the parameters:

- $\alpha_t = \alpha_g = \alpha_{ng} = 1$
- $d_{t,LB} = 350\ mm$, $d_{t,UB} = 1500\ mm$
- $d_{g,LB} = 100\ mm$, $d_{g,UB} = 2000\ mm$
- $d_{ng,UB} = 500\ mm$
- $h_{max} = 1500\ mm$
- the activation function $A_t(c)$ set zero height in all the cells more distant than 2800 mm for Dallara and Baltico and 3000 mm for DLR, as well as all those that are right below the trail.

The activation function's parameter is set at a value such that no box is instantiated outside the robot's reach. Also, no box is instantiated below the trail. This check is rapidly performed checking if any box in the trail would collide with a max height box instantiated in the cell. If not, the height is normally computed using the height function. Some slight modifications have been introduced with respect to the idealized version of equation 3.1. The second component of the function, computing the contribution of the distance from the goal, is split into a constant term and a term dependent on the distance:

$$h(c)_{goal} = \frac{1}{3}h_{max} + \frac{2}{3}h_{max}\left(\frac{d_g - d_{g,LB}}{d_{g,UB} - d_{g,LB}}\right)^{\alpha_g} \tag{4.1}$$

Combined with the elimination of boxes below the trail, this minor change to the height function allows us to grow the volume around the goal a bit faster than the rest. Indeed this change allows us to start from a positive height close to the goal while keeping the area coincident with the goal free from any volume. The exponents α_t, α_g and α_{ng} are set to 1 as it was chosen to prefer tighter boundaries. Higher exponents would make the volume scale up in a smoother way, although these exponents have less influence on the final boundaries than the distances' UBs and LBs. As stated above, the other values have been fine-tuned through trial and error. Notice they're round numbers, as the overall method isn't too sensitive to small variations in these parameters.
Tests were performed with an Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz and 8 GB of RAM. In what follows we report the data collected from the experiments. Notice that the frame corresponding to the first joint is at the origin of the reference system. The nominal paths are computed by the global planner, taking into input the starting and the goal position of the robot gripper.

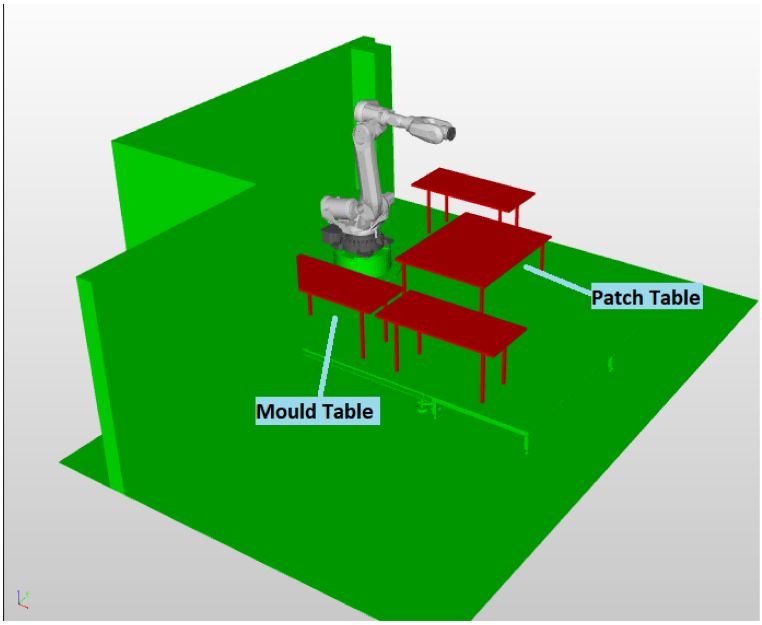
4.2 Collected Data

In this section, we are going to report and comment on the experimental data obtained testing the algorithm in the presented use-cases. We will report the Dallara use-case first as it was the first one to be tested. For each path the boundaries obtained and the CF will be briefly commented on.

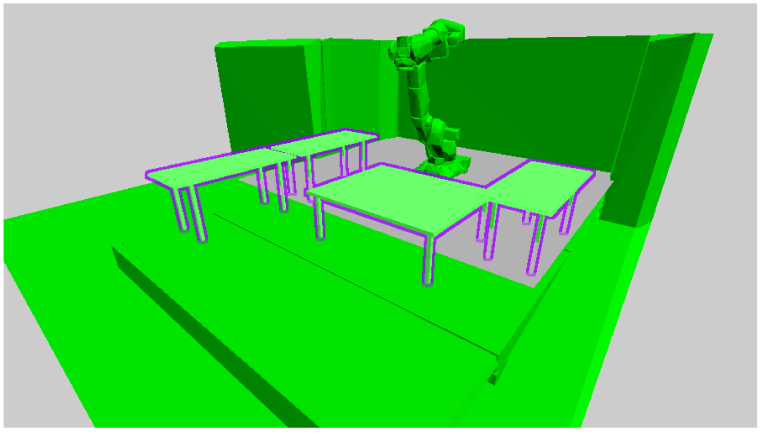
4.2.1 Use-Case Dallara

The first use-case is depicted in figure 4.3. Beyond the home position, the final poses given in input to the global planner are:

- one position above the table which will hold the patch in the real setting: $(943, -1910, 800)\ mm$
- one position above the table which will hold the mould in the real setting: $(1817, 467, 1800)\ mm$



(a)



(b)

Figure 4.3: Visualization of Dallara workspace with the robot at home pose using custom GUI (a) and PhysX (b)

For the *home-patch* path the nominal values for the joints, as computed by the global planner, are:

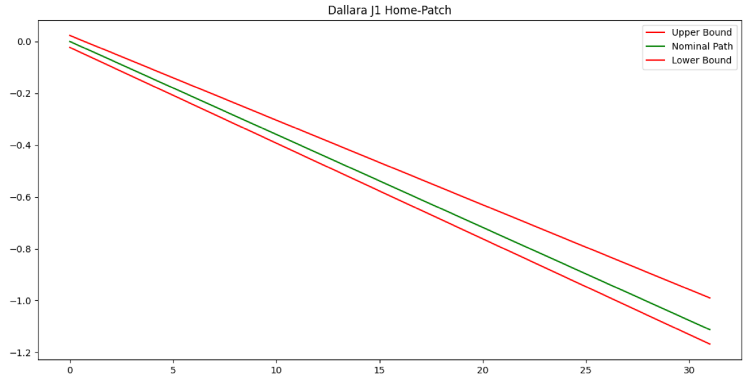
- $\{0, 0, 0, 0, 0, 0\}$ rad at home;
- $\{-1.11219, 0.734922, 0.0866255, -3.14159, -0.749248, -1.11219\}$ rad over the patch table;

Tests results are reported in table 4.1.

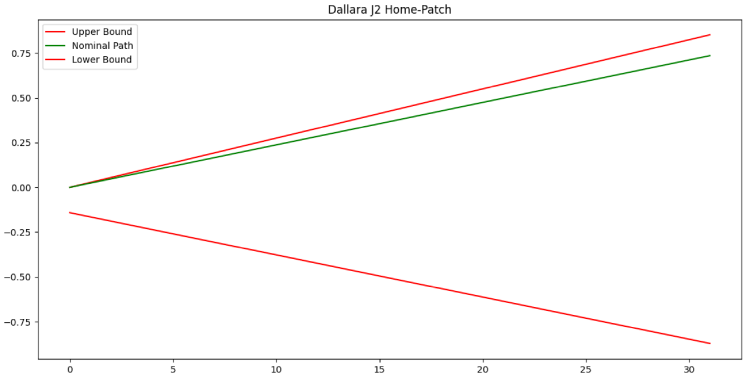
Max CBox	**CBox**	**Computational Time [ms]**	**Confidence Factor**
100	43	82	95.3839
150	62	69	95.7548
200	88	90	95.8161
500	227	121	98.9903
750	351	165	98.8935
1000	457	170	99.5000
1500	704	209	99.2742
2000	940	231	99.5903
3000	1424	284	99.8613
5000	2393	387	99.8548
7500	3618	551	99.9968
10000	4834	609	99.9871
12000	5794	771	99.8387
15000	7249	1000	99.9226
20000	9685	1149	99.8258
25000	12148	1527	99.9774
30000	14600	1771	99.9742
40000	19461	2319	99.9839
50000	24348	2680	99.8581
100000	48829	5782	99.9903

Table 4.1: Results of *home-patch* path, use-case Dallara

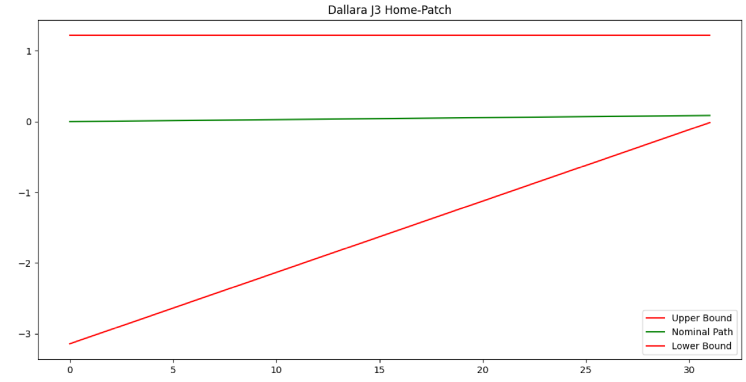
We can see the CF steadily increasing with the number of collision boxes composing the volume. As well, computational time increases, but we manage to have CF over 99% for just 170 ms spent.
In Figure 4.4 a visualization of how the boundaries behave while the robot moves towards the goal. The first joint is kept in a tight range, the second joint acquires more space getting close to the goal while the third loses mobility in one direction. This can be interpreted as the robot's rotation around the z-axis getting restricted as the robot reaches the patch table, while more freedom is added to the second joint allowing the robot to regulate the proximity with the table. The third joint's boundaries get narrower, as it controls the arm yam and it must prohibit the arm to crash unto the table. The values depicted are those obtained with 20000 collision boxes as a maximum. This value was observed to generally produce a high CF. Indeed also the next images of the boundaries will be obtained with 20000 set as a maximum.



