Università degli Studi di Padova

Department of Information Engineering

Master Thesis in Automation Engineering

Automatic offline trajectory generation for surface coverage problem from a 3D drawing

Svpervisor
Prof. Ruggero Carli
Università di Padova

Co-svpervisor
Ing. Roberto Polesel
Euclid Labs S.r.l.

Master Candidate
Luca Battistella

To my parents, without them none of this would have been possible.

Abstract

Surface coverage with a robot manipulator is widely used in different industrial processes. For this reason having an automated way to program these robots is really important.

In this work we implement an algorithm to automate the process of trajectory generation for surface coverage, given the 3D model of the target object, with a particular focus on spray painting process. As first step we divide complex surfaces into simpler patches and the algorithm successively generates the trajectory for each of them. Then we develop a tool model for a spray gun to have the ability to simulate paint thickness on target object after trajectory generation. This way we can improve paint thickness by optimizing end effector velocity for each paint pass. Once we generate all the trajectories, we check if the robot can perform them, by doing collision checking and verifying that the robot can reach all the points composing our trajectories Finally we test the generated program on a real robot, verifying also for the quality of generated trajectories. Experimental results reveal that the trajectories generated by this algorithm and the robot code allows the robot to perform the painting process on a test surface in a smooth and efficient way.

Contents

Listing of figures

1 Introduction

Coverage problem with a robot manipulator consists of moving a robot manipulator such that its tool can cover all the surface of a target object.

This topic has been researched for decades for industrial applications [1], with a particular focus on the spray painting process. The solution to this problem can be used for several industrial applications, like for cleaning train Cabs [2] (or other surfaces), grinding processes, for surface coverage for the detection of imperfections. Its main application, however, is in robotic spray painting, where there is a huge use in automotive and facilities manufacturing. Using a robot manipulator to perform these tasks, instead of a human worker, has some advantages such as an increased quality and efficiency, larger working range, and removal of workers from an environment that could be harmful to health (e.g. a spray painting cell).

For this reason, there are a lot of research papers in literature proposing different approaches. Focusing on spray painting, it is possible to distinguish two main approaches: trajectory generation from acquired data (both from point cloud and from 3D drawing) or visual servoing. Visual servoing approach is interesting for several application like surface imperfection detection or model acquisition [3] but it is not feasible to use as online painting method [4] due to the difficulty on keeping the camera cleaned from paint. In fact in a painting cell there is a lot of paint particles in the air which could dirt the camera, making visual servoing unusable. This is the main reason why, in this work, we use the approach where trajectory generation is performed from acquired data.

There are many different options on which data can be chosen to describe the target object. In some papers trajectories are generated directly from point clouds, acquired by a scanner, with Point Cloud Slicing approach [5], but in this work we choose to use a mesh approximation to generate the trajectories, because a mesh object can be created by a point cloud [6] and because usually the 3D drawing of the object to be processed exists and could be more accurate than a model generated by a point cloud or the point cloud itself.

Since the objects to be painted could be very complex, we choose to split them in simpler surfaces. There are several works concerning patch segmentation, based on different approaches like the use of semi-supervised mesh segmentation [7], the use of a hierarchical region decomposition algorithm based on quadric surface fitting [8], or the use of curvature tensors [9]. However, in this work, we adopt a simple mesh decomposition algorithm based on the normal of the triangles, since patch segmentation is not the purpose of this work and the algorithm adopted provides good results.

For trajectory generation we develop a spray painting tool model (Sec.4.1.2) to generate and optimize the trajectories. In fact, based on the model tool, we can simulate the resulting paint thickness (Sec.4.2.4) and optimize the trajectories velocities to achieve the best paint result (Sec.4.2.5). The algorithm implemented in this work is based on the iterative algorithm of [10] but it has been modified to achieve slightly better performances. Since the goal of this work is to develop an automated robot programming process, we built a Windows program which generates the robot code to paint the given 3D model (Sec.5.1), and we test it in a real industrial robot manipulator (Sec.5.2).

I like solving problems, and science provides a logical way of solving real-life problems.

David J. Anderson

2 Problem Formulation

This work is focused on the industrial application of surface coverage problem with a robot manipulator, to find a solution that allows to automate the process of robot trajectory generation.

The surface coverage problem with a robot manipulator consists on finding a trajectory such that the robot tool can cover the entire surface, avoiding to leave empty space on it. This process can be done using different sources for the target surface, such as point cloud or 3D drawing, and this work is focused on surface covering from a 3D drawing. Therefore, given a 3D model of the object, all trajectories for the robot manipulator need to be generated to perform the required task. In industry, as every task requires its own processing, the robot trajectories could be really different between each different process. In fact surface coverage is a general problem that could have different industrial applications, such as surface cleaning, smoothing, painting with a spray gun, and in each application the robot has to do a different trajectory to perform the task in a proper way. For example in a cleaning process it does not matter if the tool passes above the same part of surface twice, but in a painting process this fact leads to an increment of the paint thickness on that part, causing a dissatisfying final result. A similar consideration can be done for grinding process, in fact passing twice the same part cause an over ground surface.

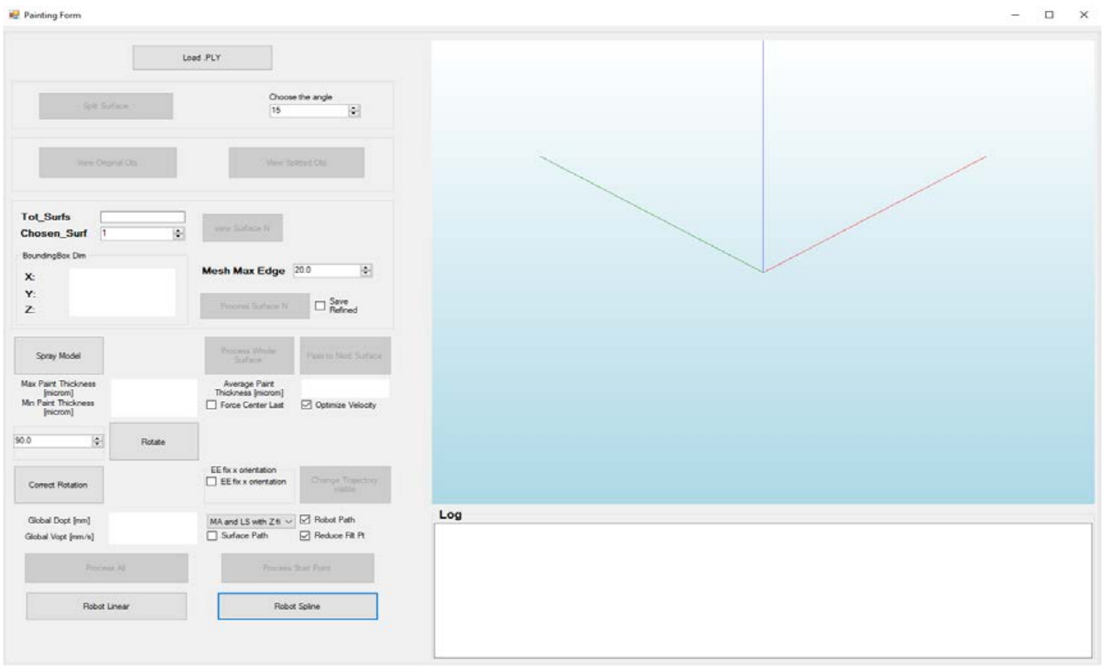
In this work, the goal is to develop a general algorithm for surface covering that, with the right tool model, can solve the problem for different processes. In particular, we focus on the spray painting process. For this reason we develop the spray tool model and the algorithm relying on the work of *V. Andulkar, Mayur and Chiddarwar, Shital* [10], which allow us to simulate and optimize trajectories and painting result on the surface. Furthermore this algorithm could face a general surface coverage problem, by changing the tool model and some tweaks made ad-hoc for the painting process.

During implementation we modify the original work to improve the results and trajectories that this algorithm generates. Since surfaces can be very different from each other, it is difficult to define a general set of parameters for the algorithm that work well for every type of surface.

For this reason, we left the ability to change the default parameters to the user to have better results based on the surface to be painted (e.g. the possibility to change the filtering method). We set as default parameters the ones that we found to work better with most of surfaces. The improvements that we find necessary are:

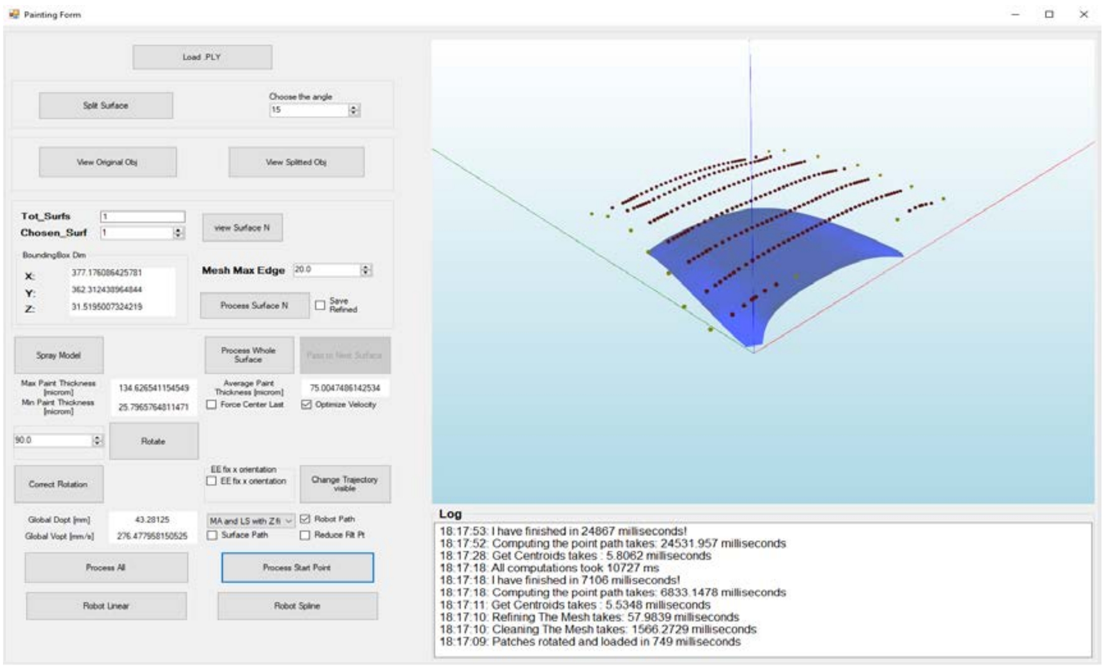
- we adopt a new method to align the object with the origin of axis;
- we divide a complex surface into simpler patches to make the algorithm working better, and, when a trajectory for each simpler patch is generated, we reassemble the original surface to have the final trajectory.
- we implement the variable x_{incr} method described in [10] but we do not use it because, thanks to surface decomposition, it gives no significant improvements to trajectory generation, while adding a lot of computational effort.
- to filter the generated trajectories we use a Least Square filter, instead of just a Moving Average filtering, in order to have better trajectories and the possibility to reduce the number of points in a trajectory leading to better result when we program the real robot.

Finally we built a Windows program (Fig. 2.0) which allows the user to generate trajectories based on the surface model and surface tool. Surface model can be loaded to the project (Fig. 2.1a) and the tool model can be edited to match with the real one (Fig. 2.0c). Once the surface and the tool are properly setted up, the algorithm can be run to generate the trajectories and simulate the resulting paint thickness given by the generated trajectory (Fig. 2.1b). The user has also the ability to change algorithm parameters to fit better the selected surface, like the filtering type or the main paint direction, by rotating the object. Once satisfactory trajectory is generated the painted object can be added to robot world together with all collision objects which are present also in the real world (Fig. 2.0d). Here robot trajectories are simulated, with its planning and collisions checking. Once trajectory planning is successful, meaning that there are no collisions and all points are reachable, the robot program is created, and after it is loaded on the real robot, the process can be performed.



Choose the angle: 15
Tot_Surfs:
Chosen_Surf: 1
BoundingBox Dim
X:
Y:
Z:
Mesh Max Edge: 20.0
☐ Save Refined
Max Paint Thickness [microm]
Min Paint Thickness [microm]
Average Paint Thickness [microm]
☐ Force Center Last
☑ Optimize Velocity
90.0
EE fix x orientation
☐ EE fix x orientation
Global Dopt [mm]
Global Vopt [mm/s]
MA and LS with Z fi
☑ Robot Path
☐ Surface Path
☑ Reduce Fit Pt

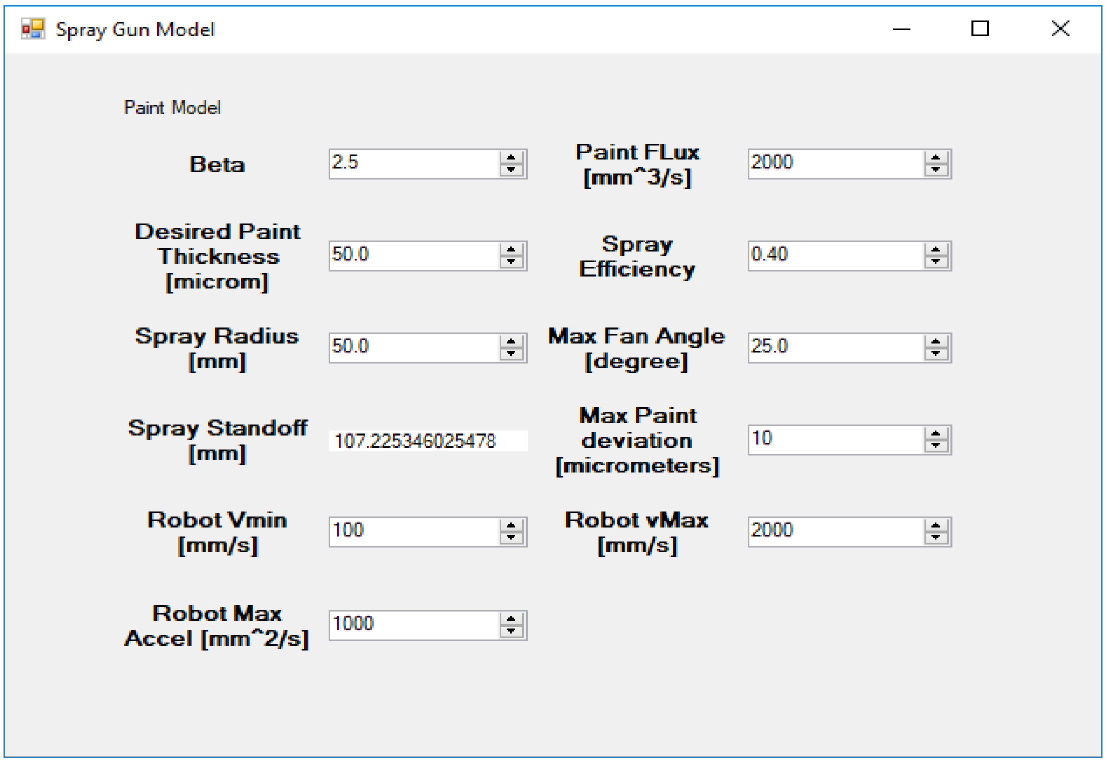
(a) Main window.



Choose the angle: 15
Tot_Surfs: 1
Chosen_Surf: 1
BoundingBox Dim
X: 377.176086425781
Y: 362.312438964844
Z: 31.5195007324219
Mesh Max Edge: 20.0
☐ Save Refined
Max Paint Thickness [microm]: 134.626541154549
Min Paint Thickness [microm]: 25.7965764811471
Average Paint Thickness [microm]: 75.0047486142534
☐ Force Center Last
☑ Optimize Velocity
90.0
EE fix x orientation
☐ EE fix x orientation
Global Dopt [mm]: 43.28125
Global Vopt [mm/s]: 276.477958150525
MA and LS with Z fi
☑ Robot Path
☐ Surface Path
☐ Reduce Fit Pt

```
18:17:53: I have finished in 24867 milliseconds!
18:17:52: Computing the point path takes: 24531.957 milliseconds
18:17:28: Get Centroids takes : 5.8062 milliseconds
18:17:18: All computations took 10727 ms
18:17:18: I have finished in 7106 milliseconds!
18:17:18: Computing the point path takes: 6833.1478 milliseconds
18:17:11: Get Centroids takes : 5.5348 milliseconds
18:17:10: Refining The Mesh takes: 57.9839 milliseconds
18:17:10: Cleaning The Mesh takes: 1566.2729 milliseconds
18:17:09: Patches rotated and loaded in 749 milliseconds
```

(b) Generated trajectory.

