Robot trajectory generation, simulation and execution for shotcrete automation

Master degree in Electrical and Electronic Engineering

Gonçalo Carriço Moniz

Leiria, September of 2023

Robot trajectory generation, simulation and execution for shotcrete automation

Master degree in Electrical and Electronic Enginering

Gonçalo Carriço Moniz

Project Report under the supervision of Professor Hugo Filipe Costelha de Castro (hugo.costelha@ipleiria.pt).

Leiria, September of 2023

Originality and Copyright

This project report is original, made only for this purpose, and all authors whose studies and publications were used to complete it are duly acknowledged.

Acknowledgments

This work was partially supported by the project RoboShot@FRC – Robotic System for Optimized Projection of Fiber Reinforced Concrete in Railway Tunnels (POCI-01-0247-FEDER-047075 and LISBOA-01-0247-FEDER-047075), under the Programa Operacional Competividade e Internacionalização (POCI), and Programa Operacional Regional de Lisboa, through Fundo Europeu de Desenvolvimento Regional (FEDER). This project was also partially financed by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia, within project UIDB/00308/2020 and under the Scientific Employment Stimulus – Institutional Call CEECINST/00051/2018.

Resumo

O *shotcrete* é um método de engenharia civil e de minas amplamente usado, que consiste em projetar betão a alta velocidade usando equipamento especializado. Hoje em dia, a trajetória da ponteira de projeção ainda é controlada por um operador (manualmente ou por teleoperação). Automatizar este processo tem o potencial de melhorar a eficiência de *shotcrete*, a qualidade, e segurança dos trabalhadores. Este relatório detalha o trabalho feito no âmbito do projeto RoboShot@FRC para automatizar a geração e execução de trajetórias para aplicação robótica de *shotcrete* no reforço de túneis ferroviários. O trabalho desenvolvido centrou-se em duas componentes principais: uma que gera um conjunto de poses com base numa malha de *input* da superfície a ser projetada (programa de geração de trajetórias) e outra que simula o processo de *shotcrete*. Foram realizados testes num ambiente simulado, e alguns testes também foram realizados em robôs reais. Com o objetivo de maximizar a homogeneidade da camada de betão, os programas de geração de trajetória e simulação de *shotcrete* foram usados em conjunto para otimizar os parâmetros da trajetória.

Palavras-chave: automação de *shotcrete*, geração de caminhos robóticos, manipulação robótica, simulação de *shotcrete*.

Abstract

Shotcrete is a widely used civil and mining engineering method that consists of projecting concrete at high-speed using specialized equipment. Nowadays, the trajectory of the projection nozzle is still controlled by an operator (manually or by teleoperation). Automating this process has the potential to improve shotcrete efficiency, quality, and worker safety. This report details the work done within the RoboShot@FRC project, to automate the generation and execution of trajectories for robot-based application of shotcrete in the reinforcement of railway tunnels. The developed work focused on two main components: one that generates a set of poses based on an input mesh of the surface to be shotcreted (path generation program), and another one that simulates the shotcrete process. Extensive tests were done in a simulated environment, and some tests were also performed in real robots. The path generation and shotcrete simulation programs were used together to optimize the trajectory parameters, in order to maximize the resulting concrete layer homogeneity.

Keywords: shotcrete automation, robot path generation, robotic manipulation, shotcrete simulation.

Contents

List of Figures

List of Tables

1. Introduction

Shotcrete is a civil and mining engineering method which consists of spraying concrete at high speed using specialized equipment [1]. It is widely used for tunnel and mine reinforcement [2], [3] but also for building and repairing other infrastructure such as domes, dams and architectonic artworks [4], [5]. Usually this process involves a shotcrete machine which is remotely controlled by an operator in real-time [1], [6], [7] or a nozzle-man which physically handles the nozzle.

The applied concrete layers need to have a designated minimum thickness to ensure structural integrity while avoiding over application to reduce costs [1]. An important shotcrete factor that affects material-efficiency is rebound, which is the mass of concrete which does not adhere to the sprayed surface, and is therefore wasted [8]. Rebound may be quantified as the ratio between the concrete volume that does not adhere to the sprayed surface, and the total volume of the projected concrete.