(a)



(b)



(c)

Figure 4.4: Possible boundaries through the *home-patch* path, use-case Dallara; Joint1 (a), Joint2 (b) and Joint3(c)

Regarding the *patch-mould* path the nominal values for the joints, as computed by the global planner, are:

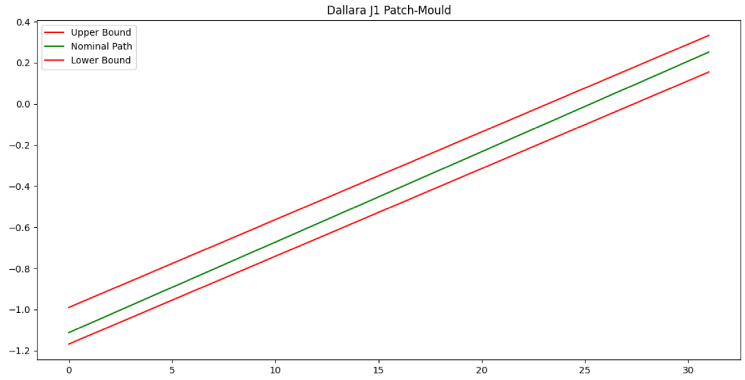
- $\{-1.11219, 0.734922, 0.0866255, -3.14159, -0.749248, -1.11219\}$ *rad* over the patch table;
- $\{0.251572, 0.282489, -0.107214, -3.14159, -1.39552, 0.251572\}$ *rad* over the mould table;

Tests results are reported in table 4.2.

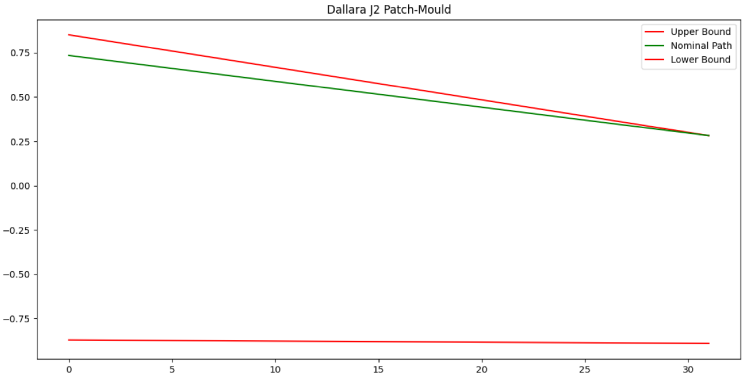
Max CBox	**CBox**	**Computational Time [ms]**	**Confidence Factor**
100	48	71	93.4484
150	70	89	94.6000
200	95	98	92.8516
500	248	151	99.1484
750	379	174	98.6484
1000	504	191	99.3742
1500	769	215	99.3032
2000	1036	246	99.7742
3000	1565	296	99.8226
5000	2630	411	99.7742
7500	3961	567	99.9774
10000	5288	717	99.9613
12000	6360	815	99.8677
15000	7969	1329	99.9161
20000	10642	1584	99.8000
25000	13326	1593	99.9903
30000	16018	2105	99.9968
40000	21370	2636	99.9774
50000	26735	3340	99.8677
100000	53574	7888	99.9968

Table 4.2: Results of *patch-mould* path, use-case Dallara

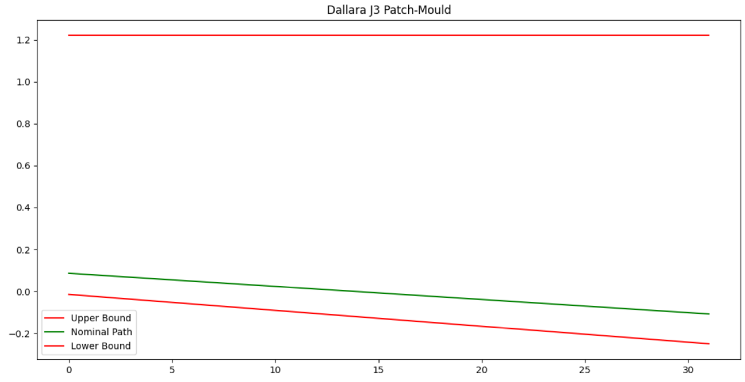
We can observe how, for lower discretization, the CF is lower than for the *home-patch* path, although it still goes above 99.9% for higher discretization. The *patch-mould* path appears to be the hardest of the three, as it is amidst two main obstacles, the patch table, and the mould table. In figure 4.5 we can observe how this time the first joint is kept in a narrow interval during the whole path, as the robot goes from one table to another table directly. The second joint upper boundary and the third joint lower boundary keep a more or less steady value during the whole trajectory, as the proximity to the tables is kept stable since both tables have the same height.



(a)



(b)



(c)

Figure 4.5: Possible boundaries through the *patch-mould* path, use-case Dallara; Joint1 (a), Joint2 (b) and Joint3(c)

Regarding the *mould-home* path the nominal values for the joints, as computed by the global planner, are:

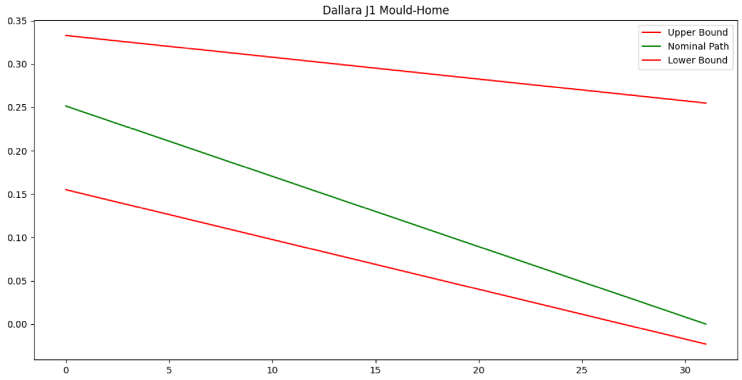
- $\{0.251572, 0.282489, -0.107214, -3.14159, -1.39552, 0.251572\}$ over the mould table;
- $\{0, 0, 0, 0, 0, 0\}$ at home;

Tests results are reported in table 4.3.

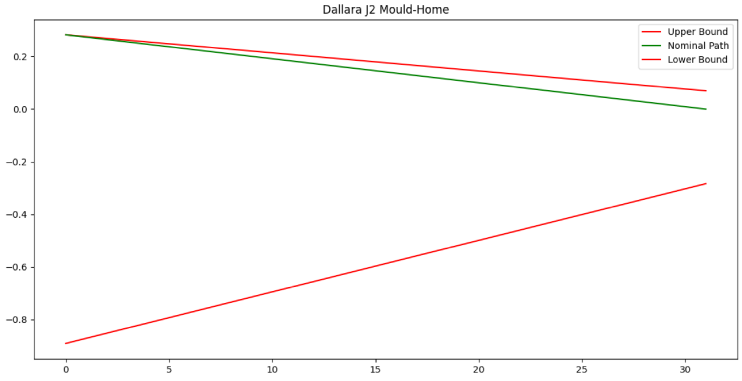
Max CBox	**CBox**	**Computational Time [ms]**	**Confidence Factor**
100	44	71	99.9484
150	64	78	98.4806
200	89	86	100
500	225	119	100
750	345	164	100
1000	456	173	100
1500	701	206	100
2000	933	202	100
3000	1414	287	100
5000	2378	365	100
7500	3578	527	100
10000	4761	598	100
12000	5709	689	100
15000	7160	837	100
20000	9547	1224	100
25000	11963	1829	100
30000	14364	1600	100
40000	19159	2015	100
50000	23948	3023	100
100000	47974	5325	100