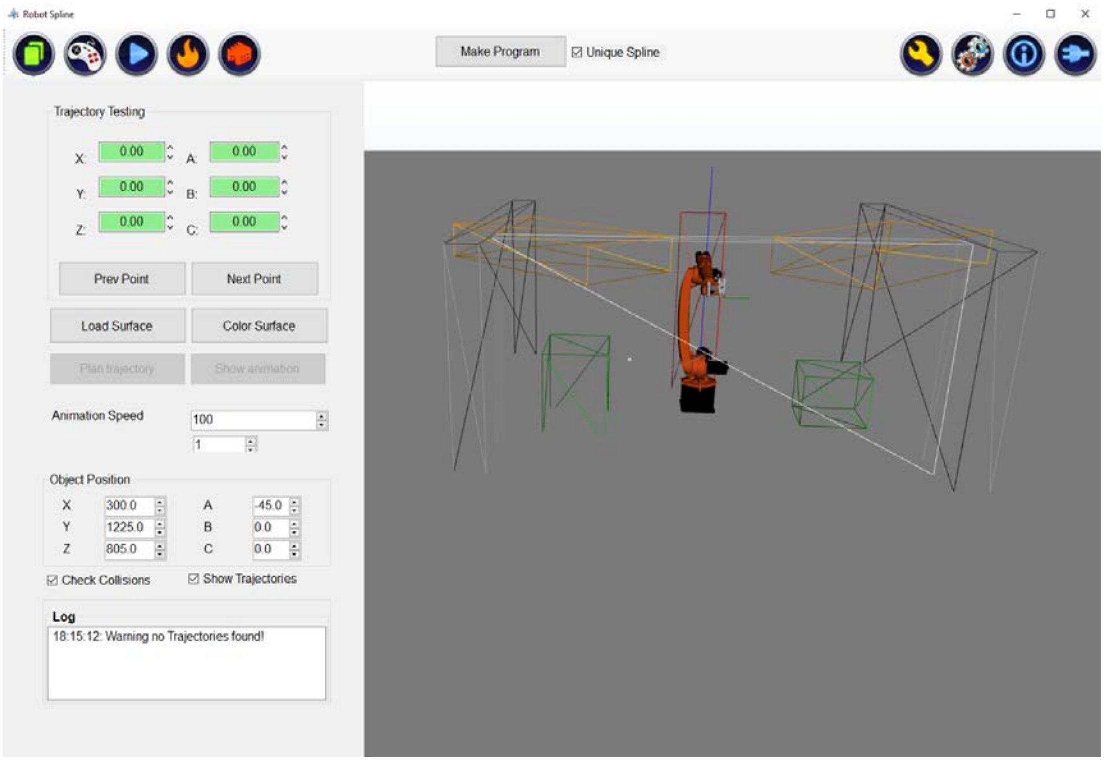


Spray Gun Model

Paint Model

Beta	2.5	Paint FLux [mm^3/s]	2000
Desired Paint Thickness [microm]	50.0	Spray Efficiency	0.40
Spray Radius [mm]	50.0	Max Fan Angle [degree]	25.0
Spray Standoff [mm]	107.225346025478	Max Paint deviation [micrometers]	10
Robot Vmin [mm/s]	100	Robot vMax [mm/s]	2000
Robot Max Accel [mm^2/s]	1000		

(c) Tool model window.



Trajectory Testing

X:	0.00	A:	0.00
Y:	0.00	B:	0.00
Z:	0.00	C:	0.00

Animation Speed 100
1

Object Position

X	300.0	A	-45.0
Y	1225.0	B	0.0
Z	805.0	C	0.0

☑ Check Collisions ☑ Show Trajectories

Log

```
18:15:12: Warning no Trajectories found!
```

(d) Robot world.

Figure 2.0: Different windows of developed program

I am among those who think that science has great beauty.

Marie Curie

3 General Algorithms used in this work

In this chapter we present all the algorithms we used in this work. In particular, we describe what is a mesh object which we use to represents the object to paint and why it needs to be refined. Afterwards, we describe the algorithm used to perform patch segmentation and two derivative-free optimization algorithms that we use for trajectory generation: Pattern Search Algorithm and an Adapted Genetic Algorithm. Finally, we present Least Square used to filter trajectories.

3.1 Mesh cleaning and refinement

To ensure that the painting algorithm performs well, the mesh approximation of the object needs to be refined and cleaned. Cleaning is necessary just for computational reason because mesh generation could create more than one point representing the same vertex of a triangle. Mesh refinement, instead, is indispensable, since it is strongly related to the algorithm performances and results, because if the mesh is not sufficiently refined could lead to unwanted behaviours of the algorithm, (e.g. inaccurate trajectories or bad paint thickness simulation with empty triangles that should be painted).

Refining a mesh means that, once we fix a maximum length l_{max}, all the edges of the triangles need to be smaller or equal to l_{max}. The choice of l_{max} is critical, in fact a bigger value leads to a faster algorithm but a less accurate paint thickness simulation and possible undesired trajectories, while a lower value leads to bigger accuracy on paint thickness simulation but makes the algorithm run slower. From experimental result we found that $l_{max} \leq 0.8R$ is a good value. In fact we have observed that in this range all the trajectories generated by the algorithm are very similar to each other and the only difference is on the precision of paint thickness simulation. For this reason, the value of l_{max} is related to the application of this algorithm. In fact, for a real time application it is preferable to choose a value close to the upper limit to make the algorithm faster, while for an offline trajectory generation we can choose a value which gives the desired precision on paint simulation and gives more accuracy on tool velocity optimization, which uses the paint thickness simulation to find the optimal velocity.

3.2 Patch segmentation

Generating trajectories for covering a complex surface could be very difficult and could lead to not optimal result. The simplest and more effective solution is to divide a complex surface in simpler patches, applying the painting algorithm to each patch and then recomposing the whole surface. In this work the segmentation criteria is based on mesh triangles normals. Chosen a maximum angle θ_{max}, the triangles are grouped based on angular difference between their normals. Specifically if two triangles have the angle between their normals smaller or equal than θ_{max} then they belong to the same patch, while if it is bigger than θ_{max} they belong to different patches. Moreover the subdivision in simpler patches allow us to suppose that there are no highly curved surfaces to paint, but just smooth ones.

This method could be inaccurate in particularly complex surfaces with edges that are not well defined, but, considering the industrial application of this work, we suppose to have all the surfaces with well remarked edges, where this method works well. Other patch segmentation methods, such as [7] and [8], can be adopted, which could lead to patch segmentation that better suits the task that has to be done.

3.3 Pattern Search Algorithm

Pattern Search Algorithm (PSA) is a derivative free optimization method. In this work it is used when a minimization problem with an objective function that has unknown derivative is faced. Since we use PSA only with two dimensional functions, the algorithm is described only in this case but can be easily extended to the n-dimensional case by changing the pattern.

Given a generic objective function $y = f(x)$, the first step is to find a pattern to evaluate the function. In two dimensional case this pattern is the following: given a point x_0, the other points are chosen as $x_M = x_0 + k$ and $x_m = x_0 - k$ where k is an hyper-parameter that needs to be tuned. This parameter affects the performances of the algorithm and its ability to explore all the function without getting stuck in a local minima. In fact if k is too small there is the risk that the algorithm gets stuck in a local minimal, while if k is too high there is the risk that it skips a minimal if two maxima are closer than k.

Starting from the initial value x_0, this algorithm evaluates the value of $f(x)$ in the points x_0, x_m and x_M resulting, respectively, in y_0, y_m and y_M. Then it selects $\min_x\{y_0, y_m, y_M\}$ as the new x_0. If the new x_0 is the same point as the previous step then it reduces the pattern by a factor d_f, then $k = k/d_f$, otherwise it continues to iterate. The algorithm runs until $k > k_{end}$. Then, after initial point x_0, parameter k, decimator factor d_f and termination condition k_{end} are chosen, the algorithm works as described in Algorithm 3.1.

Algorithm 3.1 Pattern Search

```
1: procedure $y$ = PATTERNSEARCH($f(x)$, $x_0$, $k$, $k_{end}$, $d_f$)
2:     while $k > k_{end}$
3:         $x_M = x_0 + k$
4:         $x_m = x_0 - k$
5:         $y_0 = f(x_0)$
6:         $y_M = f(x_M)$
7:         $y_m = f(x_m)$
8:         if $y_M < y_0$                ▷ This evaluate the $x$ in the pattern that minimizes $f$
9:             if $y_m < y_M$
10:                $x_{0,new} \leftarrow x_m$
11:            else
12:                $x_{0,new} \leftarrow x_M$
13:        else if $y_m < y_0$
14:            $x_{0,new} \leftarrow x_m$
15:        else
16:            $x_{0,new} = x_0$
17:        if $x_0 = x_{0,new}$          ▷ If the minimum is the same as before reduce the pattern
18:            $k \leftarrow k/d_f$
19:        $x_0 \leftarrow x_{0,new}$
20:    return $x_0$                      ▷ Return the $x_0$ that minimizes $f(x)$
```

3.4 Adapted Genetic Algorithm for velocity computation (AGA)

The Genetic Algorithm (GA) is a search heuristic algorithm that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

Each individual of the population is made of genes, that are a set of parameters (variables), that jointly represents a chromosome. Each chromosome describes a different solution, in this case a different velocity v. In each population there are different individuals, each one representing a solution. Here a chromosome is a representation of the velocity v in base 2, so each gene belongs to the set of parameters $\{0, 1\}$. The length of a chromosome is the maximum velocity that the robot could reach, described in base 2. Moreover we need to convert from base 2 to base 10 the velocity v in both sides: from base 2 to base 10 to compute fitness function and to get the final velocity, from base 10 to base 2 to run the GA.

In order to find the best speed for each paint pass, the genetic algorithm here presented focus on velocities, maximizing a function cost $f(v)$, where v is the velocity. [11]

The Algorithm main steps are summarized in Algorithm 3.2 and described above:

3.4.1 Initial Population

The algorithm is initialized by creating all individuals with random chromosomes.

3.4.2 Computing Fitness Function

Given the fitness function $f(v)$, for each individual i of the current population, the fitness function based on its chromosome is computed. In this way each individual of the current population has its fitness function $f(v_i)$.

3.4.3 New Population

Once the fitness function for each individual is computed, if the termination condition 3.4.5 is not reached, a new population of the same size of the current one is created. Individuals in current population are called parents, the ones in new population are called childes.

There are different ways to choose the individuals that compose the new population. In this work a part of individuals, 10% of population size, are chosen by elitism, which means that best individuals of the current population are added in the new one as they are. Then the remaining individuals of the new population are generated by crossover (Sec. 3.4.4) from two parents, which are chosen from the best half of parents.

It is possible to choose different values for the elitism percentile but by trials we found that 10% is a good choice and leads to a fast convergence in most cases.

3.4.4 Crossover

Crossover is the procedure to choose each new gene of the child from genes of the parents, and can be done in different ways. In this work is done by choosing a random probability p_1 for each gene, and if $p_1 > 0.5$ then i-th gene is taken from the second parent, otherwise from the first one.

After a child is generated, a mutation can occur with another random probability p_2: if p_2 is lower than mutation rate, the gene is generated randomly otherwise the one selected by crossover is kept. Here we choose a mutation rate equals to 0.01 which is a typical value used in Genetic Algorithm.

3.4.5 Termination Condition

There are different kind of termination conditions for Genetic Algorithm. In this work, we choose that GA terminates when the difference between best fitness function of previous population and best fitness function of current population is smaller than a given threshold or the number of generations exceeds a given maximum g_{max}. Here as threshold we choose 10^{-6} because if the difference between the two fitness function is below this value means that genetic algorithm has found the minima and it is not chosen 0 to get rid of approximation errors, while we choose $g_{max} = 20$ because we see that if the algorithm converges it will do in the first 10 generations. Moreover, we add the fact that if the fitness function is below 1 the algorithm keeps to iterate, this is because a fitness function lower than 1 means that the result is not so good (but could be the best possible) and we want that the algorithm keeps to search for a better result. This last aspect of fitness function is the reason why the algorithm could not converge in less than g_{max} populations, in fact when the algorithm terminates due to exceeding the limit on population number, the best fitness function has always a value under 1, which, in that case, is the best possible value.

Algorithm 3.2 Genetic Algorithm

```
1: procedure $v$ = GeneticAlgorithm(PopSize,ChromoSize,MutationRate)
2:     Initial Population
3:     Compute Fitness Function
4:     Save Best fitness
5:     repeat
6:         Generate New Population
7:         Compute Fitness Function
8:         Save Best fitness
9:     until {TerminationCondition(prevFitness, CurrentFitness) = true}
10:    return $v$                              ▷ Return the $v$ of the best individual
```

3.5 Least Square Fitting

Least Square method is used in linear regression to fit a linear function, given a model set of functions. In this work we use it to fit the function $f(x)$ that belongs to the model set of n-th degree polynomials, with $n \geq 4$.

The Least Square method is used to find the parameters of the unknown objective function $f(x)$ that better represents data points, in sense that it is the function that minimizes the sum of squared residuals (a residual being: the difference between an observed value, and the fitted value provided by a model). Once the function $\hat{f}(x)$ is estimated the data points are projected onto it.

Calling β the vector of the unknown parameters of $f(x)$, an estimator for β is given by:

$$\hat{\beta} = (X^T X)^{-1} X^T y \tag{3.1}$$

where X is a matrix whose ij element is the i-th observation of the j-th independent variable and y is a vector whose i-th element is the i-th observation of the dependent variable. In this work, x coordinate of the trajectory point is the independent variable, while the z or y coordinate is the dependent variable. In fact, we use this algorithm to filter both y and z coordinate.

Since the data to be fitted are close to the real model, the least square method is good to give a description of the trajectory in both z and y direction and to correct the approximation introduced by numerical errors and the mesh.

Moreover, after we fit the model $\hat{f}(x)$, with least square filtering we can generate as much point as desired, describing the trajectory with less points without losing precision. Then we sample the points at a desired step, keeping the start and stop paint, to have a description of the trajectory which the robot can perform without having too much or too close points, resulting in a more accurate trajectory following by the robot.

The science of today is the technology of tomorrow.

Edward Teller

4 Robot trajectory generator algorithm

In this chapter we describe the main algorithm that we use to generate the trajectories. Firstly we presents all requirements needed to run the main algorithm, such as a mesh object and spray gun model, with computation of the optimal velocity and optimal overlap in a planar surface. Then we describe the trajectory generator algorithm where we simulate final paint thickness and optimize each paint pass, iterating it until all surface is covered.