The shotcrete process creates a hostile environment for the workers involved and can contribute to the deterioration of their health [9], [2], [7]. Furthermore, the operator needs to be highly skilled to correctly control the spraying trajectory, and the crew of workers needs to be experienced [9], [5], [10], [11]. According to [9], highly proficient shotcrete operators are difficult to recruit which can cause labour shortages.

In any case, human control and perception can be replaced and improved upon by using modern technologies. Automating the shotcrete process, by implementation of shotcrete robots, has not only the potential to relieve workers from adverse work conditions, but also to improve quality (proper layer thickness) and efficiency [12], [13], [10], [2]. The job of the operator, which is critical to ensure proper layer thickness, can be done by an autonomous shotcrete robot which relies on the use of sensors to assess the surrounding environment and computer software to control the nozzle pose.

No mass-produced commercial autonomous shotcrete robots have been identified [9], however, a Finnish company, Normet, has announced it will release a fully autonomous shotcrete system in the future [14]. Furthermore, several papers have expanded upon this possibility such as [7], [12], [1], [9], [2]. The work described in this report distinguishes

itself from these articles by allowing different path types to be generated, by providing a more detailed explanation of developed path generation methods, and by developing a path parameters optimization algorithm based on shotcrete simulation.

The work described in this report was done in the context of the project RoboShot@FRC – Robotic System for Optimized Projection of Fiber Reinforced Concrete in Railway Tunnels. This project included the participation of several researchers from IPLeiria (Polytechnic University of Leiria) and the University of Minho, as well as from the companies Leirimetal and EPOS. The goal of the project was to create a system to automate the shotcrete process with a focus in railway tunnel applications. With regards to this project the author of the current report co-authored 2 publications: [15] and [16]:

- G. Moniz and H. Costelha, 'Path Generation and Execution for Automatic Shotcrete in Railway Tunnels', in *2023 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Apr. 2023, pp. 214–219. doi: 10.1109/ICARSC58346.2023.10129548.
- J. A. O. Barros *et al.*, 'A Multidisciplinary Engineering-Based Approach for Tunnelling Strengthening with a New Fibre Reinforced Shotcrete Technology'. Rochester, NY, Jul. 07, 2023. doi: 10.2139/ssrn.4503644

The first publication focused on the path generation aspect of the RoboShot@FRC project. The second publication expanded upon the several systems being developed for the RoboShot@FRC project, namely, tunnel assessment by LiDAR scanning, development of fiber injection prototypes and the development of the path generation program (with this last part corresponding to part of the work described in this report). The RoboShot@FRC project included several goals in different fields, such as: creating new reinforced concrete mixes with improved properties; developing a mixing system to provide for real-time adjustable reinforced concrete mixtures, particularly regarding the introduced fibers; real-time laser scanning for continuous shotcrete corrections; and the development of a robot-based automated shotcrete system, with this last part being the one described in this report.

The work presented in the current report was developed within the RoboShot@FRC project with the goal of creating a program to generate robotic trajectories automatically. To figure out which trajectory parameters result in the most homogeneous concrete layers, a shotcrete simulation program was also developed.

This document is divided in 6 chapters. The current chapter introduces the work done. Chapter 2 consists of a review of the scientific literature concerning shotcrete automation. Chapter 3 expands upon the computer program developed to generate and execute robotic paths for shotcrete automation. Chapter 4 describes the development done to simulate the shotcrete process, in order to optimize the path generation process. In Chapter 5 the tests and results are revealed. A conclusion of the work done as well as suggestions for future work are presented in Chapter 6.

2. Literature Review

This chapter reviews the current literature regarding the shotcrete process and the work done to automate it. For this review, the search process was done using the keywords “shotcrete automation” and “robotic path planning”.

2.1. The shotcrete process

The shotcrete process, or method, consists of projecting concrete at high speed onto a surface using specialized equipment [4], [5]. It is used for tunnel and mine reinforcement [2], [3], [8], but also for other applications such as swimming pools, domes, dams, slope stabilization and architectural artworks [4], [5]. It can be used both to build infrastructures as well to repair them [5], [17]. While certain types of concrete may be used specifically for shotcreting, it should be understood that shotcrete is a method and not a type of material [5].