Table 4.3: Results of *mould-home* path, use-case Dallara

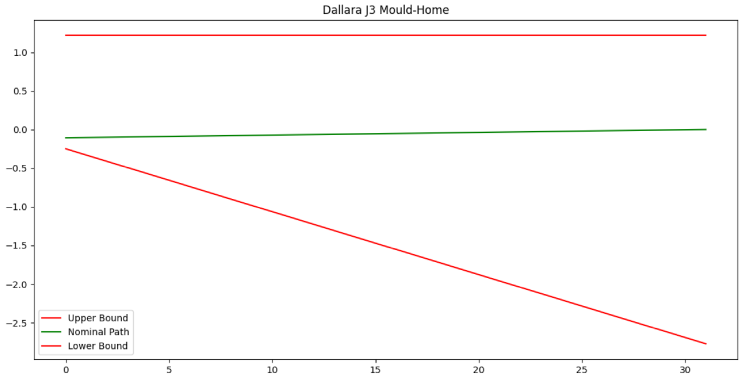
We can notice the CF remarkably going to 100% even for very low discretization and maintaining steadily the value. The *mould-home* path is clearly the easiest path, as the robot steps away from the obstacle into a safe pose. In figure 4.6 we can observe the first and third joint boundaries widening up going toward the home point. Curiously the second joint narrows, but as the CF is so high we can deduce that this is not likely to be for obstacle avoidance but rather just for pushing toward the home point target.



(a)



(b)



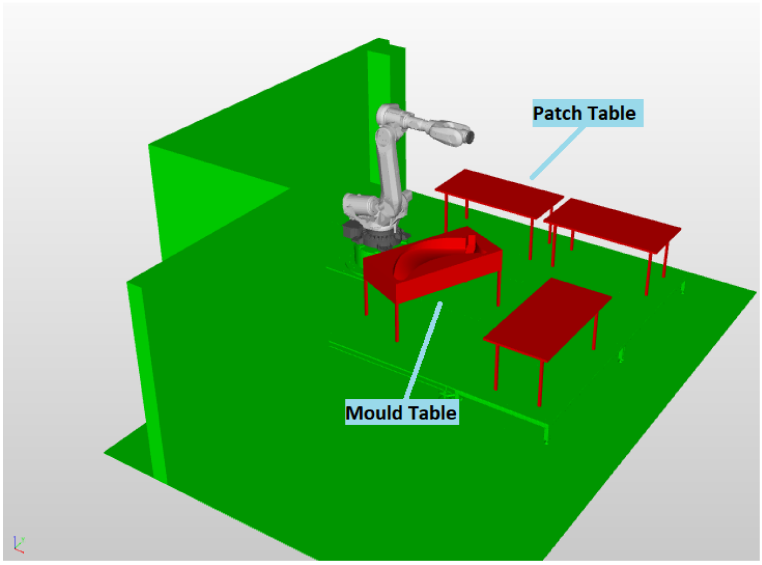
(c)

Figure 4.6: Possible boundaries through the *mould-home* path, use-case Dallara; Joint1 (a), Joint2 (b) and Joint3(c)

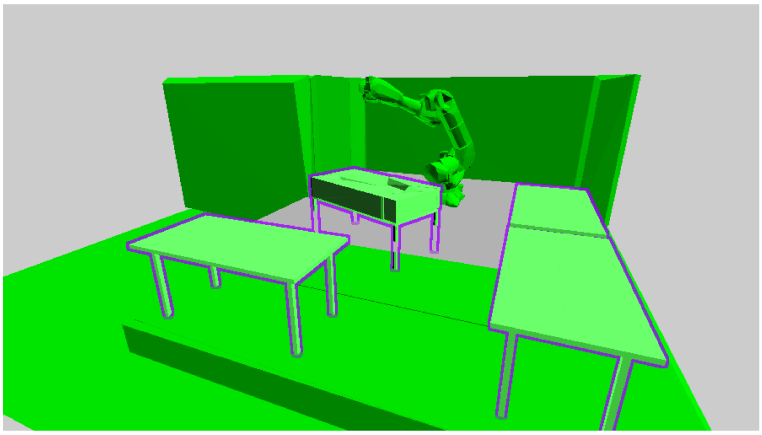
4.2.2 Use-Case Baltico

The second use-case is depicted in figure 4.7. Notice this model of the workspace also presents a mould on the corresponding table. Beyond the home position, the other gripper's position given in input to the global planner are:

- one position above the table which will hold the patch in the real setting: $(1218, 1809, 800)\ mm$
- one position above the table which will hold the mould in the real setting: $(1381, -844, 1800)\ mm$



(a)



(b)

Figure 4.7: Visualization of Baltico workspace with the robot at home pose using custom GUI (a) and PhysX (b)

The nominal values for the joints for the *home-patch* path, as computed by the global planner, are:

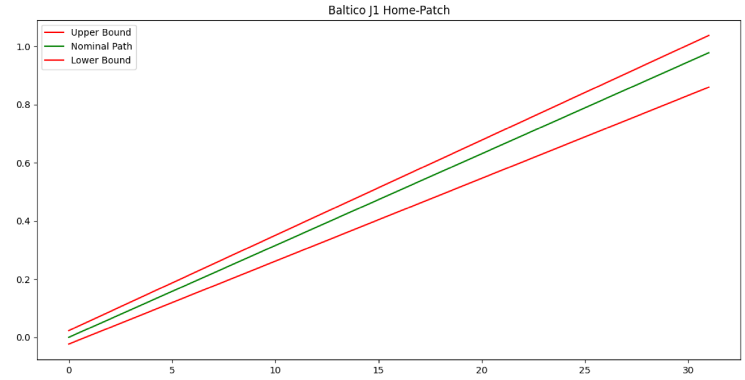
- $\{0, 0, 0, 0, 0, 0\}$ *rad* at home;
- $\{0.978215, 0.766918, 0.0255917, -3.14159, -0.778286, 0.978215\}$ *rad* over the patch table;

Tests results are reported in table 4.4.

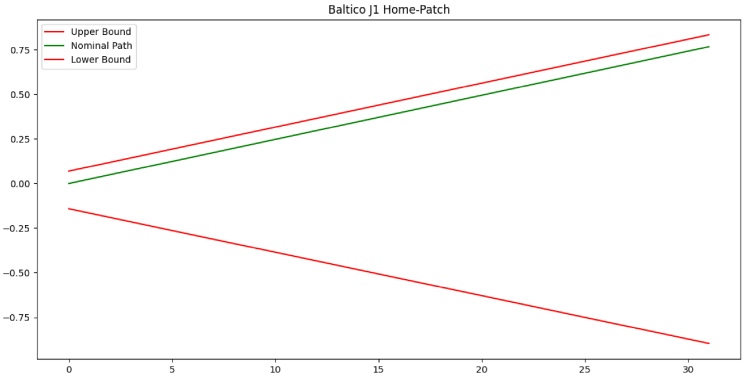
Max CBox	**CBox**	**Computational Time [ms]**	**Confidence Factor**
100	50	67	96.9968
150	72	79	97.5161
200	103	90	98.6097
500	264	143	98.7613
750	400	166	99.1581
1000	549	197	98.3903
1500	823	216	99.3581
2000	1100	225	99.8806
3000	1658	313	99.371
5000	2822	447	99.6548
7500	4258	600	99.8452
10000	5660	681	99.8935
12000	6781	867	99.8581
15000	8531	1157	99.8774
20000	11384	1316	99.771
25000	14285	1766	99.7774
30000	17141	1788	99.8387
40000	22837	2353	99.8548
50000	28623	3068	99.8613
100000	57415	6683	99.929

Table 4.4: Results of *home-patch* path, use-case Baltico

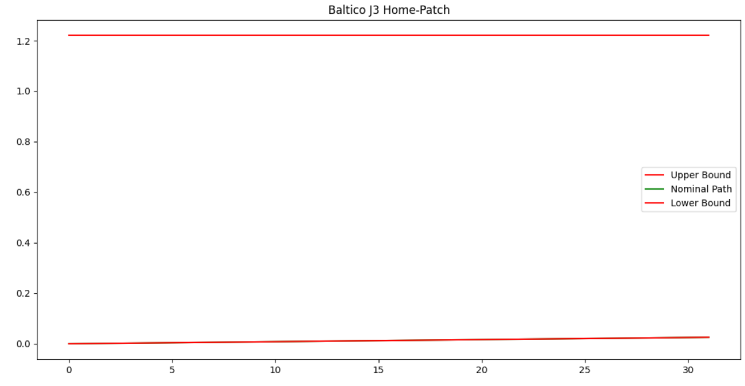
As we can see in Figure 4.8, the situation is pretty similar to the corresponding case of the Dallara use-case in figure 4.4, concerning the first two joints. The third, on the other hand, keeps a quite wide interval for the whole path. One likely explanation is that the end-effector goal point is at a position that leaves the arm sufficiently away from the table, thus allowing a good deal of mobility. We can observe the CF stably surpassing 99% quite soon, from a 1500 maximum amount of collision boxes onward, with a low computational time of 216 *ms*.