4.1 Requirements

Here all requirements necessary to run the main algorithm are described:

- the mesh object which is how we describe the data
- the model tool we use to describe the process and to simulate the paint results
- the algorithm we use to align the object so that we can run the trajectory generator algorithm

4.1.1 Mesh object

The object to be painted is given by a 3D model, so we approximate it with a mesh object. This is a polygon approximation of the real object and we choose it because a free-form surface can be easier represented by a polygon approximation rather than by parametric representation. In this work we adopt a triangular mesh, as the example shows in Fig. 4.1

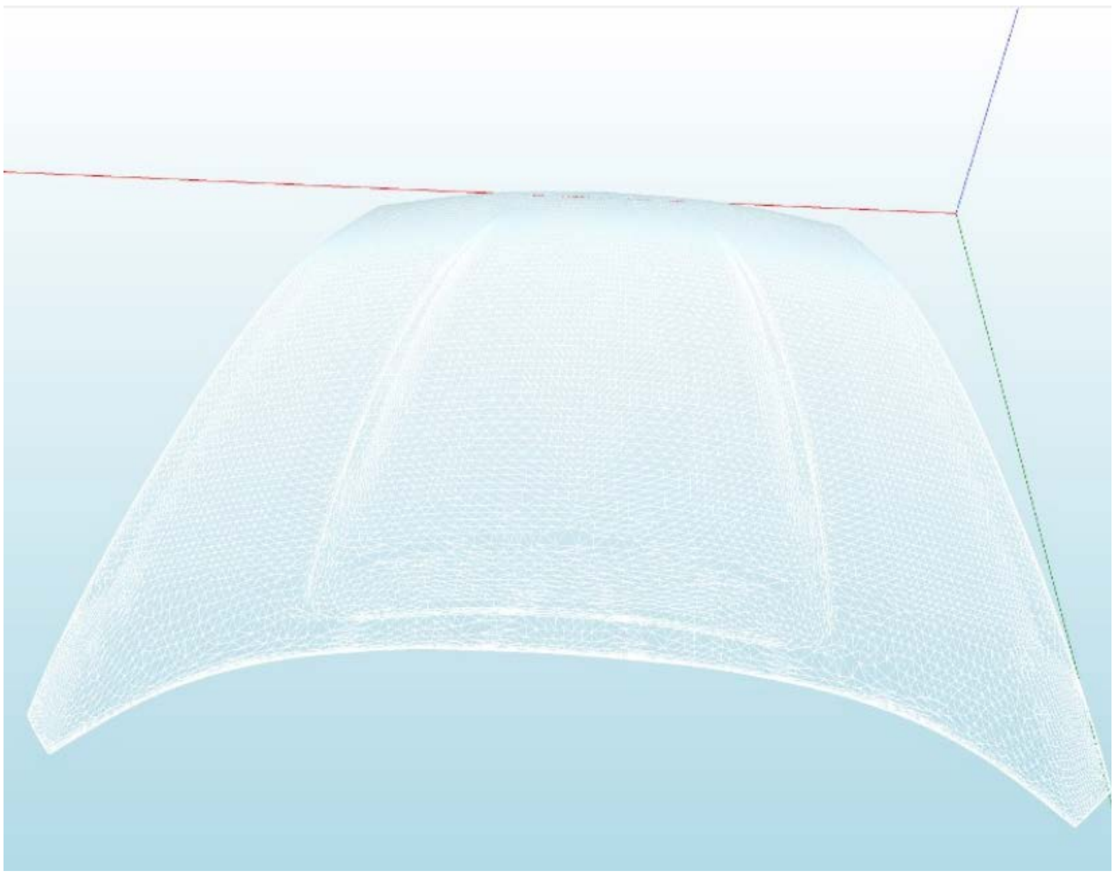


Figure 4.1: Example of triangular mesh object approximation

Before running the algorithm, the model needs to be preprocessed. Preprocessing consists of: refining the mesh as described in Sec.3.1, and splitting it in simpler surfaces as described in Sec.3.2.

The main advantage is that the mesh object can be generated both from a CAD model of the object, both from a Point Cloud ([6]) acquired in Real-Time with a camera or a scanner.

4.1.2 Spray gun and paint distribution model

Tool model is one of the most important part of our algorithm because allow us to have a description of resulting paint thickness. Since the main objective of the painting process is to have a desired thickness of the paint q_d, as uniform as possible, we need a good model for the spray tool, in order to optimize that. In this model, the spray tool has a spray pattern that can be modelled by a cone as shown in Fig.4.2.

Paint particles are emitted from the tool radially within the spray cone with a fan angle of θ. The tool is supposed to work at a distance h from the surface (tool stand-off), so the maximum radius of the cone can be determined as $R = h$ tan(θ), which is defined as spray radius. The tool spray direction is parallel to the normal of the surface. When the spray cone intersects a surface, it forms a spray pattern. Another aspect we need to take into account is that material deposition rate depends on many parameters, like paint flow of the spray gun, the efficiency (defined as how much paint of paint flow deposits on the surface) or the gun stand-off. In this work, with a reasonable approximation, we consider all these parameters to be constant. Then we can define paint deposition rate as function of only distance between a point inside the spray cone and the center, that we call $r = h$ tan(γ).

To model paint deposition rate we use the Beta paint distribution model, which provides flexibility in selecting β based on the characteristic of the spray gun, and also the possibility to obtain different distributions such as elliptical ($\beta = 1.5$), parabolic ($\beta = 2$), and Gaussian ($\beta \geq 2.5$) as shown in Fig.4.3a. Different paint distributions can be also obtained varying the spray radius, instead of β, as it is shown in Fig.4.3b.

Using Beta paint distribution (in [10]), the model paint deposition rate is given by:

$$\dot{q}(r) = \frac{\eta Q_0 \beta}{\pi R^2} \left(1 - \frac{r^2}{R^2}\right)^{\beta - 1} \tag{4.1}$$

where $\dot{q}$ is the rate of film accumulation at an arbitrary surface point $S = (x, y, z)$ at a distance r from the center, Q_0 is paint flow of the spray gun, η is the efficiency and R is the spray radius.

Paint Thickness computation

As the spray gun moves along the surface, paint is deposited over the surface. Resultant paint thickness can be obtained by integrating the paint deposition rate $\dot{q}(r)$ for the time of travel. The spray gun velocity is considered constant, an aspect that is also supported by automotive spray painting specialist [10]. When the spray gun moves along x axis with a constant velocity v, the paint thickness at any point L can be obtained as:

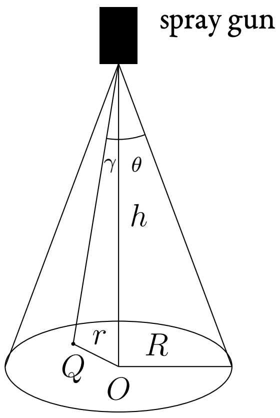
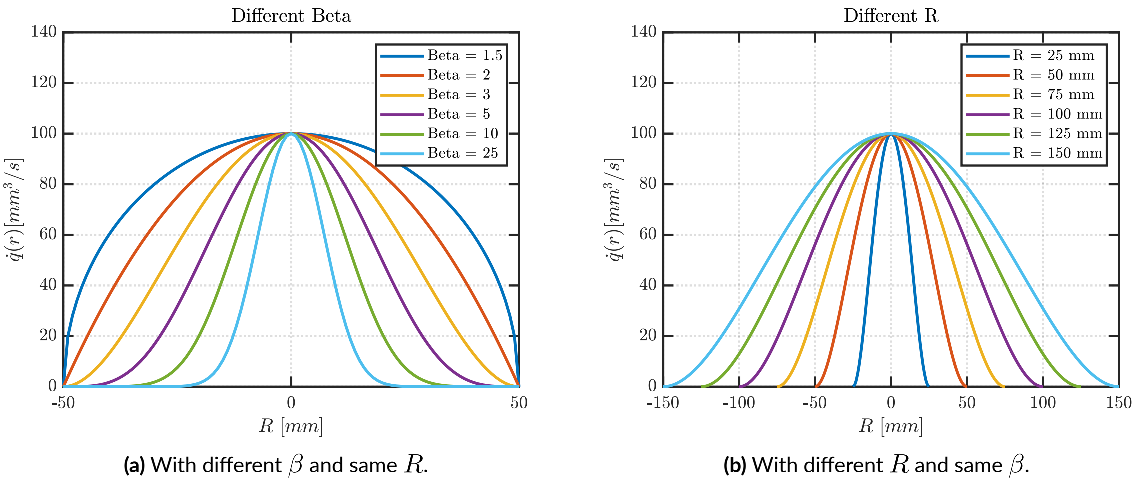


Figure 4.2: The model of the spray tool



(a) With different β and same R.

(b) With different R and same β.

Figure 4.3: Different values of Paint deposition rate changing the parameters

$$q(y,v) = 2\int_0^t \dot{q}(r)dt \quad 0 \le r \le R\,, \quad \text{with } \dot{q}(r) = \frac{\eta Q_0 \beta}{\pi R^2}\left(1 - \frac{r^2}{R^2}\right)^{\beta-1} \tag{4.2}$$

where t is the spray time on L and y is the distance of L from the tool center projected, as it is shown in Fig.4.4. Then t and r are given by:

$$t = \sqrt{R^2 - y^2}/v, \qquad r = \sqrt{(vt)^2 + y^2} \tag{4.3}$$

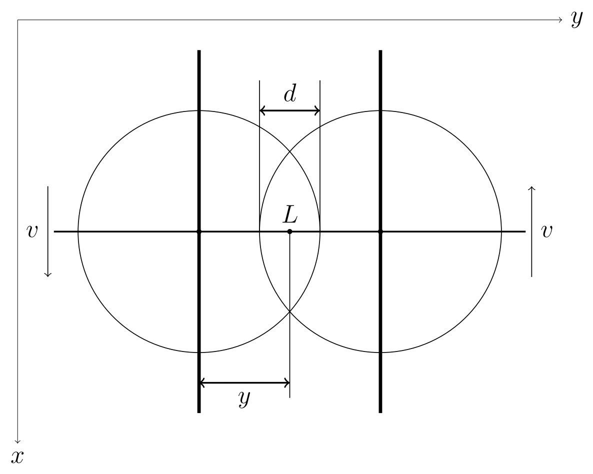


Figure 4.4: Material distribution of a point L on a plane: y is the distance of the point L to the first path; v the tool velocity, and d the overlapping distance

Let $t = x/v$, Eq.4.2 can be rewritten as:

$$q(y,v) = \frac{2}{v}\int_0^{t'} \dot{q}(r')dx\,, \quad \text{with}\, t' = \sqrt{R^2 - y^2}, \qquad r' = \sqrt{x^2 + y^2} \tag{4.4}$$

Eq.4.4 returns the paint thickness of the point L, for a single painting pass, given the distance y from tool center projected and the painting pass velocity v.

Optimal overlap distance and velocity on a planar surface

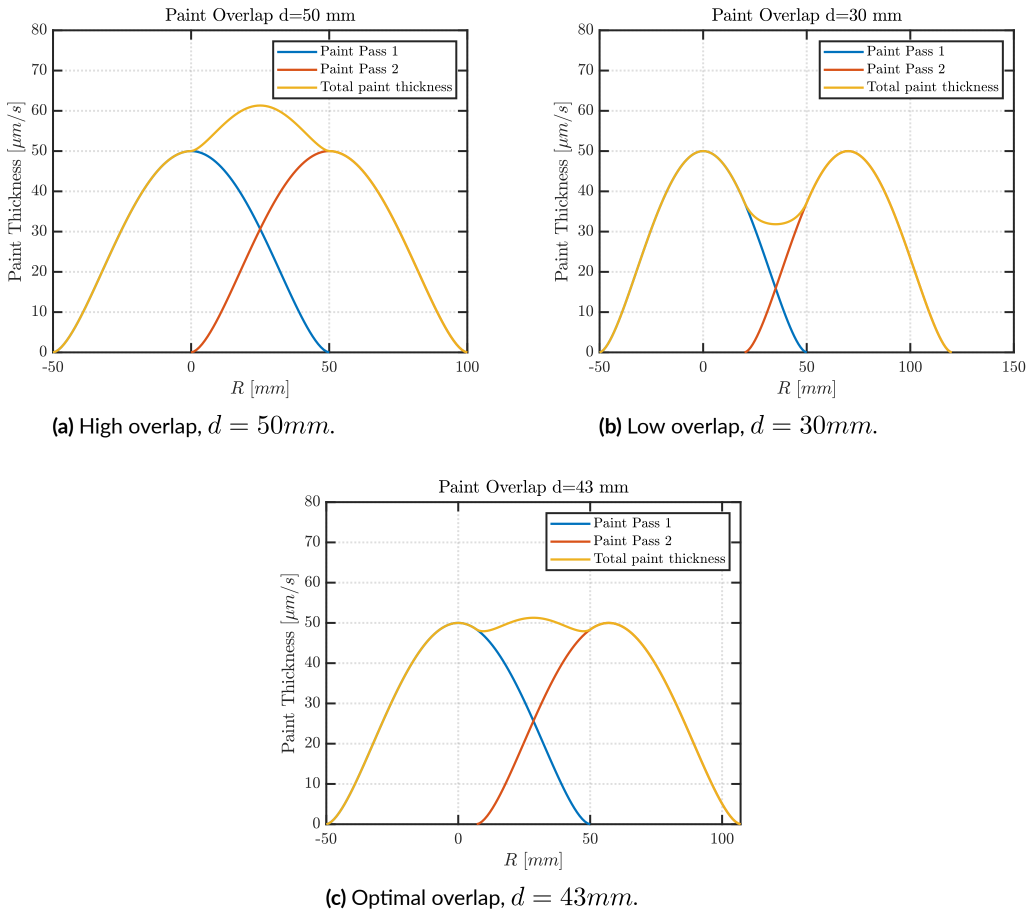
To have minimal paint thickness deviation we compute the optimal overlap d_{opt} between two adjacent paint pass and the overall optimal velocity v_{vopt}. Since the surface behaviour is unknown, and could be highly non linear, we compute these optimal values referring on a planar surface.

To compute optimal overlap d_{opt} we define the function:

$$\bar{q}(y,d,v) \triangleq \begin{cases} q_1(y,v) & 0 \le y \le R-d \\ q_1(y,v) + q_2(y,d,v) & R-d < y \le R \\ q_2(y,d,v) & R < y < 2R-d \end{cases} \tag{4.5}$$

where $q_1(y,v) = q(y,v)$ and $q_2(y,d,v) = q(2R-d-y,v)$.

The function 4.5 calculates paint thickness deposited by two neighbour pass on a point L, as it is shown in Fig.4.4. The final goal is to minimize paint thickness in the overall surface, and the overlap d_{opt} between two path is a critical parameter. In Fig.4.5 it is shown how important is to have an optimal overlap distance between two neighbour passes. In fact if d is too low it happens that it will be a region in which the paint thickness is lower than the desired paint thickness q_d, while if the overlap d is too high it will be a region with paint thickness value higher than the desired. With optimal overlap, instead, the paint thickness deviation is minimized.



(a) High overlap, $d = 50mm$.

(b) Low overlap, $d = 30mm$.

(c) Optimal overlap, $d = 43mm$.

Figure 4.5: Paint thickness for different values of overlap

To find the optimal overlap d_{opt} we define a cost function to be minimized in order to get the minimum paint thickness deviation. The cost function is:

$$E(d, v) \triangleq \lambda_1 E_1(d, v) + \lambda_2 E_2(d, v) \tag{4.6}$$

Where:

$$E_1 \triangleq \int_0^{2R-d} (q_d - \bar{q}(y,d,v)^2)dy \tag{4.7}$$

$$E_2 \triangleq (\bar{q}_{max}(d) - q_d)^2 + (q_d - \bar{q}_{min}(d))^2 \tag{4.8}$$

$$\lambda_1 = \frac{1}{2R-d}, \qquad \lambda_2 = 1 \tag{4.9}$$

Eq. 4.7 is necessary to penalize the mean square error of the material quantity (i.e. paint thickness) deviation from the required material quantity q_d, while Eq. 4.8 penalizes the material quantity deviation from the average material quantity. The parameter λ_1 is chosen to take an average of $E_1(d,v)$ in $[0, 2R-d]$.

The overall minimization problem is:

$$\min_{d \in [0,R], v} E(d,v) \tag{4.10}$$

Since Eq.4.4 is inversely proportional to tool velocity v, Eq.4.5 can be rewritten as:

$$\bar{q}(y,d,v) = \frac{1}{v}\rho(y,d) \tag{4.11}$$

and the same could be said for $\bar{q}_{max}(d)$ and $\bar{q}_{min}(d)$.

To find the minimal $E(d,v)$, first compute

$$\frac{\partial E(d,v)}{\partial v} = 0 \tag{4.12}$$

The solution of Eq.4.12 is given by:

$$v = \frac{\frac{1}{2R-d}\int_0^{2R-d} \rho^2(y,d)dy + \rho^2_{max}(d) + \rho^2_{min}(d)}{q_d\left(\frac{1}{2R-d}\int_0^{2R-d} \rho(y,d)dy + \rho_{max}(d) + \rho_{min}(d)\right)} \tag{4.13}$$

Since tool velocity v can be expressed as a function of the overlapping distance, then Eq.4.6 can be minimized by properly choosing the distance d. To solve Eq.4.10 we use Pattern Search Method (described in Sec.3.3) to calculate the optimal overlapping distance d_{opt}.

Constraints

A good paint quality is defined by deviation in paint thickness from the desired paint thickness. For this reason the constraint is on paint thickness deviation:

$$\Omega_s \triangleq \min |q_d - q_s| \forall s \in \{1, \ldots, n\} \tag{4.14}$$

Other constraints such as material waste, cycle time, reachability, are neglected in this work.

4.1.3 Object alignment

The main assumption of this algorithm is that the patch to be processed has its oriented bounding box with a corner corresponding to the origin of the world frame and the main edge aligned with x axis. The oriented bounding box is found using Principal Component Analysis (PCA), which is a fast method but could be inaccurate in some cases (when the point distribution is not uniform) leading to sub optimal solution. For this reason, to correct the rotation, we divide the bounding box in half with respect to x coordinate, keeping a free region on the middle to avoid that the minima of the two regions coincide. Then we take the point with minimum y coordinate in both halves, obtaining two points, a and b, where a is the point in the half with x coordinate closer to 0. Then we compute the vector $\boldsymbol{l} = b - a$, the angle α of $\boldsymbol{l}$ with respect to x axis and we rotate by $-\alpha$.