Concrete itself consists of a composite material which includes a binding medium and embedded aggregate particles. Usually Portland cement, which is a hydraulic cement, is used as a binder [18]. As water and hydraulic cement bind together (hydration) the concrete sets, which means it loses plasticity while gaining strength [19].

The use of the shotcrete method has several advantages over the traditional cast-in-place concrete method, such as being more time-efficient, but also because it can be used in situations where the traditional method is not practical or feasible [4], such as tunnels or domes.

When concrete is applied, it needs to be compacted, which increases its density by removing entrapped air. Compaction is quite important because it increases abrasion resistance and improves the concrete’s strength [20]. With the shotcrete method, the force with which the concrete hits the sprayed surface causes compaction [4].

There are two main types of shotcrete: dry-mix process and wet-mix process. Both use compressed air to project concrete [5]. With the dry-mix process, water gets added to the concrete at the spraying nozzle, while with the wet-mix process the concrete is previously mixed with water. The two shotcrete methods have advantages and disadvantages, and the choice of which one to use for a given project depends on the characteristics of the project itself [21]. However the most widely used method is the wet-mix type [13]. One of its advantages over the dry-mix method is that the concrete pump suffers less wear, because the wet-mix is less abrasive than the dry-mix [21]. On the other hand, the equipment required for the wet-mix method is usually more expensive [21]. It can range from $60k to $115k, while the dry-mix method equipment is usually priced between $45k and $86k [21].

In any case, shotcreting always requires the following: a concrete pump, an air compressor, hoses and a nozzle [5]. Some applications may also require additional equipment, such as a

robotic arm or an accelerator dispenser [5]. Accelerators are substances which, when added to fresh concrete, speed up the hydration process [22]. They are usually added at the nozzle just before the concrete is sprayed [5]. For some applications the nozzle is manually controlled by the nozzle-man as shown in Figure 2.1. For heavier applications, a robotic arm, controlled remotely and in real-time by an operator, is used, as shown in Figure 2.2.

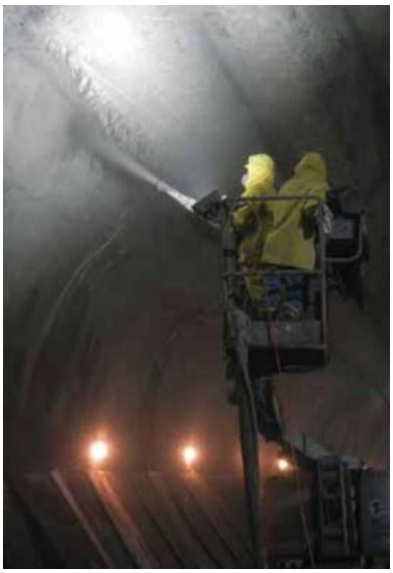


Figure 2.1: Shotcrete for rock support with manually controlled nozzle (taken form[23]).

Figure 2.2: Shotcrete where the control of the nozzle pose is done by an operator (taken from [24]).

One of the problems of having a worker physically handling the nozzle, is the exhaustion to which he is subjected to, given the weight of the equipment [25]. Using a human-controlled robotic arm allows for more time-efficient shotcreting, as the applied concrete mass flow can be much greater, compared to a worker manually handling the nozzle [12], while also allowing much greater reach. For tunnel and mine reinforcement, manually-controlled robotic arms are frequently used [4]. Depending on the used pump, the concrete flow may change over time, following a periodic nature. In such cases the nozzle may be made to rotate while spraying, so that the concrete is deposited more homogeneously.

The shotcrete process needs to provide quality concrete placement to guarantee structural integrity of the sprayed surface. For economical reasons, it's also important to be time-efficient and minimize rebound losses.

Three broad challenges can be identified for the shotcrete industry:

1. Quality assurance (assuring minimum layer thickness and homogeneity).
2. Time-efficiency.
3. Material-efficiency (reducing rebound and over-application[1] of concrete).