(a)



(b)



(c)

Figure 4.8: Possible boundaries through the *home-patch* path, use-case Baltico; Joint1 (a), Joint2 (b) and Joint3(c)

The nominal values for the joints for the *patch-mould* path are:

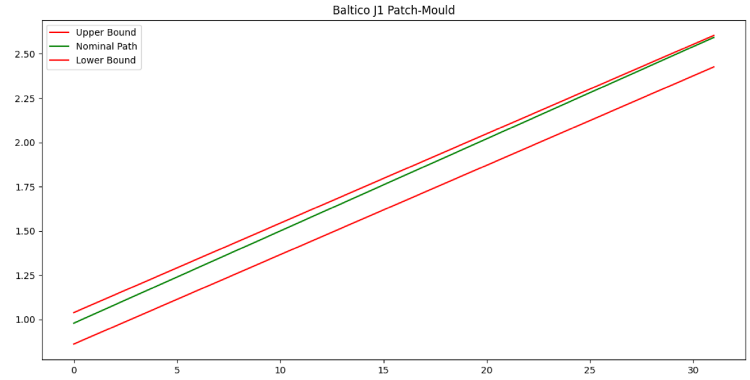
- $\{0.978215, 0.766918, 0.0255917, -3.14159, -0.778286, 0.978215\}$ rad over the patch table;
- $\{2.59301, -0.629514, -2.18935, 1.00597e-16, -1.89352, -0.548579\}$ rad over the mould table;

Tests results are reported in table 4.5.

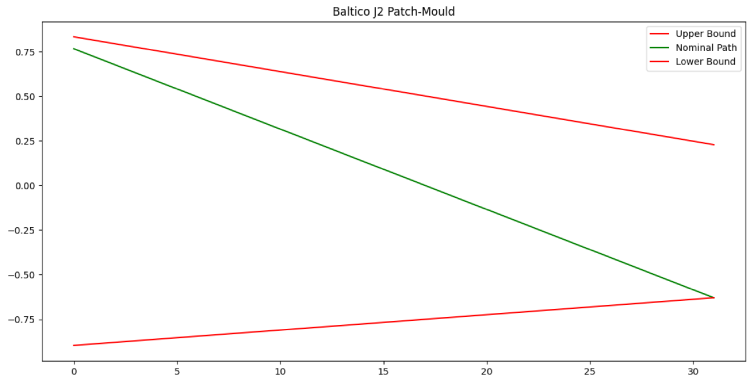
Max CBox	**CBox**	**Computational Time [ms]**	**Confidence Factor**
100	54	72	95.8742
150	78	93	95.6226
200	109	95	97.9226
500	280	144	97.529
750	425	176	98.5258
1000	588	190	97.3839
1500	878	215	98.0355
2000	1175	266	99.6226
3000	1772	330	98.8806
5000	3013	465	99.3935
7500	4538	665	99.7516
10000	6040	802	99.6742
12000	7251	948	99.6645
15000	9120	1099	99.7323
20000	12168	1454	99.3903
25000	15240	1785	99.6839
30000	18301	2096	99.6645
40000	24395	2876	99.6484
50000	30566	3689	99.5387
100000	61276	8010	99.7452

Table 4.5: Results of *patch-mould* path, use-case Baltico

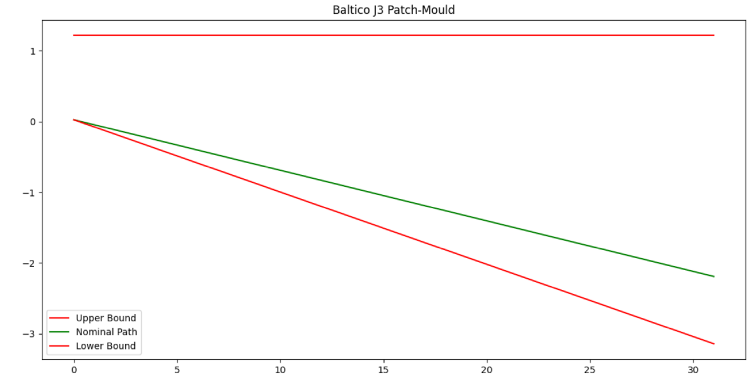
The situation depicted in figure 4.9 is analogous to the *patch-mould* path in Baltico, with narrow boundaries on the first joint, the second joint's ones getting narrower and the third's widening. It is peculiar that for the second joint, the nominal value also constitutes the lower bound, meaning that the end-effector goal is at a critical position. We notice overall high values for the CF, but as it was for Dallara, this path presents the lowest CF for this whole use-case, at low discretization, with 95.8742%. The *patch-mould* path confirms to be the hardest.



(a)



(b)



(c)

Figure 4.9: Possible boundaries through the *patch-mould* path, use-case Baltico; Joint1 (a), Joint2 (b) and Joint3(c)

For the *mould-home* path the nominal values for the jointspace:

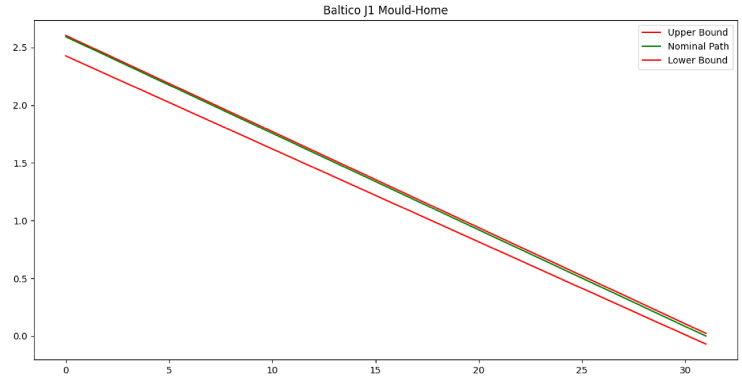
- $\{2.59301, -0.629514, -2.18935, 1.00597e-16, -1.89352, -0.548579\}$ *rad* over the mould table;
- $\{0, 0, 0, 0, 0, 0\}$ *rad* at home;

Tests results are reported in table 4.6.

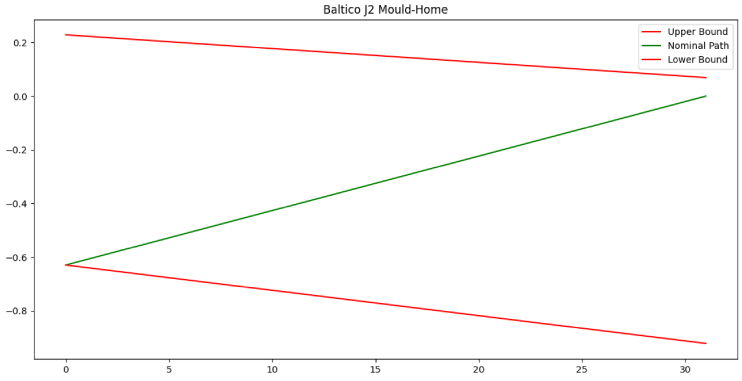
Max CBox	**CBox**	**Computational Time [ms]**	**Confidence Factor**
100	43	64	99.929
150	63	83	99.9419
200	85	85	99.9355
500	216	122	100
750	333	161	100
1000	442	181	100
1500	676	199	100
2000	894	221	100
3000	1357	323	100
5000	2282	411	100
7500	3454	523	100
10000	4611	745	100
12000	5524	793	100
15000	6912	1046	100
20000	9222	1360	100
25000	11561	1376	100
30000	13891	1614	100
40000	18519	2280	100
50000	23161	3160	100
100000	46436	5954	100

Table 4.6: Results of *mould-home* path, use-case Baltico

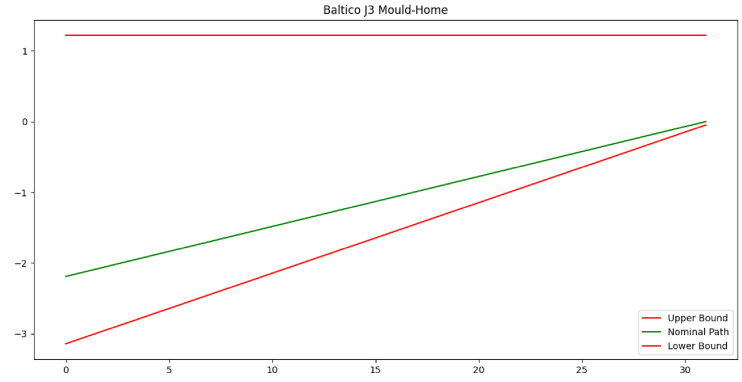
As it was for Dallara the *mould-home* path appears to be the easiest, giving raise to an early, in terms of maximum number of boxes, CF at 100%. Figure 4.10 shows that boundaries again attempt to guide the joints toward home. The first joint follows a narrow path, where the starting and ending points are also quite apart due to the tables' disposition in the workcell. Differently from Dallara, this workcell leaves room for the second joint to variate, but the third joint is on the other hand pushed towards its lower boundaries, again to push the robot toward home.