4.2 Coverage Algorithm

Here the main algorithm is described. In first sections we describe how the trajectory for a paint pass is generated, then how we simulate resultant paint thickness for each paint pass and then the velocity optimization part. Finally we describe how this process is iterated for all paint passes until the final trajectory is generated and the surface is fully covered.

4.2.1 Surface section generation

The generation of surface sections starts from the origin of axis and proceeds along x axis until the end of the bounding box is exceeded. We call T_k the triangle k in the surface and $p_{T_k}(x, y, z)$ its centroid. To generate each surface section we sort the triangles T_k to be painted inside the range $0 \leq (x, y) \leq 2R - d_{opt}$, where d_{opt} is computed as described in Sec.4.1.2. Since it is not guaranteed that all the sorted triangles can be reached by the spray, we sort again all triangles, starting from lower y to the higher y (the added stripes of triangles are along x axis), until the spray area, taken as πR^2, is not exceeded.

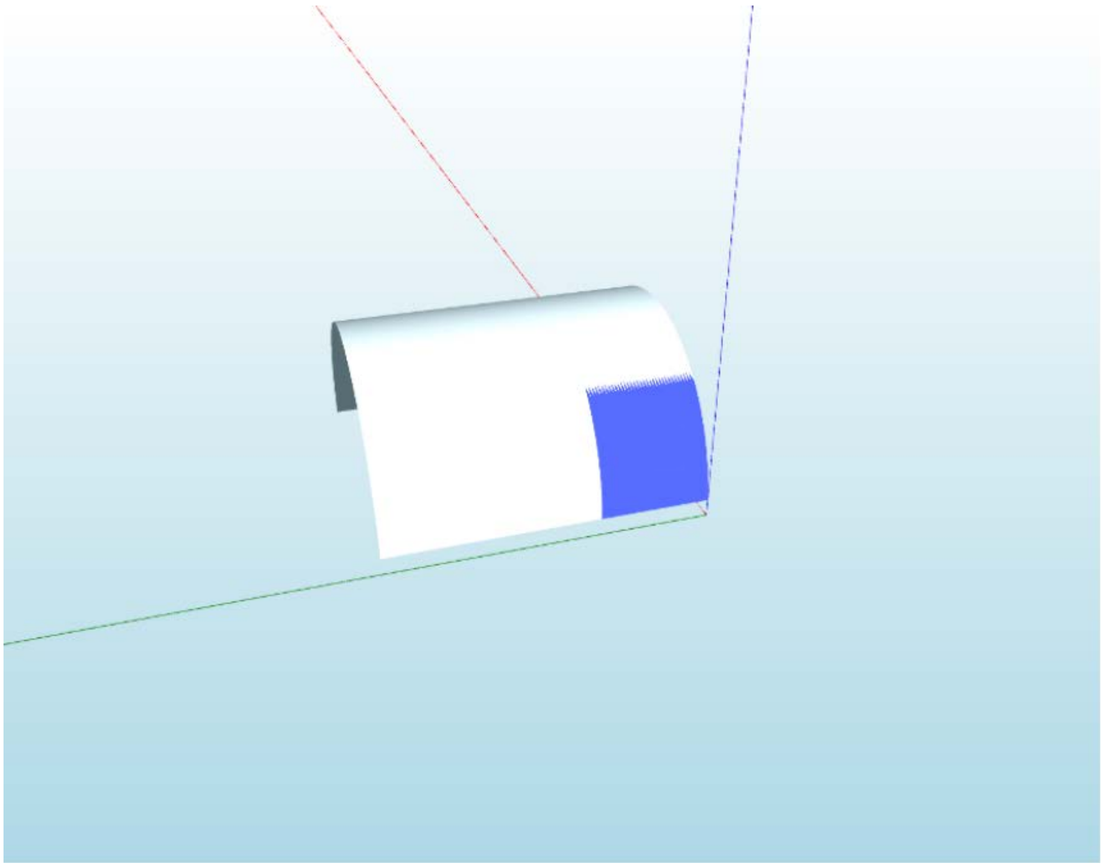


Figure 4.6: A Surface Section generated by the algorithm

4.2.2 Surface Section point and surface section normal generation

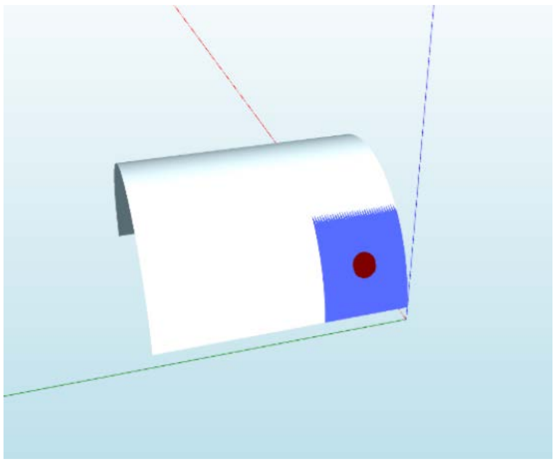
Once the triangles on surface section are sorted, the centroid $p_{ss}(x, y, z)$ and its normal $\boldsymbol{n}_{ss}(x, y, z)$ are computed by:

$$p_{ss}(x, y, z) = \frac{\sum_{s=1}^{m} p_{T'_s}(x, y, z) A_s}{\sum_{s=1}^{m} A_s} \tag{4.15}$$

where $p_{T'_s}(x, y, z)$ is the centroid of the triangle s in current surface section, A_s is its area and m is the total number of triangles in surface section.

$$\boldsymbol{n}_{ss}(x,y,z) = \frac{\sum_{s=1}^{m} \boldsymbol{n}_{T'_s}(x,y,z)A_s}{\sum_{s=1}^{m} A_s} \Bigg/ \left\| \frac{\sum_{s=1}^{m} \boldsymbol{n}_{T'_s}(x,y,z)A_s}{\sum_{s=1}^{m} A_s} \right\| \tag{4.16}$$

where $\boldsymbol{n}_{T'_s}(x,y,z)$ is the normal of the centroid of triangle s in surface section.


(a) Surface point generated.

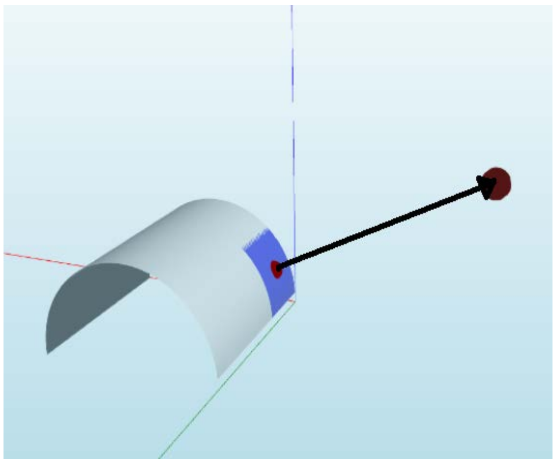

(b) Trajectory point generated.

Figure 4.7: Surface Section main points

4.2.3 Spray pattern implementation: Paint pass 1

To perform the whole trajectory we use a raster spray pattern (Fig.4.8), although this approach can eventually be applied to any spray pattern.

After a surface section, with its centroid and normal, is computed as described in Secs.4.2.1 and 4.2.2, the next surface section is generated using an incremental distance x_{incr} along x axis. The new surface section is generated in the range $(x_{pre} + x_{incr}, y_{pre}) \leq (x, y) \leq (x_{pre} + x_{incr} + 2r - d_{opt}, y_{pre})$, where x_{pre}, y_{pre} are the coordinates of previous surface section. This process is iterated until x axis limit of the object's bounding box is exceeded. For computational reason x_{incr} cannot be too small but at the same time cannot be too high because trajectory and resulting paint would be affected by the surface curvature. Moreover, we implemented a variable x_{incr}, as described in [10], but we see by testing that it gives no improvements, adding a computational effort to the algorithm. For this reason we choose $x_{incr} = R/5$, which we found by trials to be a good trade-off between computation complexity and trajectory generation accuracy.

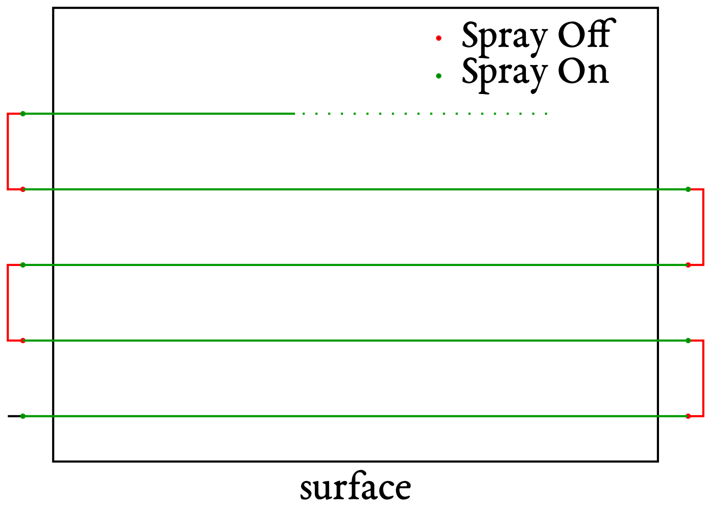


Figure 4.8: A raster spray pattern

To have a raster spray pattern with the lateral movement outside the object (Fig.4.8), the turning points between two consecutive passes, where also the spray has been toggled on/off, are generating using a surface section located at a distance x_{incr} from the border of the patch. In this case, and when a surface section contains no triangles, the centroid $p_{ss}(x, y, z)$ is set to the middle of surface section area (intended as $(x_{pre} + x_{incr}, y_{pre}) \leq (x, y) \leq (x_{pre} + x_{incr} + 2r - d_{opt}, y_{pre})$), while the normal is set parallel to z axis.

Afterwards all the unuseful points of the trajectory (i.e. the ones that has no triangles in its corresponding surface section) are removed, except for the ones where the spray needs to be toggled on/off.

Filtering

Due to irregular structure of the mesh, computed trajectory is not smooth (especially in y direction), so the generated points are filtered with a double sided moving average smoothing filter (MA) in y direction, because we want to have a line as smooth as possible to let the robot moving uniformly. The moving average filter is described by Eq.4.17:

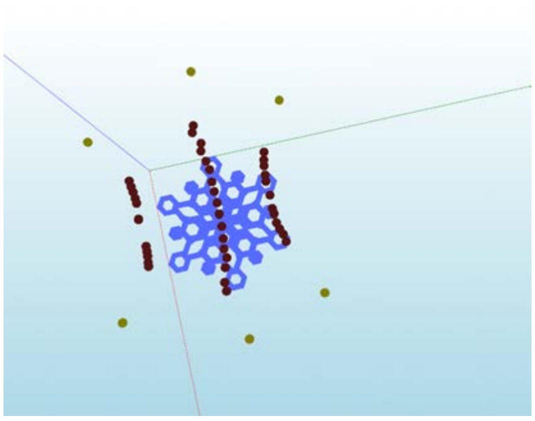
$$p_{ss,j}(y) = \frac{1}{2M+1} \sum_{i=-M}^{M} p_{ss,i}(y) \tag{4.17}$$

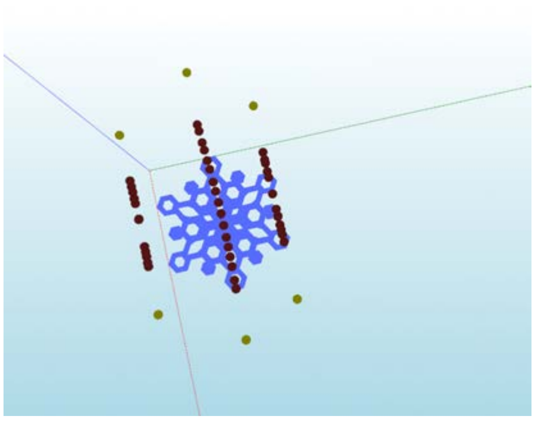
To improve the filtering result, after using the moving average, we use a least square filter (LS), as described in Sec.3.5. Firstly, we fit the polynomial which describes the trajectory and then we project all the points in the fitted function. Moreover, to have a description of the trajectory which allows the robot to move uniformly, the points are fitted with a fixed distance d, giving a more homogeneous description. By experimental result we have found that a good trade-off between trajectory approximation and capability of robot to follow smoothly the generated trajectory is using $d = 30$ mm (where d is the fixed distance between two points mentioned above). In this way y direction is filtered with moving average filter, and then fitted with a 4^{th} degree polynomial. The choice of degree is done by considering the fact that we not desire too much overfitting, because the robot trajectory should be as linear as possible in y direction, but the shape of the trajectory should also be preserved. Subsequently the trajectory is filtered in z direction, only using a least square estimator. For this purpose we choose a 6^{th} degree polynomial, to smooth the trajectory preserving its profile in z direction.

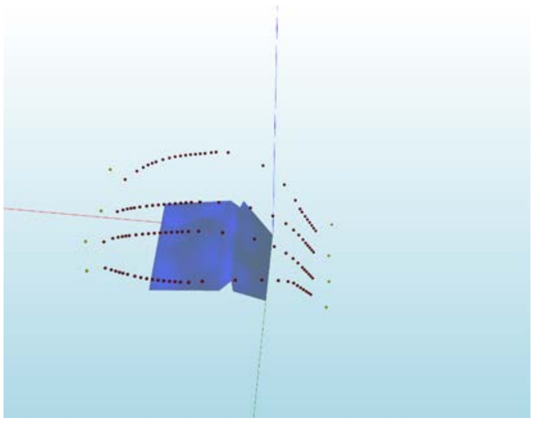
Simulation shows that different kind of filtering could lead to more uniform trajectories (Fig. 4.9) than the ones generated without filters. Depending on the surface, it is not guaranteed that filtering the trajectories with Least Square in both y and z direction is the best choice, in fact in Figs 4.9c-4.9f are shown different filters on the same surface, and the only way to determine which is the best is by trial. Moreover, we have found experimentally that Least Square filtering in both y and z direction is the filter that gives the better trajectories in most cases. Summing up, where possible, least square instead of only moving average is preferred to have the possibility to describe the trajectory with less points without losing accuracy, even if they both lead to similar results, which leads to a better implementation in the real robot.

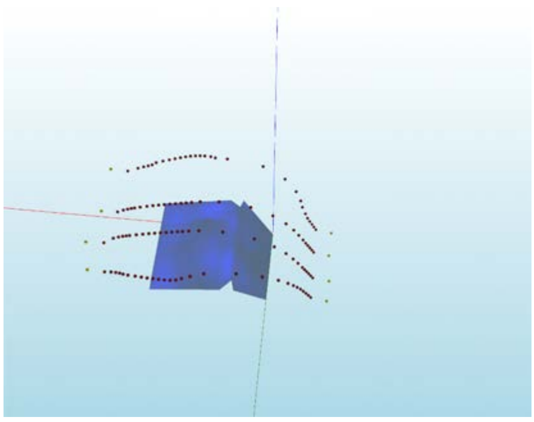
4.2.4 Paint Thickness function for Paint pass 1

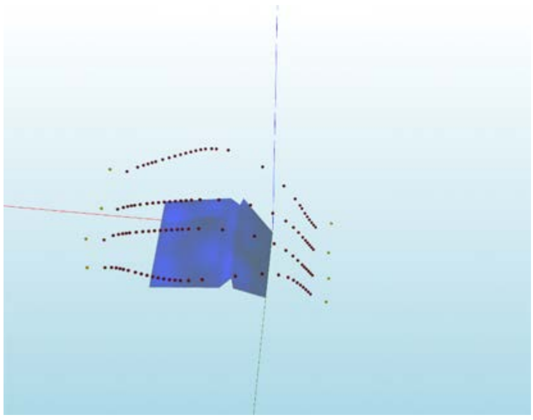
While the spray gun tip moves over the surface with a gun stand-off h, paint is deposited on the surface. Then is necessary to compute paint deposition over the surface to have a simulation of the final result and to have the capacity to improve the paint deposition, having a feedback on how the final result would be. Paint thickness for the triangulated surface is computed by considering gun velocity and inclination of the tool with respect to the surface and paint distribution, while we assume a linear motion over triangles when the gun moves between two subsequent surface section points. The paint thickness function for a triangulated surface with T number of triangles is computed as described in [10] and [12] and


(a) No filters on Snowflake.