(a)



(b)



(c)

Figure 4.10: Possible boundaries through the *mould-home* path, use-case Baltico; Joint1 (a), Joint2 (b) and Joint3(c)

4.2.3 Use-Case DLR

The proposed solution has been tested, with some modifications, in a third use-case which will be named DLR, operating in the aerospace industry. Here the robot has a linear axis to move across the workspace, and therefore many relevant obstacles lie close to the robot at any moment. Figure 4.11 depicts the CAD model of the workcell. Obstacles' proximity gave rise to some low CF in the first few attempts. To overcome this, the obstacles' collision boxes were added to the final volume. This improved the CF significantly, without noticeably raising the computing time. Indeed we are just adding a couple of collision boxes to a set of a few hundred or even thousands (depending on the discretization) of boxes.
Another peculiarity of this use-case is the aforementioned linear axis. Since this does influence the overall spatial positioning but not how the arm orients in space, its way to influence collisions has been considered to be minor with respect to the first three joints. Therefore, referring to Algorithm 3, we just added an arbitrary 3% tolerance to the nominal value of the linear axis, before proceeding with the other steps as usual. This value was obtained after some fine-tuning tests.

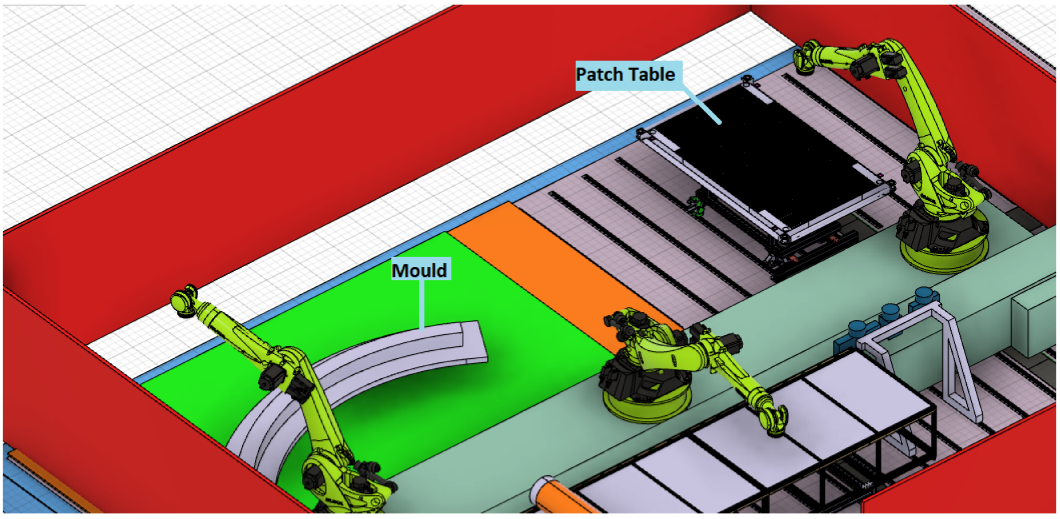


Figure 4.11: CAD model of the DLR workcell

The end-effector positions corresponding to the various goals are:

- $(-916.442, 2115, 800)$ mm over the patch table, corresponding to $\{-1.14135, 1.775, -1.36, 1.72, -0.0237, 0.9, 1.198\}$ rad in the jointspace;
- $(-7601.93, 1702.58, 1200)$ mm over the mould, corresponding to $\{-5.4, 1.175, -0.895, 1.294, -0.014, 1.246, 1.317\}$ rad in the jointspace;
- the home joints are set at $\{-2.19246, 1.303, -1.69, 1.848, -0.023, 1.423, 1.194\}$.

In tables 4.7, 4.8, and 4.9 the main results are reported as in the previous use-cases. In figures 4.12, 4.13 and 4.14 we can observe the boundaries linearly interpolated over the nominal trajectory as in the previous use-cases. From the tables, we notice how the CF is

comparable to the ones in Dallara and Baltico for what concerns the *patch-mould* and *mould-home* paths. The *home-patch* path on the other hand shows a slight underperformance, likely due to the proximity of the linear axis to the table for the patch, as well as having deliberately tried to move the pinpoint above the center of such table. This position is likely one of the hardest, as the robot has to stretch the arm over the table and very close to the table, but being so goal obstacle some volume is removed from its surroundings leaving more freedom to the joints than in the other scenarios. The final CF can still be considered good, as it stably goes above 90%. The other paths behave similarly to the previous use-cases, widening the boundaries towards the home pose and tightening it towards the goal point over the mould.

Max CBox	**CBox**	**Computational Time [ms]**	**Confidence Factor**
100	37	398	90.5355
150	52	479	91.1097
200	79	453	92.9387
500	199	453	92.5355
750	305	472	92.8419
1000	417	504	91.7452
1500	629	522	91.5839
2000	866	562	93.2097
3000	1309	648	93.0548
5000	2239	772	92.7677
7500	3328	975	93.3129
10000	4440	1124	92.7194
12000	5345	1208	92.9839
15000	6742	1452	92.8194
20000	9055	1657	92.9226
25000	11276	1898	93.0742
30000	13491	2222	93.0645
40000	18055	2816	93.2581
50000	22714	3583	92.9419
100000	45567	7209	92.8355

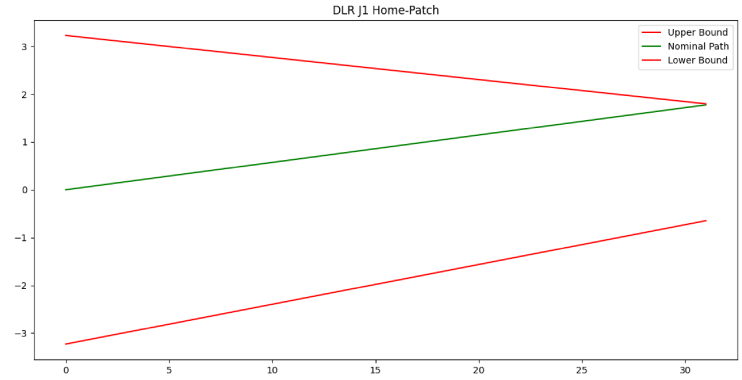
Table 4.7: Results of *home-patch* path, use-case DLR

Max CBox	CBox	Computational Time [ms]	Confidence Factor
100	66	498	88.1
150	89	497	88.4097
200	136	575	89.5581
500	336	670	87.1032
750	529	618	86.7935
1000	713	552	87.8
1500	1084	534	93.729
2000	1464	723	92.6613
3000	2204	885	94.7742
5000	3793	975	95.9032
7500	5655	1415	95.9774
10000	7518	1684	97.0613
12000	9051	1617	96.971
15000	11353	2056	98.1613
20000	15286	2434	98.1935
25000	18987	3822	98.0226
30000	22827	3427	98.1323
40000	30506	4839	96.9839
50000	38319	5932	98.0613
100000	76690	13646	98.1387

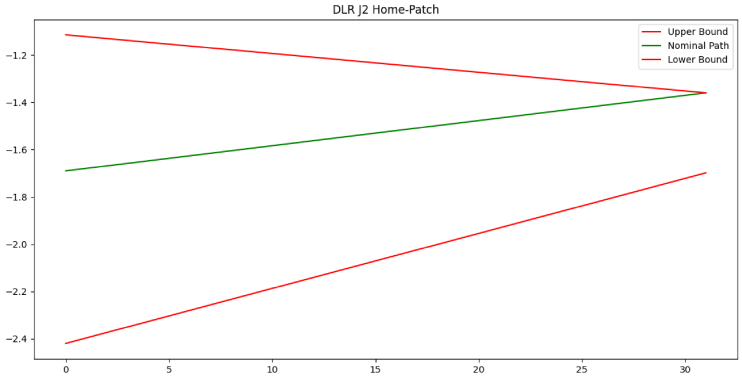
Table 4.8: Results of *patch-mould* path, use-case DLR

Max CBox	CBox	Computational Time [ms]	Confidence Factor
100	52	884	93.7387
150	82	542	92.7839
200	118	581	91.9548
500	305	538	99.8774
750	477	508	99.9968
1000	643	557	99.9516
1500	961	607	100
2000	1293	640	100
3000	1963	712	100
5000	3342	984	100
7500	5003	1061	100
10000	6661	1222	100
12000	8041	1336	100
15000	10046	1817	100
20000	13475	2419	100
25000	13475	2419	100
30000	20229	2763	100
40000	27033	3406	100
50000	33794	4790	100
100000	67751	10320	100

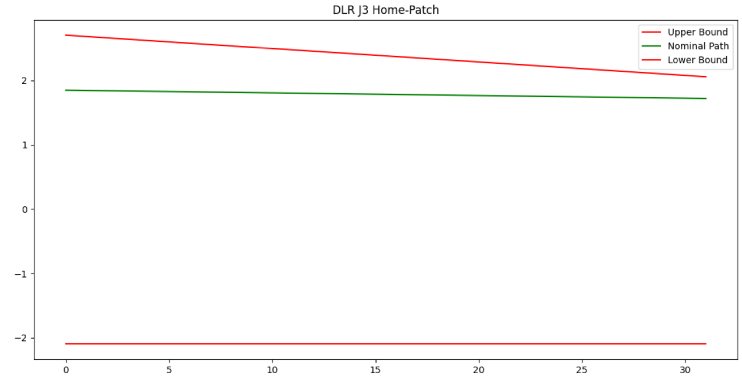
Table 4.9: Results of *mould-home* path, use-case DLR



(a)

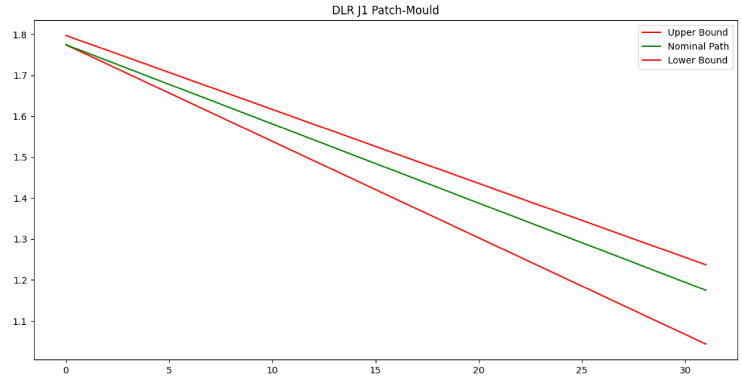


(b)

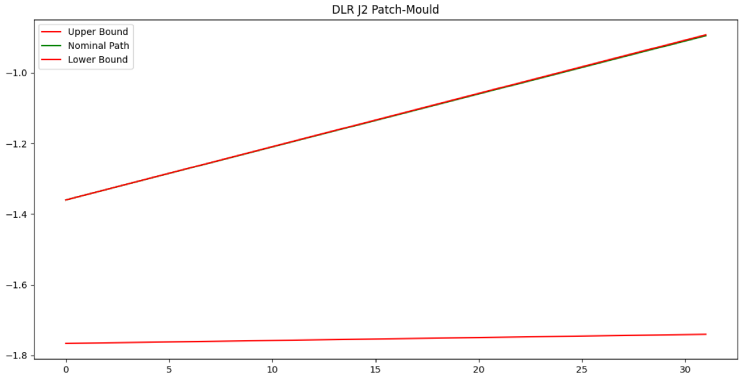


(c)

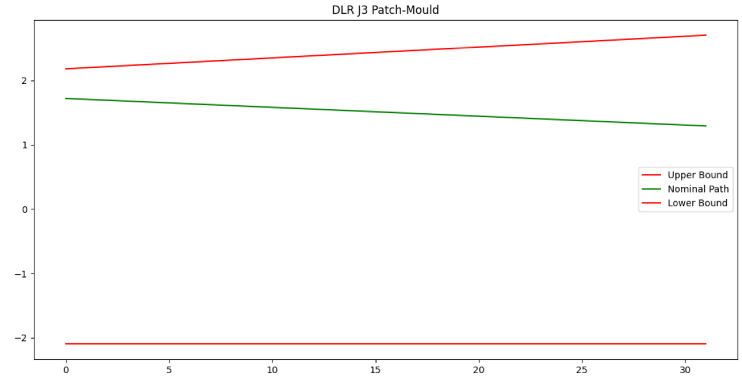
Figure 4.12: Possible boundaries through the *home-patch* path, use-case DLR; Joint1 (a), Joint2 (b) and Joint3 (c)



(a)

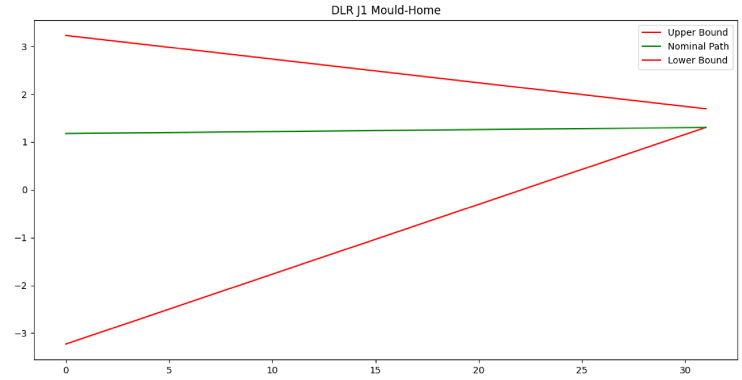


(b)

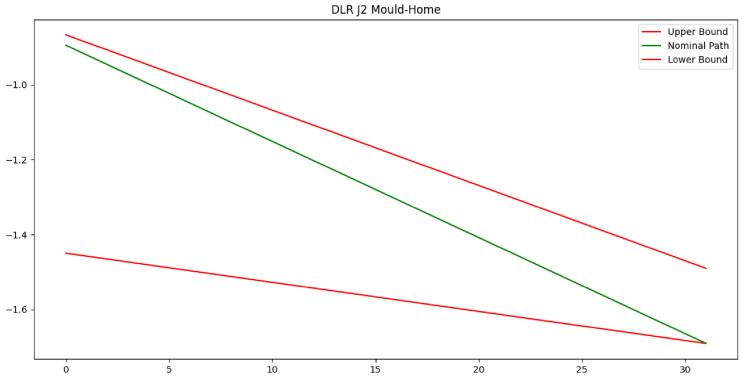


(c)

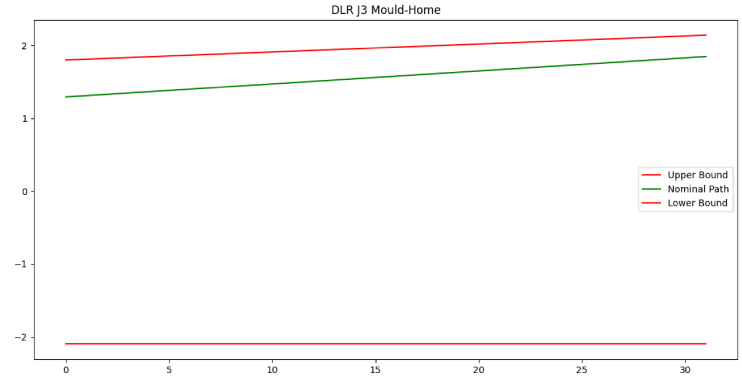
Figure 4.13: Possible boundaries through the *patch-mould* path, use-case DLR; Joint1 (a), Joint2 (b) and Joint3 (c)



(a)



(b)



(c)

Figure 4.14: Possible boundaries through the *mould-home* path, use-case DLR; Joint1 (a), Joint2 (b) and Joint3(c)

4.3 Discussion

The algorithm performs efficiently in the use cases presented. The worst CF obtained is 92.8516% in the *patch-mould* path of Dallara's use-case, while in the slightly modified version for DLR we touch a low of 86.7935%, although for very low discretization. As already stated, the *patch-mould* path appears to be the hardest one, as indeed it is the path closer to the highest amount of obstacles. Unexpectedly though, while still performing better than *patch-mould* at low discretization, the *home-patch* path has the lowest CF in DLR in the latter tests. This could be due to situational reasons, such as a slightly more challenging disposition of the obstacles and in particular a final pose more oriented toward the center of the patch table rather than the center of the mould. The proximity of the obstacles to the linear axis and thus to the whole robot system is likely the root cause of the performance difference from DLR to the other two use-cases, as the linear axis implies the robot is always pretty close to the obstacles. The trade-off is clearly that the robot can move fast and efficiently across the workcell, although this doesn't help for the specific objectives of this thesis work. Regarding the *patch-mould* path in Baltico and Dallara, a CF over 99% is still obtainable, as in both these use-case the obstacles are much more spaced out than in DLR. The *mould-home* path on the other hand is the easiest path, as the robot travels away from the obstacles. The algorithm achieves a remarkable $CF = 100\%$ already for very low discretizations of the height-map, with 216 total collision boxes for Baltico and just 89 for Dallara. On DLR at 100% in *mould-home* there are 961 instantiated boxes. More boxes are usually instantiated in DLR, as the workcell appears to be larger, and this appears to be the fundamental reason behind its increased computational times. Now that we've observed that the boundaries seem useful in terms of avoiding collision, let's indeed give a look at the computational times.