(b) MA and LS filters on Snowflake.


(c) No filters on a chair.


(d) MA filter on a chair.


(e) LS filter on a chair.

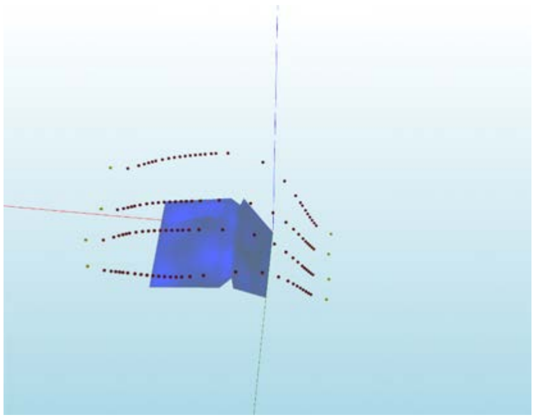

(f) MA and LS filter on a chair with Z filtering.

Figure 4.9: Filtering effects on trajectory generation

results in:

$$q_s \triangleq \begin{cases} \left(\frac{h}{l_s}\right)^2 \frac{\cos\gamma_s}{\cos^3\theta_s} q(y,v) & \gamma_s < 90 \\ 0 & \gamma_s \geq 90 \end{cases} \tag{4.18}$$

where γ_s is the angle between normal of the spray and normal of the point s on the surface, θ_s is the angle between the spray normal and the segment l_s, which is the one that connects the spray tool position with the point s. This formula takes into account both the paint deposition on a planar surface $q(y, v)$ (Eq.4.4) and the different orientation between the spray path and the surface using Area Magnification approach [12].

To have a correct paint deposition computation over the surface we need to sort again the surface triangles T_k which composes the paint pass. In fact the triangles in surface section are sorted in the range $(x_{pre} + x_{incr}, y_{pre}) \leq x, y \leq (x_{pre} + x_{incr} + 2r - d_{opt}, y_{pre})$, but the spray has diameter of $2R$. For this reason, the triangles that compose the paint pass are sorted taking the ones that are closer than R from the surface section point. The paint thickness of a each triangle is computed considering as the point s in Eq.4.18 its centroid $P_{T_k}(x, y, z)$ and for each point s the contribution of each spray paint point is summed to obtain the final paint thickness. For this reason, the more the surface is refined, as described in Sec.3.1, the more realistic the paint thickness simulation is.

4.2.5 Optimal speed for paint pass

If the surface is not planar, using constant speed for the entire paint pass could lead to bad painting. In fact we want a uniform velocity on the surface to have uniform paint deposition, and this velocity could be different from the robot end-effector velocity. For this reason a speed optimization of the end effector is performed.

Speed section computation

We use an approach similar to the one used for computing surface section (Sec.4.2.1) to compute speed surface sections. For each surface section point i which composes the paint pass trajectory, as speed section it is considered the union of surface sections relative to point i, $i - 1$ and $i + 1$. This is done to have more uniformity on paint pass speed.

Then for every velocity section i we compute the optimal speed as described in the next Section, that will be the speed of pass point i.

Optimal speed computation

For each triangle $k \in 0, \dots T$ in the speed section the paint thickness function

$$f(q_k) = q_k \cdot v_{opt} \tag{4.19}$$

is computed, where q_k is the paint thickness of triangle k and v_{opt} is the speed of paint pass i computed for a planar surface. In this way $f(q_k)$ does not depend any more on v and it is possible to compute q_k from any v without recomputing paint deposition. Indeed for any v:

$$q_{k,v} = f(q_k)/v \tag{4.20}$$

Then the optimal speed for the considered speed section is computed by defining the optimization problem considering the constraint given in Sec.4.1.2: in this work we use an Adapted Genetic Algorithm (Sec.3.4) to compute $v_{opt,i}$, which is the optimal speed for pass point i. The performance of Genetic Algorithm is strongly related to the choice of fitness function W, but since it is an heuristic algorithm its results could be different on every run, though all very similar between them.

Firstly we need to compute for each triangle k the paint difference function F_k:

$$F_k = (|q_{k,v} - q_d|), \quad \text{where } k \in (1, 2, \dots T) \tag{4.21}$$

where q_d is the desired Paint Thickness.

Since the goal of optimization is to find v such that there are as few as possible triangles k outside the maximum paint deviation Δq and as much as possible close to desired paint thickness q_d we define:

$$K \triangleq \sum_{i=1}^{T} k_i \quad \text{where } k_i = \begin{cases} \lambda_1 & F_k > \Delta q \\ \lambda_2 & 0.4 \cdot \Delta q < F_k < \Delta q \\ \lambda_3 & F_k < 0.4 \cdot \Delta q \end{cases} \tag{4.22}$$

where λ_1, λ_2, λ_3 are the hyperparameters and they should be chosen in a way that $\lambda_1 < \lambda_2 \leq \lambda_3$ in order to maximize the fact that more triangles are in the interval in which $F_k < 0.4 \cdot \Delta q$ and as less as possible triangle in the interval $F_k > \Delta q$.

From experimental results has been shown that good parameters are: $\lambda_1 = -0.2, \lambda_2 = 0.1, \lambda_3 = 1$.

Now the fitness function can be computed for the velocity section as:

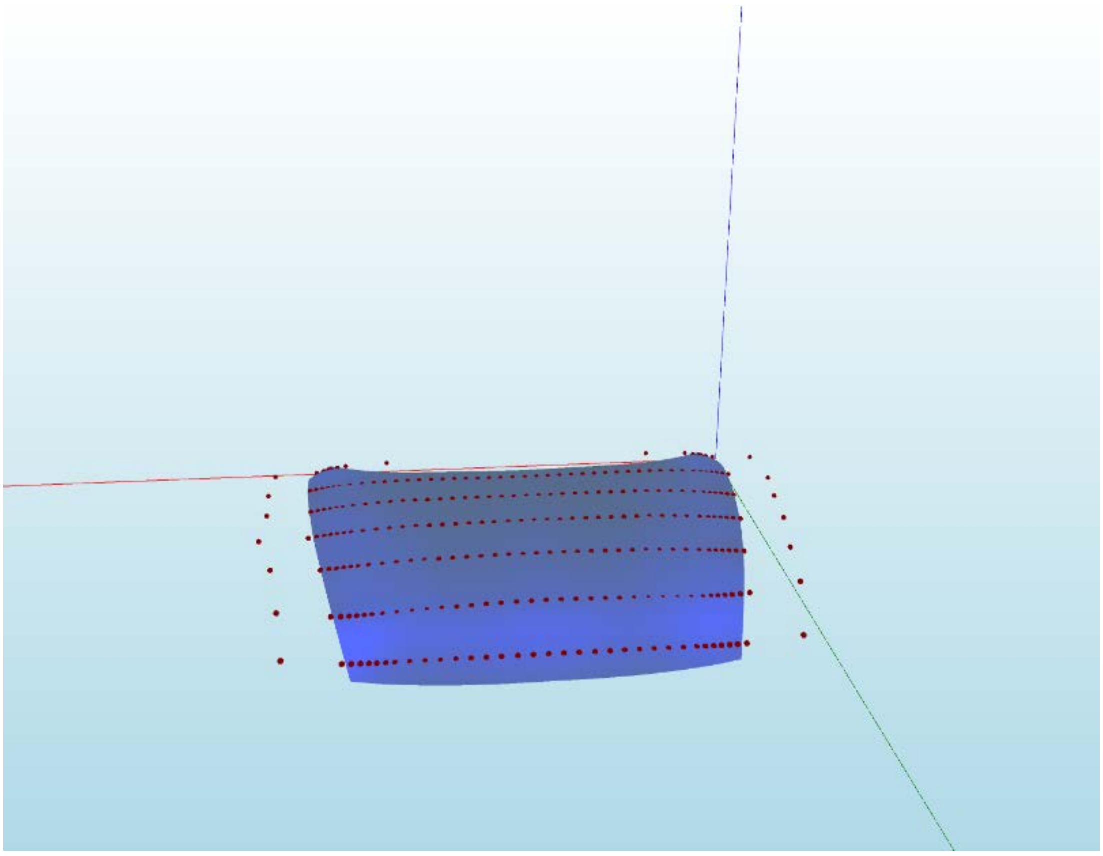
$$W \triangleq \begin{cases} 2^K & v_{min} \leq v \leq v_{max} \\ -\inf & \text{otherwise} \end{cases} \tag{4.23}$$

Looking at Eq.4.23 can be understood why in the termination condition $W > 1$ is required, if it is possible. In fact $W < 1$ means that $K < 0$, and this can happen only if there are more triangles in the speed section with $F_t > \Delta q$ than the ones with F_t in the desired range.

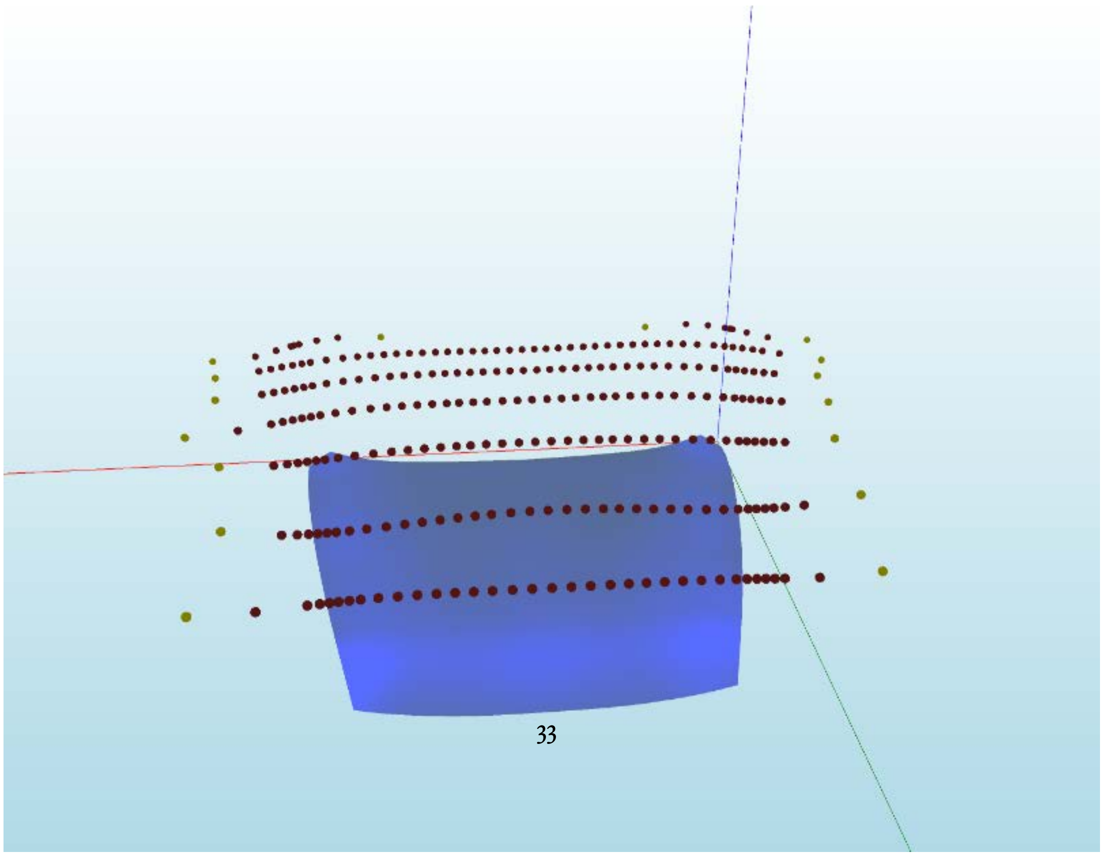
Also other fitness functions has been tried, such as the one described in [10], but the presented one gives speeds that leads to better paint thickness over the surface.

4.2.6 Spray pattern implementation: Paint pass n

To implement the raster spray pattern, after the first Paint Pass is computed, the y is incremented by $R - d/2$ from the y_m that is the average of y coordinate of previous pass points. Then x coordinate is decreased by x_{incr} until it goes to 0 and the paint pass is computed as described in Sec.s4.2.1-4.2.4. Then y coordinate is again increased by $R - d/2$ from y_m and this procedure continues iteratively until all the surface is covered (i.e. y is greater than the y dimension of the bounding box). When the algorithm terminates it has generated two trajectories showed in Fig.4.10: one on the object surface and one relative to the robot tool which is composed by the points of surface trajectory moved by a distance h along their normal.



(a) Surface



(b) Robot tool

Figure 4.10: Trajectories generated by the algorithm.

In the twenty-first century, the robot will take the place which slave labor occupied in ancient civilization.

Nikola Tesla

5 Algorithm implementation and verification

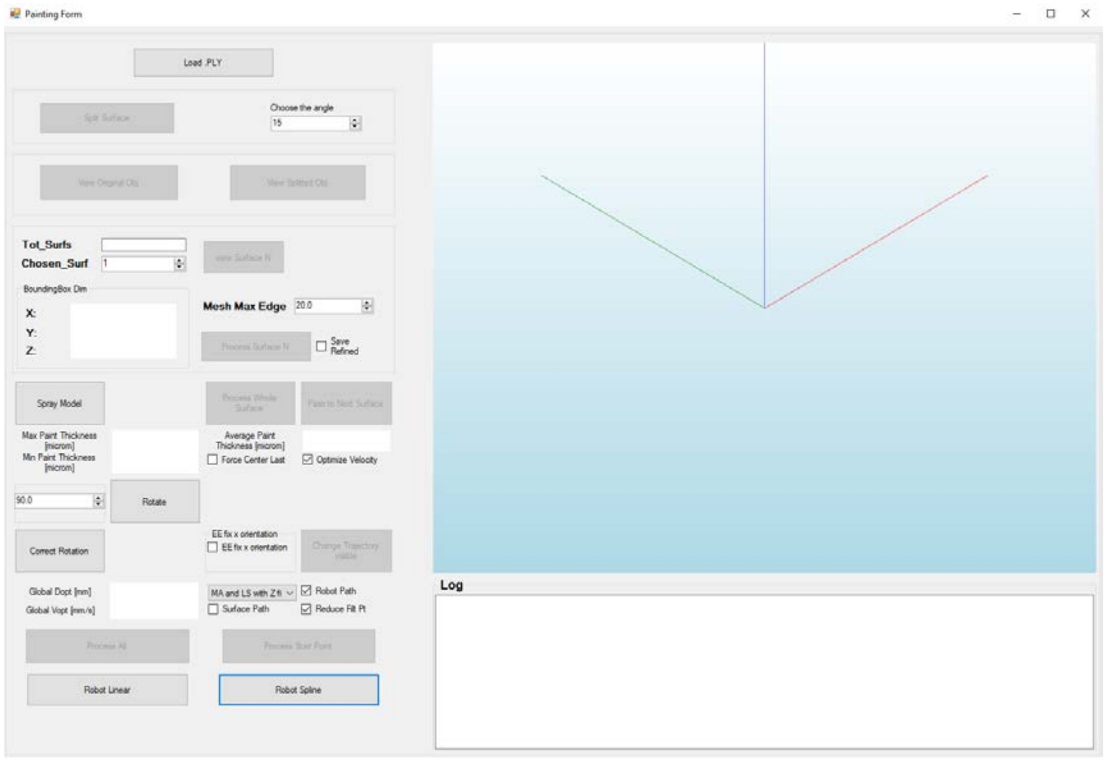
THE FINAL GOAL of this work is to verify the presented covering surface algorithm on a real robot manipulator. To do this, after the trajectories are generated, all the points from the object world have to be translated to the robot world. Then one must check if the trajectories can be executed by the robot without any collision and finally the robot program is generated to perform the task on the real object.

5.1 COVERING SURFACES WINDOWS PROGRAM

To run the algorithm and generate the robot program, we create a Windows program (Fig 5.1a). User can load a 3D model, configure all the main parameters (e.g. tool parameters, filter to use), generate the trajectory and simulate the final result of paint thickness on the surface.

Once satisfactory trajectories are generated the user can open a new window and choose the position in the robot world for the object to be painted (Fig. 5.1b).