As it was theoretically evaluated in section 3.4, the algorithm behaves linearly with respect to the total number of collision boxes, as shown in figure 4.15. This is a good feature, in particular, confronted with the double exponential complexity of the CAD-based method as presented in section 2.3. The highest times recorded are:

- 7.89 s for Dallara, in *patch-mould*
- 8.01 s for Baltico, in *patch-mould*
- 13.64 s for DLR, in *patch-mould*

These values correspond to the highest discretization tested. Actually, we can reach a high CF consistently in around 2 seconds in all the use-cases. DLR appears to have a higher computational time compared to Dallara and Baltico, as the workcell is bigger, and thus more boxes are generally instantiated in order to compose the volume. This implies that, during the jointspace exploration step, the robot links are tested to collide with a larger amount of boxes, thus increasing the computational time as predicted by equation 3.3.

Having a linear complexity is a good feature, as theoretically we could meet any time specifics provided a powerful enough hardware. The whole computational time is composed of the time to generate the volume and the time to compute one boundary pair for each point in the nominal path. As there's no conflict between computing the upper boundary and the lower boundary, the two processes could be performed in parallel, although this hasn't been tested in the current work.
We begin to identify the close relationship between CF, computational time, and discretization.
Observing 4.16 we could take 15000 collision boxes and up as a safe parameter for the algorithm in the Dallara use-case, while approximately 18000 and up should be available for the Baltico use-case. In the latter, we can observe a slight instability in the performance, particularly in the *home-patch* path, which is the one with the highest overall proximity to the obstacles. Nonetheless, the CF stably stays above 99%. With regard to DLR, a discretization corresponding to 15000 collision boxes should suffice, in the limits of its performance compared to the previous use cases.
It could be argued, as the CF is always above 90% in any scenario, that an "experienced" user could be fine also with the low discretization of the height-map, obtaining an almost instantaneous computation of the boundaries. It is possible to draw some qualitative patterns in how the boundaries are formed, and this could also come intuitive to the human operator given enough practice.

Considering figures 4.4 to 4.10 we can notice how the first joint is usually kept in a tight range. Figure 4.6 could appear as an exception, but, actually, it's just that the initial and the final values for this joint are quite close and so the plot appears magnified. As the first joint commands what the robot is facing, a tight range implies that the final boundaries try to keep the robot aligned with the goal, and this is indeed the case for the home-mould path. The second joint's boundaries have the tendency to widen closer to the goal, allowing some freedom to deform the nominal trajectory. This enhances the adaptability of the robot next to the patch or to the mould, as we can expect those are the regions where the human operator would introduce the most significant deformation of the nominal trajectory. The only exception is in figure 4.10 where the boundary gets wide also at the home point. In the *home-patch* and *patch-mould* paths, the third joint's boundaries follow the nominal directions or stay approximately the same, as here the aim is that of not allowing the robot to crash towards the goal obstacle. On the other hand, when the path ends at the home point more freedom is allowed.
As we can observe from figures 4.12 to 4.14, the boundaries behave similarly in DLR, but we can notice, particularly for the first two joints, that they get even tighter than before when we get close to the obstacles, sometimes being almost equal to the nominal values. This is likely due to the proximity of the linear axis to the obstacles.
The evolution of the boundaries during the trajectory is just one possible way to use them, based on the fact that the joints' states in output from the global planner are linearly con-

nected. The algorithm does produce in the end one boundary for each point in the jointspace defined by the global planner. Subsequent boundaries could be interpolated through any mathematical interpolation method, not just linearly. The only advisable heuristic is to use the same interpolation decided for the nominal trajectory. In the end, the final choice in how to manage the boundaries belongs to the MPC design, while what was used here had the aim of being used to define a testing suite.

Considering the simulations presented in this chapter, the proposed algorithm appears to meet our requirements in terms of computational time and avoiding of collisions. It has the main advantage over the APF solutions presented in Chapter 2 of being computed offline and not incurring local minima-like problems, as well as directly addressing robot adaptability to the human user. In the next chapter, we will summarize what has been discussed so far and make some considerations for further developments.

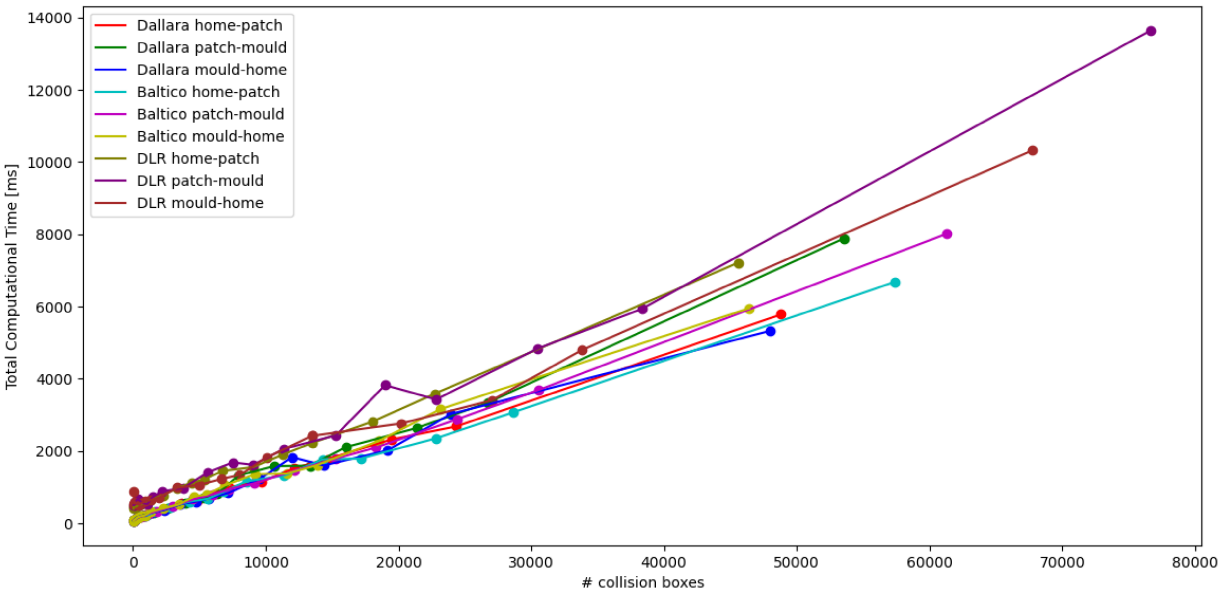
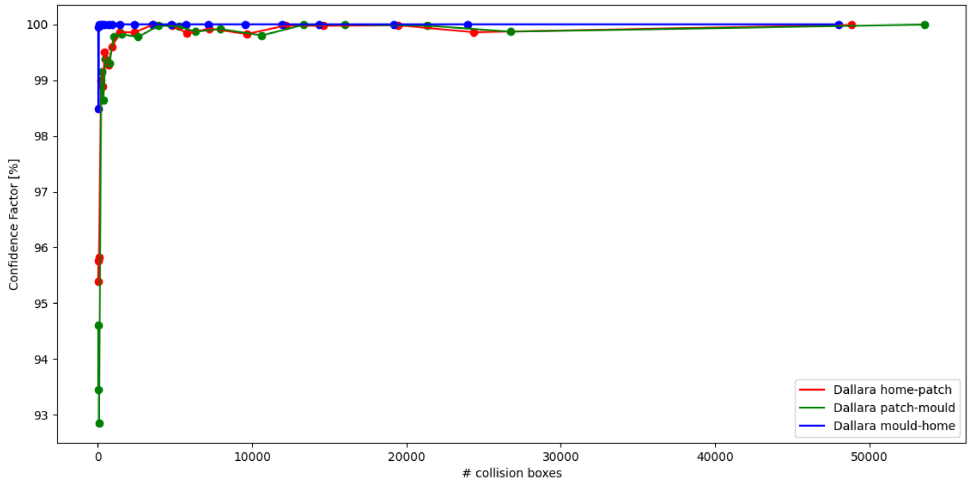
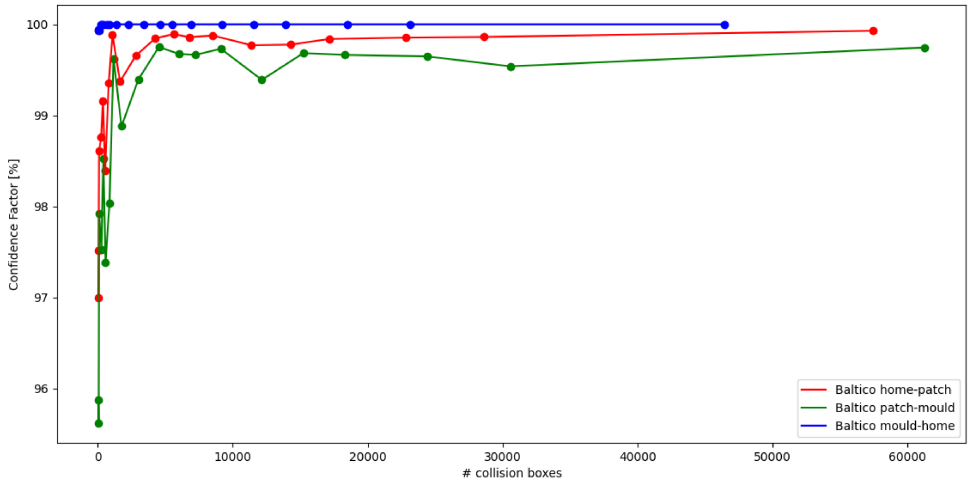