Then all the points of the trajectories are translated in the robot world frame, as described in Sec.5.2.2, and trajectory planning is performed. Since the target object can potentially be placed anywhere in the robot world, trajectory planning is a fundamental step to check for collisions between robot and objects, and to check if all the points belong to the workspace of the robot (i.e. are reachable). Once the trajectory planning has been completed successfully the user can generate the robot program that will be uploaded in the real robot.



Choose the angle 15

Chosen_Surf 1

Mesh Max Edge 20.0

90.0

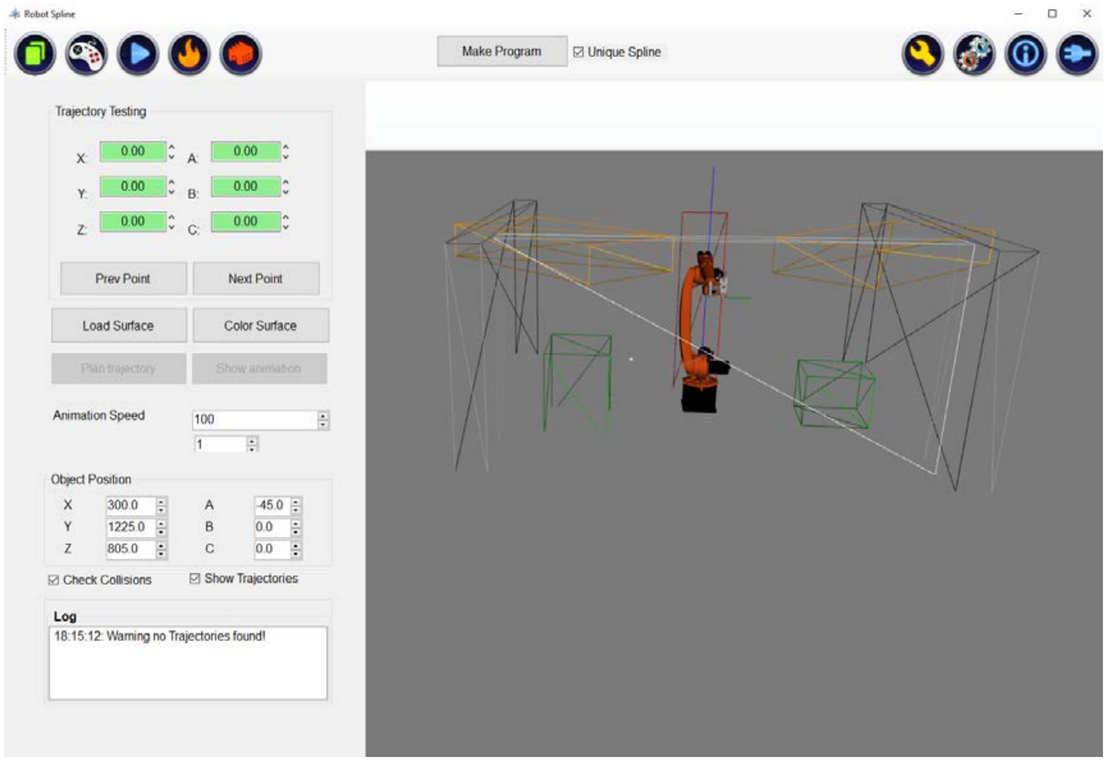
MA and LS with Z fi

☑ Optimize Velocity

☑ Robot Path

☑ Reduce Fit Pt

(a) Main form



☑ Unique Spline

X: 0.00 A: 0.00
Y: 0.00 B: 0.00
Z: 0.00 C: 0.00

Animation Speed 100
1

Object Position
X 300.0 A -45.0
Y 1225.0 B 0.0
Z 805.0 C 0.0

☑ Check Collisions ☑ Show Trajectories

```
18:15:12: Warning no Trajectories found!
```

(b) Robot Form

Figure 5.1: Windows of our program.

5.2 Robot program generation

In this section we describe all the steps that were necessary to generate a program that can be uploaded to the robot to perform the task.

5.2.1 Robot movement and point types

To perform a trajectory with the real robot a list of goal points must be passed to it. These points can be passed to the robot both with joint position coordinates and cartesian coordinates. Joint positions coordinates are unique, since they describe the effective joint position of the robot, while in cartesian coordinates the same point could be reached by the robot with different configuration (e.g. with an axes rotated by 360 degree). For this reason when a point in cartesian coordinates is given, it is possible to set also some configuration bits that describe the unique configuration of the robot.

A robot has a set of different types of movements to go from the actual position ($P0$) to a destination point ($P1$):

- LIN (linear): this type of movement makes the robot to perform a straight line from $P0$ to $P1$. This can be used to approximate a trajectory with a set of straight lines.
- CIRC (circular): this type of movement makes the robot to perform a circular movement from $P0$ to $P1$ passing through P_{01} which is the point where the curve changes its derivative sign.
- PTP (Point To Point): this type of movement makes the robot to go from $P0$ to $P1$ with the fastest path (i.e. the path with smaller axes movement). This usually leads to a curve path since usually robot joints are usually revolute.
- SPLINE: this is a motion type that is particularly suitable for complex, curved paths. A Spline block passes through all points with a single movement and constant velocity, using a spline approximation. Such paths can also be generated using approximated LIN and CIRC motions, but splines have advantages, such as there is no approximation in the path, in fact all points are reached precisely (thing that could not happen with LIN and CIRC), and constant velocity is better maintained. On the other side if the global trajectory contains a discontinuity point the robot could not perform as expected: it will perform a longer and different path to be sure to keep the velocity constant. For this reason, when we design the trajectory our algorithm is designed to make trajectories which do not contain discontinuity points in the paint pass. Some discontinuity points could be present when the robot moves from a paint pass to the next one but in that case the spray is off and will not affect the painting result.

All the movements of a robot are performed with respect to its Tool Center Point (TCP), which it is the frame that describes the position and orientation of the robot tool and usually is put on the tip of the tool, but can be set potentially everywhere. The robot TCP is computed by robot calibration (but can be also set numerically) and each robot can have different tools, where each one is represented by its TCP. Let us suppose that we want to set a TCP on the tool tip, then robot calibration consists of moving in the same fixed point as the tool tip (where we want to set the TCP), with different orientations. This way, the robot is able to compute its new TCP frame. Finally, when a destination point in cartesian coordinates is given to the robot, the tools to which it refers must be specified, otherwise a default one will be used, leading to unexpected behaviours, because a robot reaches the point passed as goal with its TCP, so he needs to know which one has to use to perform the movement correctly.

5.2.2 Adding painted objects to the robot world

The coverage algorithm is executed in the object world. When an object is added to its world, it is put with its center on the world reference frame. So the object is aligned to x axis (as described in Sec. 4.1.3), it is necessary to keep track of the transformation between initial position and final position, and this is done with the homogeneous matrix ${}^{wo}T_o$ that describes this transformation.

In the robot world, instead, the robot is added with the robot base frame as world frame and the objects can be added everywhere in the world. To let the robot to perform the trajectory computed by the algorithm it is necessary to transform the trajectory points coordinates in the world reference frame that correspond to the robot frame.

Then, when the object is placed on the robot world, the homogeneous matrix that describes the object points as robot world points ${}^{wr}T_o = {}^{wr}T_b * ({}^{wo}T_o)^{-1}$ is generated, where ${}^{wr}T_b$ is the position of the object with respect to the robot world.

In this way the trajectory points p_o can be transformed in robot world coordinates with:

$$p_r = {}^{wr}T_o p_o \tag{5.1}$$

5.2.3 Trajectory planning

After the transformation of trajectory points as described by Eq.5.1, it is possible to plan the robot trajectory. Since the trajectory above the object is already planned by our coverage algorithm, the only trajectories to be planned are: the one that move the robot from hold position to the start position to perform the process, and the one that move it from the last process point back to hold position. In fact is supposed that the object to be painted is put in a position where the robot can perform the task without any collision. In industrial processes the hold position is a robot position that allows to load/unload the object to be processed (e.g. robot home position, a position out of the camera when there is a vision system, etc.). For the trajectory generated by our program, trajectory planning consist on checking if all the points generated are reachable by the robot, and, if they are not reachable, is necessary to change the pose of the object on robot world, that obviously reflects object position in the real world with respect to the robot base. Since the trajectory points are put at a distance of $d = 30$ mm between them (by design), and in reality the robot cannot jump instantaneously from a point to the next one, these points (which are the ones that will be passed to the robot) are interpolated using the robot movement described above (e.g. LIN, PTP), then interpolated trajectory reflects the movement that the real robot will perform. During this process we also check if robot trajectory has some collisions with other objects in the world, doing this in all interpolated points. For the trajectories generated by the algorithm, if there are collisions, the only solution is to move the object somewhere else, while for the movements from and to hold position the trajectory planner manage themselves to find, if possible, a trajectory without collisions, with the only prerequisite that start and end points are in a non-colliding position. Collision checking is a essential step because non collaborative industrial robot has, depending on robot model, a maximum axes velocity in order of 1-2 m/s, which is translated to a maximum end effector velocity even bigger. For this reason a collision with a human or an object will cause serious damages. Moreover, they are not capable to check when a collision happens in real world. For this reason collision checking is an essential step as much as modelling accurately in simulation the real world near robot to be sure that the robot will not collide even in the real world.

5.2.4 Robot program generation

When the trajectories are correctly planned, the robot program that will perform the process on the real robot can be generated. Since some robots cannot have all types of movements described in Sec.5.2.1, here we adopt two types of program to perform the trajectories:

1. LIN motions to perform the paint pass and PTP to perform all the other movements where the spray is off.

2. Singular SPLINE movement to perform all the covering process and PTP to move from/to hold position.

In both programs PTP movements are performed passing the point in robot joint position to be sure that the robot reaches it with the same configuration of the simulation, while other points are given in cartesian position. Points given in cartesian position are generated in a way that their rotations are not too different between two consecutive points. In this way it is guaranteed that the robot do not change configuration during these movements. Moreover as hold position we set the robot home position.

LIN motion trajectories

With this program we set the TCP on the tip of robot tool (Fig.5.2), so the trajectory is made of points generated at a distance h from the surface points along their normal (Fig. 4.10b), where h is defined in Sec.4.1.2. LIN motions are used during the paint pass to have

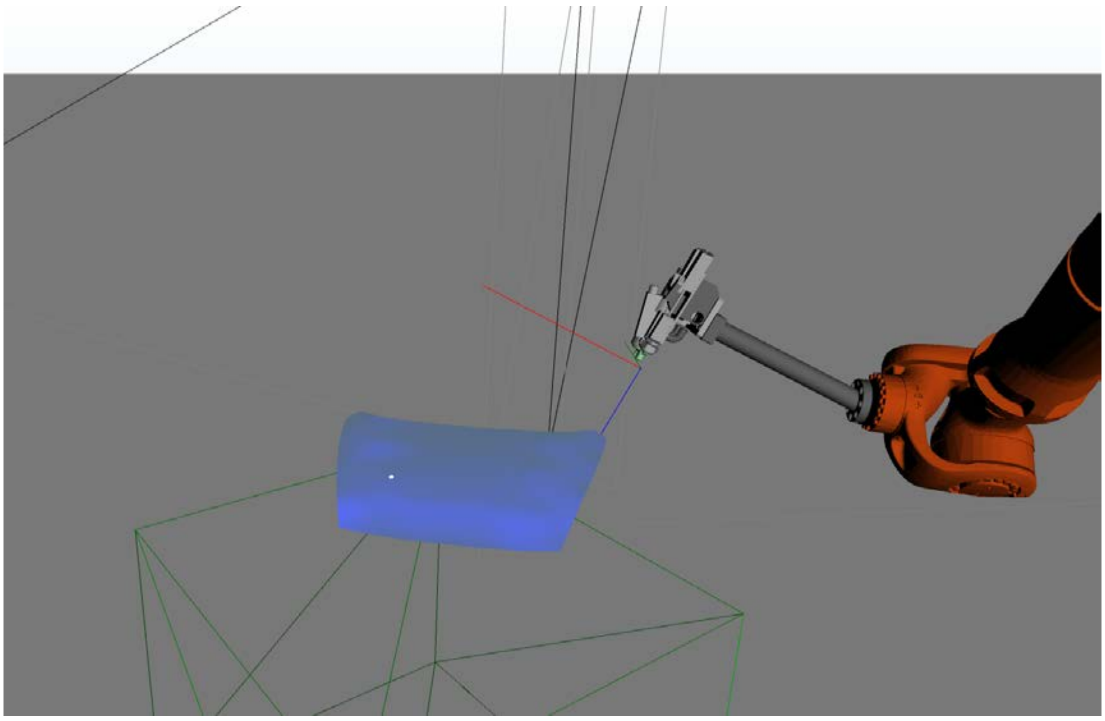


Figure 5.2: Robot with TCP on the tip of its tool

a straight trajectory between two points when the spray is open. This has been done to guarantee a fixed orientation during the paint process. Moreover to each LIN movement the optimal velocity for the end effector is given, as it is computed in Sec.4.2.5, to obtain the optimal paint thickness on the surface. Outside the spraying area (i.e. where the paint pass is performed) the movements are done with PTP movement, which allows to put the robot in a good axes positions to perform the next paint pass (i.e. with joints far from their limits). Unfortunately, due to limitations on robot axes velocities and accelerations, it is not guaranteed that the robot can perform the LIN movement with given velocity causing a worse painting quality than the one expected by simulation. However, due to the filtering that we performed on velocities, it is reasonable to expect that velocities performed by the robot are not too far from the optimal ones.

SPLINE motion trajectories

Since Kuka robots (and some other brands) have the capability to perform SPLINE movement, in this project we use this type of movement. The main advantage of SPLINE movements is that the given trajectory is performed with constant velocity on TCP. For this reason we set the TCP on the surface of the object (Fig.5.3) and in this case trajectory points are the one generated on the surface (Fig.4.10a). In this way we can set as velocity of spline

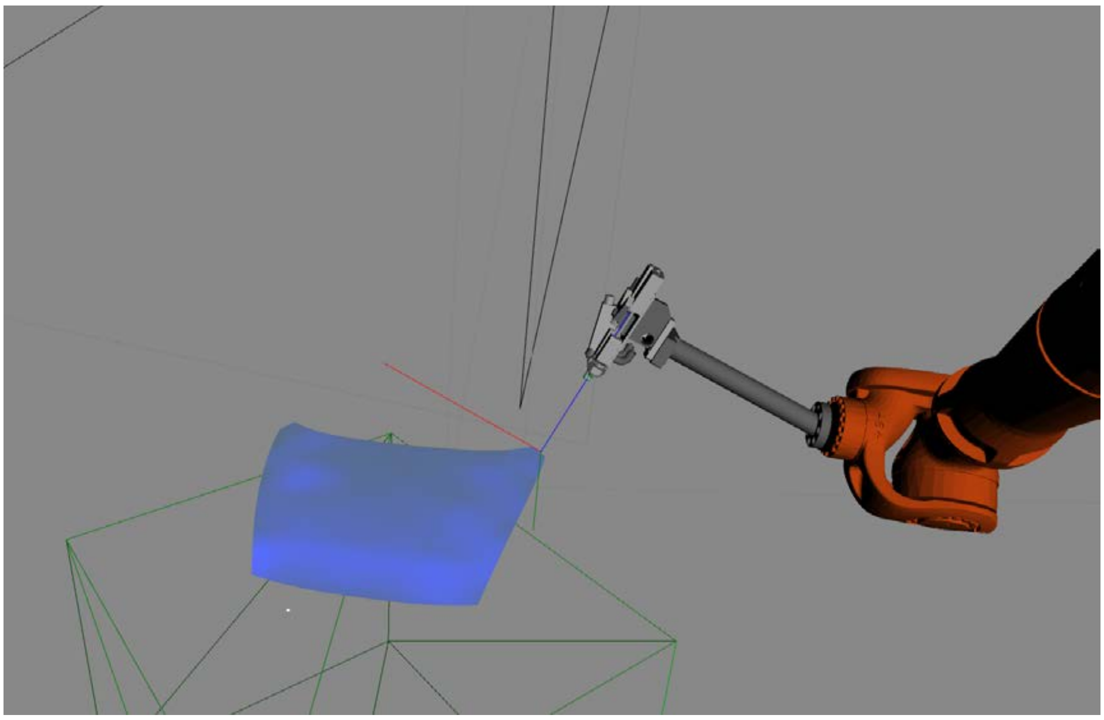


Figure 5.3: Robot with TCP on the surface

block the one computed on Sec.4.1.2 because we desire to have this constant optimal velocity on the surface during the entire painting process, regardless of the end effector velocity. With spline block the controller of the robot, computing the path, takes care of the physical limits of the robot. Then, if the velocity and the path is feasible, the robot computes the process at given velocity, while, if it is not feasible, it computes the process at the higher possible velocity. This moreover guarantees a smoother movements during the process and a better final result due to the uniformity of the movements. Finally the movements to and from hold position are made in PTP, following the trajectories generated by the planner, while the rest of the process is done with one unique spline block.