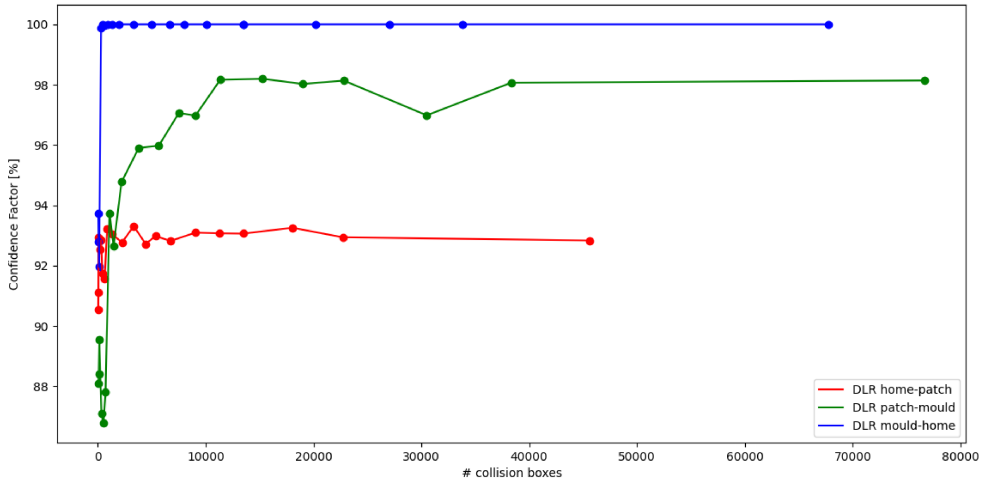
Figure 4.15: Plot of the total computational time for the various path tested



(a)



(b)



(c)

Figure 4.16: Confidence Factor vs Number of Collision Boxes for Dallara (a), Baltico (b) and DLR (c)

Chapter 5

Conclusion

We began our discourse by presenting in Chapter 1 the DrapeBot project, suggesting that we wanted to develop a way to improve the robot's adaptability to the human operator's presence. In Chapter 2 we started by analyzing some MPC based approaches to similar problems, noticing how those were at best oriented to the improvement of human safety rather than our interest. As well, APF didn't seem to provide a better solution considering its reactionary use could have produced uncontrollable results with the complex movements of a human being. Then we relied on the possibility of finding suitable subspaces of the jointspace, but the techniques already present in the literature seem to have performance issues, the main reason being they're typically used for very different purposes, such as analyzing a robot system rather than affecting its mobility.
In Chapter 3, we developed a solution suitable to our scenario, based on the idea of placing virtual obstacles around the nominal path and testing which joints' values give raise to collision with such obstacles in order to define boundaries in the jointspace. The solution is tested in the 3 use-cases provided by the DrapeBot project: Dallara, Baltico, DLR. We now ask ourselves how the solution could be improved and how it could be further generalized.

5.1 Future Works

In the volume generation, it was decided to do not to instantiate any volume outside the volume reach, as this couldn't influence robot motion. Is it possible to further decrease the amount of instantiated collision boxes? This would further speed up the algorithm. One simple approach could be to eliminate a given percentage of the boxes, choosing the boxes randomly. But what could be a reasonable percentage? Would it be significantly different use-case by use-case, path by path? This could be found with some simple optimization algorithm.
In DLR we assumed to bound the linear axis by itself and then repeat the same process of the previous use-cases to the other joints. What if we treated the linear axis as the other joints? Would the result be very different? This solution was discarded as it would have doubled the computational time, but it could still be viable.
In general, could the algorithm be used with n-DOF systems? At which point would it

start to perform badly, and would it just be issues related to the number of computations or would robot structures take relevance? We have reason to think it's possible to generalize the algorithm to n-DOF systems, in the use-cases tested we didn't find particular difficulties handling both revolute and prismatic joints. Also, nothing in the abstract version of the algorithm denies the possibility of adding more iterations to perform computations over an increased number of joints. There could be some computational complications if the number of joints becomes too high, but we could adopt analytical techniques such as the one performed in Chapter 3 (see Tables 3.1 to 3.1) in order to simplify joints relationship or discard certain joints which would appear to do not carry significant spatial information.
In order to answer most of these questions, the algorithm should be tested in new scenarios, implying different use-cases from the ones tested in this work. As this is not available for the current project, these questions cannot be empirically addressed. From the theoretical point of view, each time we increase the DOF we double the amount of computation, so an upper limit certainly exists. On the other hand, most industrial applications do not require more than 6 or 7 degrees of freedom, thus we feel confident that the algorithm as described and tested so far can be adopted in many different scenarios.

Bibliography

[1] *Drapebot webpage*, https://www.drapebot.eu/.

[2] *Human-robot collaboration on draping of composite parts*, https://cordis.europa.eu/project/id/101006732/it.

[3] S. Das, J. Warren, D. West, and S. M. Schexnayder, "Global carbon fiber composites supply chain competitiveness analysis," *CEMAC technical report*, 2016.

[4] B. Siciliano and O. Khatib, "The handbook of robotics," *Springer-Verlag*, 2008.

[5] M. Morari and J. Lee, "Model predictive control: Past, present and future," *Computational Chemical Engineering*, 1999.

[6] D. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, 2014.

[7] E. Camacho and C. Alba, "Model predictive control," *Springer*, 2013.

[8] B. Siciliano, L. Scivacco, L. Villani, and G. Oriolo, "Robotics: Modelling, planning and control," *Springer*, 2009.

[9] G. Incremona, A. Ferrara, and L. Magni, "Mpc for robot manipulators with integral sliding modes generation," *IEEE*, 2017.

[10] M. Faroni, M. Beschi, and N. Pedrocchi, "An mpc framework for online motion planning in human-robot collaborative tasks," *IEEE*, 2019.

[11] M. Eckhoff, R. Kirschner, E. Kern, S. Abdolshah, and S. Haddadin, "An mpc framework for planning safe trustworthy robot motions," *IEEE International Conference on Robotics and Automation*, 2022.

[12] S. Gai, R. Sun, S. Chen, and S. Ji, "6-dof robotic obstacle avoidance path planning based on artificial potential field method," *IEEE*, 2019.

[13] S. Park, M. Lee, and J. Kim, "Trajectory planning with collision avoidance for redundant robots using jacobian and artificial potential field-based real-time inverse kinematics," *International Journal of Control Automation and Systems*, 2020.

[14] J. Zhao, Y. Wang, L. Jin, X. Duan, and W. Zhang, "Obstacle avoidance strategy of improved apf method in c-space," *Proceedings of the 2021 IEEE International Conference on Robotics and Biomimetics*, 2021.

[15] S. Dalai, M. Irfan, S. Singh, K. Kishore, and D. Akbar, "Heuristic guided artificial potential field for avoidance of small obstacles," *International Conference on Control Automation and Systems*, 2021.

[16] C. Wang, L. Meng, T. Li, C. DeSilva, and M. Meng, "Towards autonomous exploration with information potential field in 3d environments," *International Conference on Advanced Robotics*, 2017.

[17] Z. Elmi and M. Onder, "Path planning using model predictive controller based on potential field for autonomous vehicles," *IEEE*, 2018.

[18] H. Shena, Y. Zhao, J. Li, G. Wuc, and D. Chablat, "A novel partially-decoupled translational parallel manipulator with symbolic kinematics, singularity identification and workspace determination," *Elsevier*, 2021.

[19] X. Yang, H. Wang, C. Zhang, and K. Chen, "A method for mapping the boundaries of collision-free reachable workspaces," *Elsevier*, 2010.

[20] R. Jhaa, D. Chablat, L. Baronc, F. Rouillier, and G. Moroz, "Workspace, joint space and singularities of a family of delta-like robot," *Elsevier*, 2018.

[21] J. Kuffner and S. LaValle, "Rrt-connect : An efficient approach to single-query path planning," *Proceedings of the 2000 IEEE International Conference on Robotics Automation*, 2000.

[22] S. Glottschalk, "A framework for fast and accurate collision detection for haptic interaction," *IEEE*, 1999.

[23] H. Samet, "The quadtree and related hierarchical data structures," *Computing Surveys*, 1984.

[24] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx," *ICRA*, 2015.