5.3 Real experiments

In this section we describes the validation of our algorithm, firstly running it from a scanned mesh, and then running it on a real robot.

5.3.1 Algorithm test with object from scanner

As first experiments we test our algorithm with some mesh objects retrieved from a Photoneo (in Fig.5.4) which is a 3D scanner. The mesh obtained by the PhotoNeo is not as perfect as a 3D model, but the algorithm is still able to generate a good trajectory, exploiting the capability of this work to be adapted for a Real-Time utilization and to work without having the model of the object to be painted.

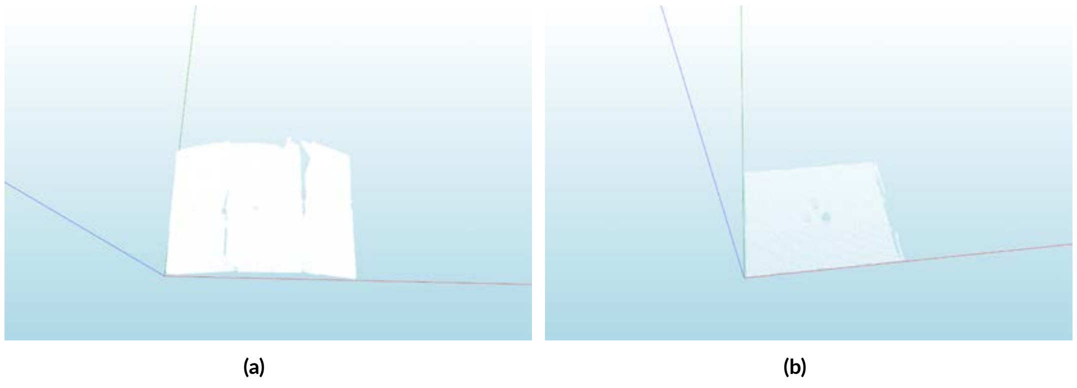
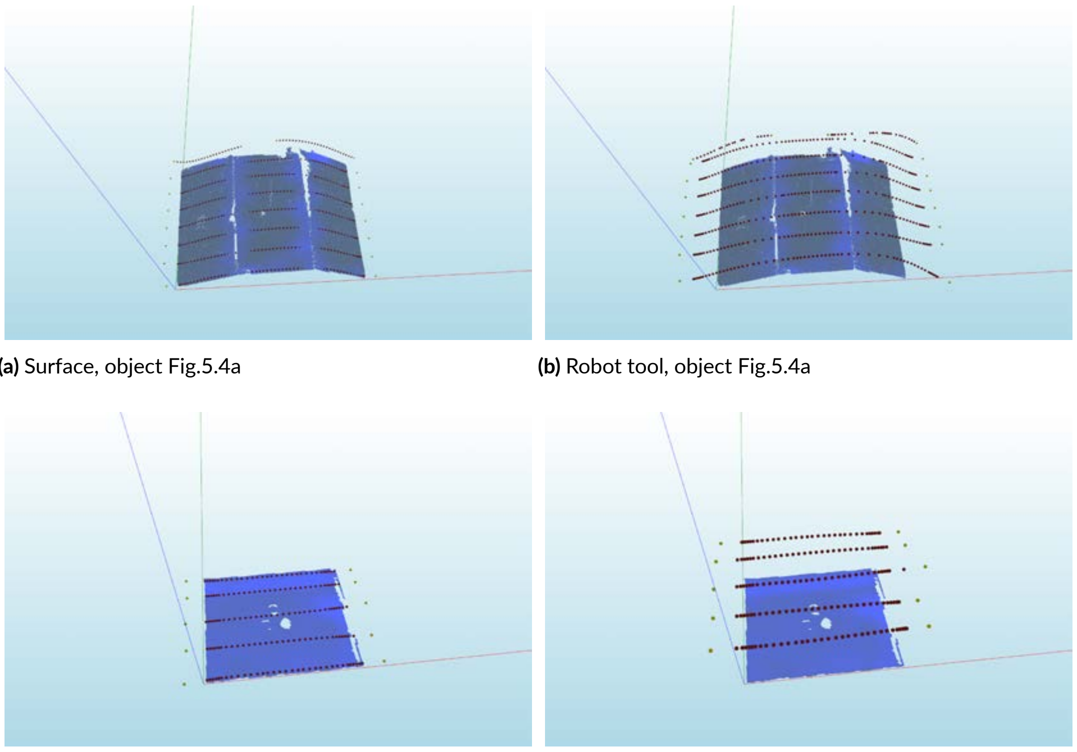

(a) (b)

Figure 5.4: Objects scanned with the PhotoNeo.

The trajectories generated from our algorithm are shown in Fig.5.5 and, despite the noise added by the scanner, the trajectories are good enough to give an accurate painting result, making our algorithm robust to noise, as long as it does not distort the shape of the surface.


(a) Surface, object Fig.5.4a **(b)** Robot tool, object Fig.5.4a

(c) Surface, object Fig.5.4b **(d)** Robot tool, object Fig.5.4b

Figure 5.5: Trajectories generated by the algorithm.

Therefore, the main problem we found using mesh object from a scanner is the noise. In fact here the mesh of the objects in Fig.5.4 is manually refined removing all the parts which correspond to noise or parts which are not part of our object, like the floor. Moreover, doing this, we have tried to simulate an automated way of noise reduction, meaning that we have removed all the noise outside the target surface, isolating the target object from the rest of the scan. However, the noise on the surface to be painted is kept to mimic an automated object selection, but it is possible to perform some ad hoc processes to refine a noise object to get a better model.

5.3.2 Robot implementation

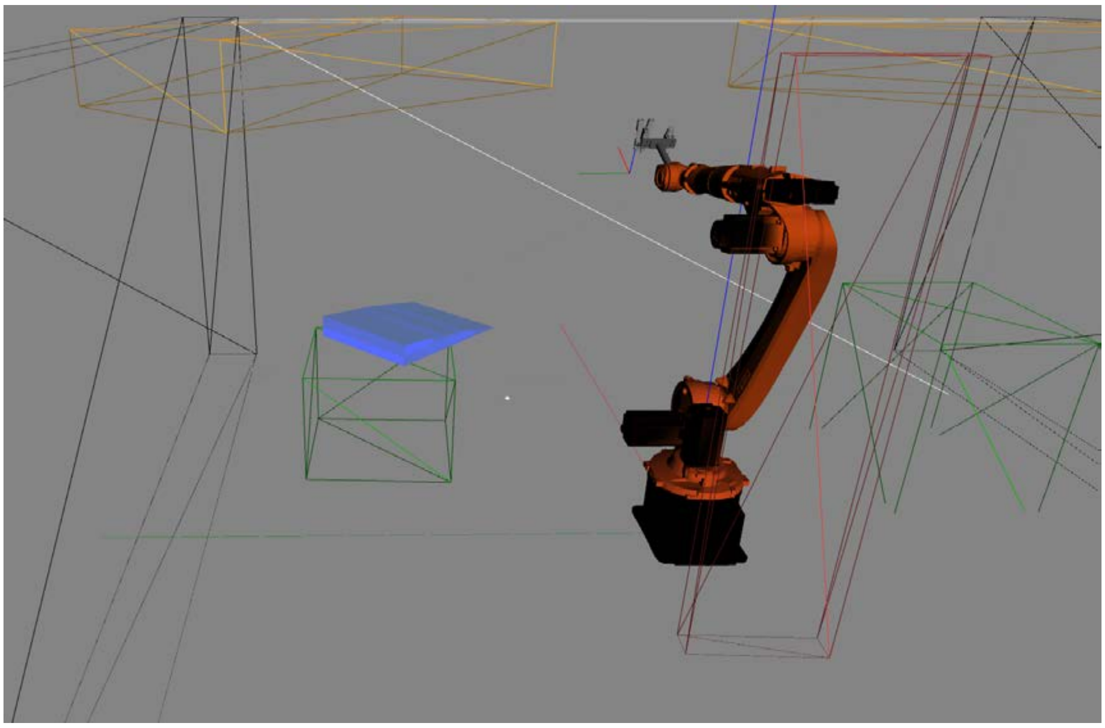
Once the robot program has been generated, we tested it in a real industrial cell. Unfortunately, it was not possible to test the resulting paint thickness, because the only available cell was a spray glue dispenser robot, but it was still possible to test the generated trajectory and the robot movements on a target object. The robot used for the experiments was a Kuka KR16 R1610, with a KRC4 controller, which is capable to perform spline movements. This robot is composed of 6 revolute joints and the tool is attached at a distance of 30 cm from the robot end effector, inclined by an angle of 20 degrees. Like all industrial robots it has different working mode, which set different limits on maximum axes velocity. These mode are necessary to test the robot programs and verify that the robot does not perform unexpected behaviours which could lead damages to people or things in working range of the robot. After we checked that the program generated works as expected, for the tests we set the velocities at the maximum possible, without any limit on joint velocities and accelerations. In Fig. 5.6a is shown the simulated environment of the working cell, where the parallelepipeds are the collision objects, while the real cell is shown in Fig.5.6b.

The object where we test the algorithm is in Fig.5.7a, and the use of the PhotoNeo was not necessary because we have the CAD model of the object on which we run the experiments. Moreover we perform the trajectory tests only in the diagonal surface of that object (Fig.5.7b) because in a real spray application the object would be put in a support which would make possible to reach all its faces, but here we were capable only to paint that surface.

The trajectories generated are shown in Fig.5.7, and has been tested with both programs generated in Sec.5.2.4, with the robot at maximum speed.

With LIN movements and optimal velocities computed in Sec.4.2.5 the robot did not perform as expected, in fact when two consecutive points have too much displacement in orientation the robots tends to slow down, not reaching the optimal velocity and leading to an overpainting in that zone. Using, instead, SPLINE movements they guarantee that the motions of the robot are performed with a constant velocity on the surface, leading to an overall smoother process and expectedly a better paint result.

In both cases the robot programs automatically generated by this program work without any problem, making this work a good approach to automatic offline programming of painting robots, remembering that, as we show with experiments, is preferable to use SPLINE movements to obtain a better process result.


(a) Simulated

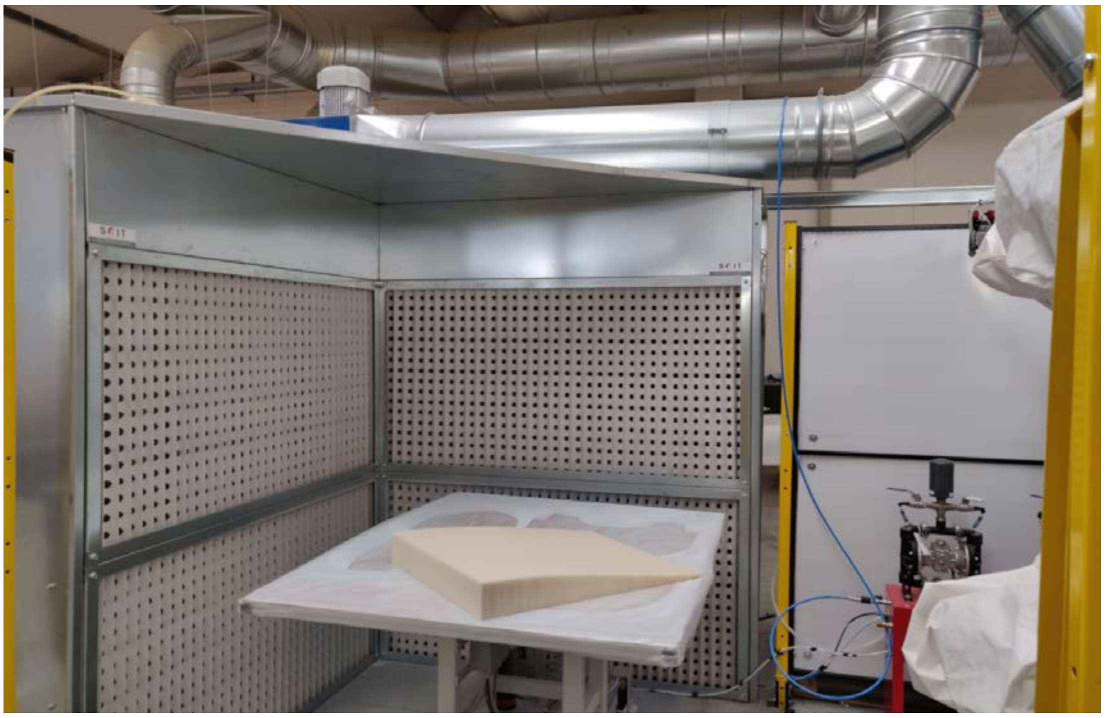
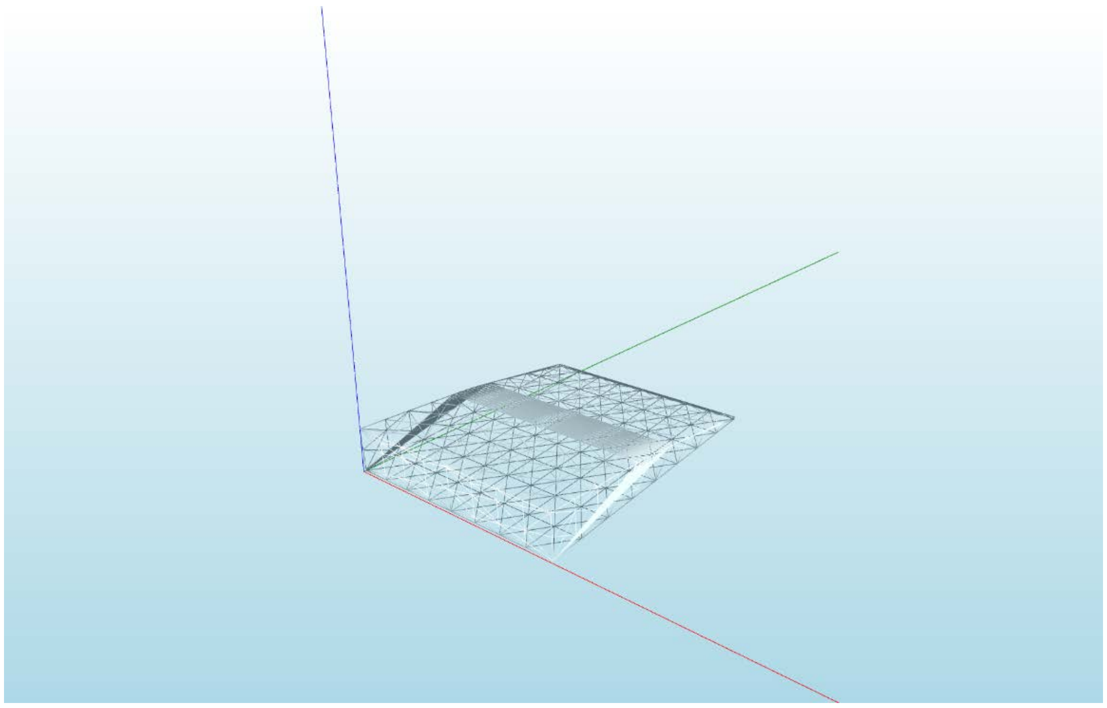
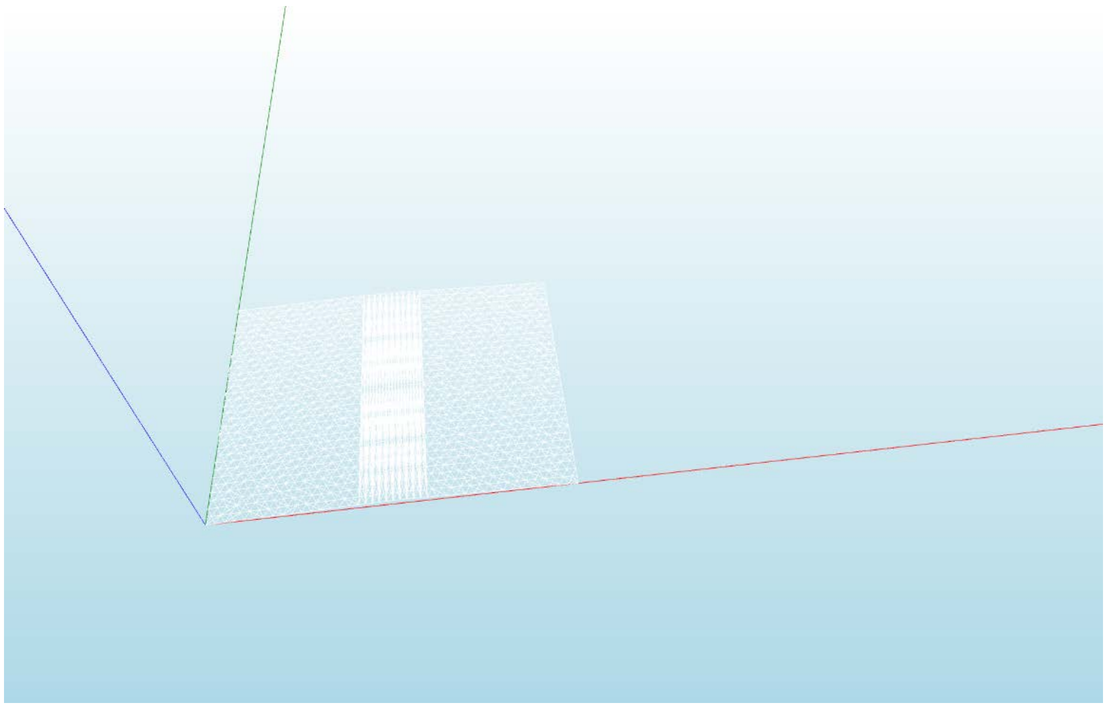
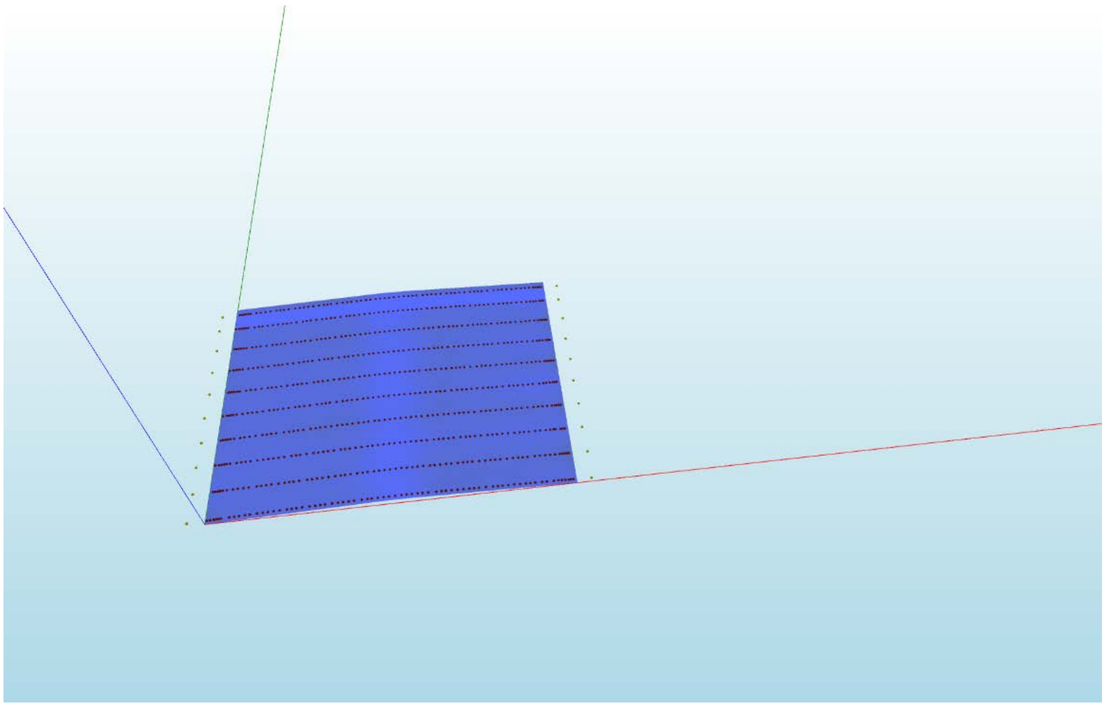

(b) Real.

Figure 5.6: Robot working cell


(a) Complete object.


(b) Target diagonal surface.


(a) Surface

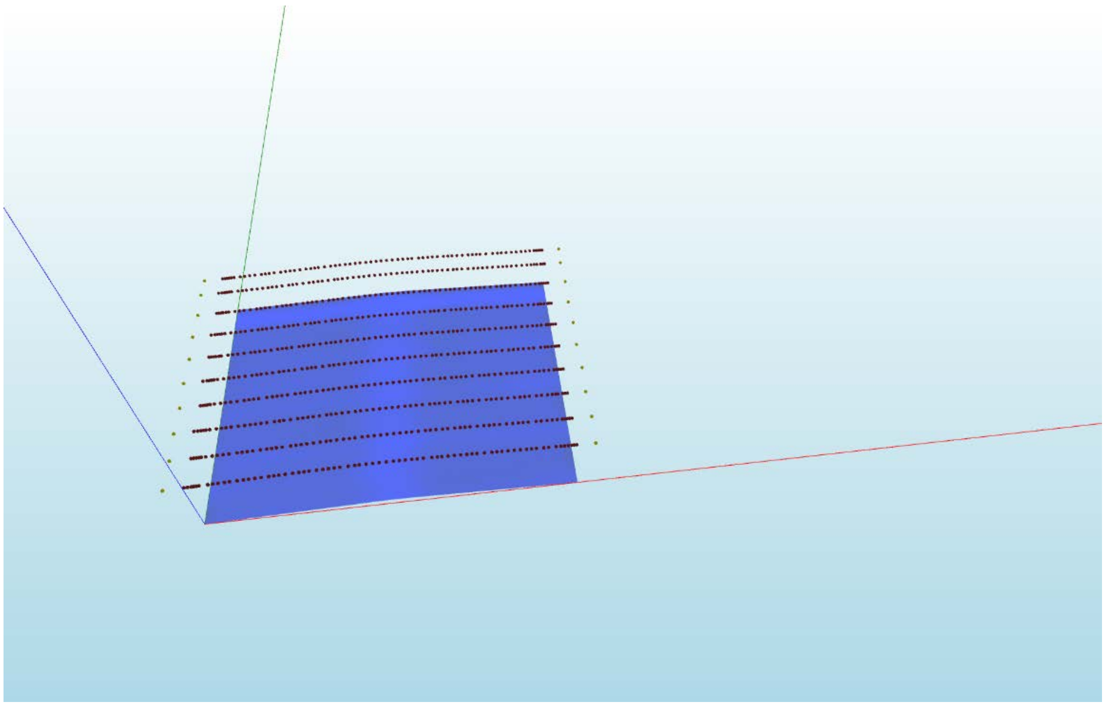

(b) Robot tool

Figure 5.7: Trajectories generated by the algorithm.

6 Conclusion

In this work the problem of covering surface with a robot manipulator from a 3D drawing is faced. The main advantage of 3D drawing is that it can be derived both from a design model, both from a scan. Here the focus was on spray painting process, but this algorithm can be easily adapted to other processes by changing the tool model and some other ad hoc parts (like velocity optimization or paint thickness computation). The first step of robot trajectory generation has been to split the object in simpler surfaces, to obtain surfaces without high curvatures and to allow the algorithm to perform better. Then the object has been reassembled in the robot world to be painted. An important factor of this algorithm is that the trajectories generated optimize the resulting paint thickness deviation, to get the most uniform paint possible.

For the path of the final trajectory a raster spray pattern is adopted, with all the lateral movements performed outside the surface and with the spray turned off. First of all, the optimal overlap d_{opt} and the optimal velocity v_{opt} for a paint pass on planar surface are calculated. Then for each paint pass the algorithm generates surface and trajectory points, computing the centroid and normal of each surface section, and then all points are filtered using Least Square filter. Once the paint pass is generated for each point the optimal velocity of the robot end effector is computed with an Adapted Genetic Algorithm. After the paint pass is done, $y_{incr} = y_m + R - d_{opt}/2$ is calculated, where y_m is the median of y coordinate of all points which compose the surface trajectory and y_{incr} is the y value of the next paint pass. This process is iterated until all the surface is covered.

Finally we plan the trajectory on a simulated robot to verify that is feasible and collision checking is performed. Then the robot program is built automatically to perform the task. A particular focus is given in the use of SPLINE movement for robot programming, which, given v_{opt} as velocity of the block, guarantees that the velocity of paint sprayed on the surface is constant and equal to v_{opt}, regardless of robot end effector velocity (unless it reaches the physical limits of the robot).

The main advantage of our work is that the procedure of painting an object and writing the robot program is done almost entirely without human interaction, making it a fast way to program an industrial painting cell. This work can also be adapted for real time trajectory generation, since our algorithm is fast, in particular if spline movements are used, which does not requires the speed optimization for each trajectory point. Moreover, there are some possible improvements and parts that are not taken into account in this work. For example, it is possible to change the surface division method, using an ad hoc method that splits the object in optimal parts to be painted. In fact the patch division algorithm we adopted not always divides the patch in optimal ones. For example in cylindrical object the algorithm will fail because for a surface section there are more triangles with the same (x, y) centroid, but with different z. This is because continuous cylindrical object has no faces to be split in with the method we adopt. A solution is to split this object in two halves as in Fig.6.1 and paint each half at time. Moreover the images taken from a scanner, like we have done with the PhotoNeo, needs further preprocessing before they can be used in the algorithm to avoid unwanted trajectories. In fact a mesh taken from the scanner needs be cut and filtered to get rid of noise introduced by the sensor and this part was not automated in our work.

Other problems could be painting zones inside a hole (e.g. the inside of a box), because due to the limited space and the possible collisions with the object is needed an ad hoc trajectory. Furthermore the borderline effects of painting process are not taken into account. In fact when the object is resembled painting a patch could paint also some parts of its neighbourhood patches causing overpainting. Regarding optimization of the trajectories, here we consider only the paint thickness deviation, but other optimizations, such as on material waste or path execution time, can be used and it would be possible to generate different trajectories based on what we want to optimize.

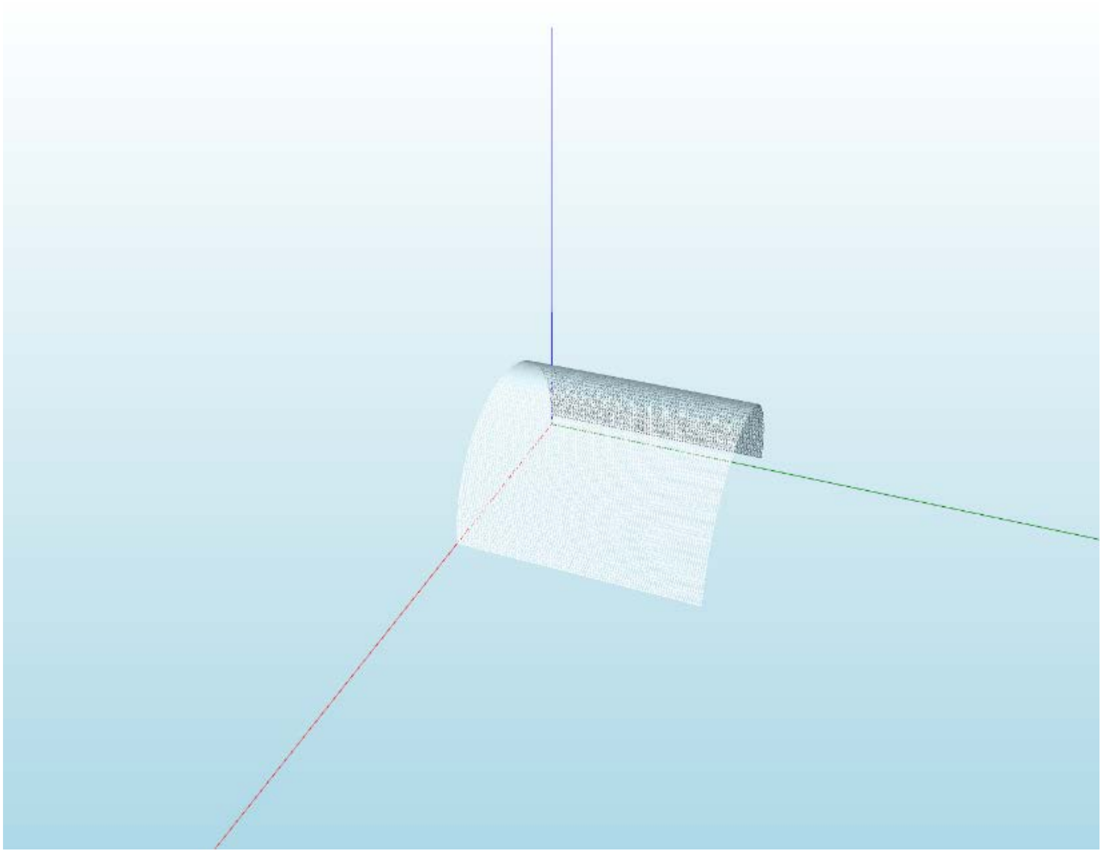


Figure 6.1: Cylindrical object split in a way that could be painted

References

[1] P. Hertling, L. Hog, R. Larsen, J. W. Perram, and H. G. Petersen, "Task curve planning for painting robots. i. process modeling and calibration," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 2, pp. 324–330, April 1996.

[2] J. Moura and M. S. Erden, "Formulation of a control and path planning approach for a cab front cleaning robot," *Procedia CIRP*, vol. 59, pp. 67 – 71, 2017, proceedings of the 5th International Conference in Through-life Engineering Services Cranfield University, 1st and 2nd November 2016.

[3] D. Nakhaeinia, R. Fareh, P. Payeur, and R. Laganière, "Trajectory planning for surface following with a manipulator under rgb-d visual guidance," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Oct 2013, pp. 1–6.

[4] R. Chen, G. Wang, J. Zhao, J. Xu, and K. Chen, "Fringe pattern based plane-to-plane visual servoing for robotic spray path planning," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 3, pp. 1083–1091, June 2018.

[5] M. Li, Z. Lu, C.-f. Sha, and L. Qing Huang, "Trajectory generation of spray painting robot using point cloud slicing," *Applied Mechanics and Materials*, vol. 44, 12 2010.

[6] F. Remondino, "From point cloud to surface: The modeling and visualization problem," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 34, 03 2004.

[7] J. Lv, X. Chen, J. Huang, and H. Bao, "Semi-supervised mesh segmentation and labeling," *Comput. Graph. Forum*, vol. 31, pp. 2241–2248, 2012.

[8] H. Zhang, C. Li, L. Gao, and G. Wang, "Hierarchical mesh segmentation based on quadric surface fitting," in *2015 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)*, Aug 2015, pp. 33–40.

[9] L. Guillaume, D. Florent, and B. Atilla, "Curvature tensor based triangle mesh segmentation with boundary rectification," in *Proceedings Computer Graphics International, 2004.*, June 2004, pp. 10–25.

[10] M. V. Andulkar and S. Chiddarwar, "Incremental approach for trajectory generation of spray painting robot," vol. 42, pp. 228–241, 05 2015.

[11] L. Fa-zhong, Z. De-an, and X. Gui-hua, "Trajectory optimization of spray painting robot based on adapted genetic algorithm," *Proceedings of IEEE International Conference on Measuring Technology and Mechatronics Automation (ICMTMA 2009)*, vol. 2, pp. 907 – 910, 05 2009.

[12] H. Chen, N. Xi, and Y. Chen, "Multi-objective optimal robot path planning in manufacturing," vol. 2, pp. 1167–1172 vol.2, Oct 2003.

Acknowledgments

I would like to thank everyone who has helped me in this work, starting from Ing. Roberto Polesel, who has given me the opportunity to do this thesis with its company and helped me in the various step of this work, and Silvia Pontarollo for helping me fix the inevitable issues that occur during the development of a project. I thank my professor, Dr. Ruggero Carli, who made me passionate about these topics, in particular in robotics, during my academic studies and helped and supported me in this project. I also thank all the employees of Euclid Labs S.r.l. for their willingness to help me solve any type of issue during my project and for making it a pleasant experience in the company.