[1] Over-application means creating layers that are wider than intended [1], and represents a waste of both time and material.

To address quality assurance, one needs to further automate the shotcrete process, by using a robotic arm that performs a computer programmed trajectory, and uses real-time control to assess the thickness of the sprayed surface [13].

Time-efficiency can also be improved greatly by automating this process, because real-time quality control reduces the need for post-work repairs, that is, having to re-spray parts of the surface that did not meet quality standards. The rebound losses can also be minimized by using autonomous shotcrete robots, because reducing rebound can be accomplished by keeping the nozzle perpendicular to the surface and at a well-defined distance to the wall (usually 1 to 1.5m [1]), which is something the robotic arm can do with great precision if computer controlled. Other factors that affect rebound are not dependent on whether the robotic arm is human or computed controlled, such as the presence of reinforcement bars and parameters related to the concrete mix composition [5]. If the trajectory control is well done, it can also minimize the instances where concrete over-application occurs.

2.2. Automating the shotcrete process

The shotcrete process requires control of the spraying trajectory to ensure proper layer placement. Currently, the operator commands the nozzle pose remotely and in real-time, and evaluates layer thickness by visual observation [2]. The difficulty of this job is increased by the fact that shotcreting creates a particle-filled environment, which not only hinders observation, but can also be damaging to the operator's health [7]. It is therefore desirable to automate the shotcrete process, not only to ensure worker's safety but also to increase efficiency and quality [13]. This section focuses on shotcrete automation and is divided in 4 sub-parts. The first subpart gives an overview of what has been done to achieve fully-autonomous shotcrete systems. The second subpart focuses on environment perception. The third and fourth sub-parts focus, respectively, on trajectory generation and real-time layer thickness control.

2.2.1. Overview

Automating the shotcrete process involves building or adapting a certain set of equipment to successfully obtain a robot-based system that can assess its environment, generate a spraying trajectory, and control the nozzle. The process needs to ensure the placement of layers with a designated thickness, time-efficiency, and material efficiency. According to [11], automating this process has the potential to improve quality while reducing costs. While no robots with such autonomy have been found commercially [9], at least one company, Normet, has announced a fully-autonomous shotcrete system, to be launched in the future. The name of the system is SmartSpray ProPlus and it will be capable of computing, according to the authors, a "path for nozzle movements so that defined areas are sprayed automatically". It will also be capable of maintaining a certain nozzle-to-wall distance and a constant speed, to achieve desired layer thickness [14]. Furthermore, several researchers have implemented automatic shotcrete systems or solved a particular problem associated with this goal.

In [7], a shotcrete system was adapted to perform autonomously on a tunnel, using a laser scanner for environment perception. To estimate the volume of concrete to be placed, the difference in cross-sectional area between the pre-shotcrete surface and target surface was integrated along the tunnel length. The tunnel surface data was stored on a computer and, using a graphical model the authors had previously created, autonomous path generation and autonomous shotcreting was achieved. After testing and comparing with manual operations, the developed autonomous system revealed a smaller rebound, around 9%, while the manual operation resulted in 25% rebound. The speed of the process was also improved, however, no layer thickness control was done, which means that there was no assurance that the sprayed layers met the minimum requirements for structural integrity. This also means that the increased productivity might not actually be a significant improvement, because if re-spraying was needed, the whole operation would take longer. For these reasons, the system developed by the authors would most likely have to be adapted to include layer thickness control before it could be used in industry.

In [11], a system that could be used with manual, semi-autonomous or fully-autonomous control, was developed. The environment perception was achieved with a laser measurement device. In the semi-autonomous mode, the operator could control the trajectory and velocity of the nozzle, but the orientation and distance to the wall were controlled by a computer program. In the fully autonomous mode, both the path generation and control were done autonomously. However, this paper only allowed for a single path type to be generated, and did not explain the workings of the path generation algorithm implemented.

In [1], a Sika-Putzmeister machine was successfully adapted to perform autonomously by scanning the working surface with 3-D LiDAR (Light Detection and Ranging) and using a computer to generate and control the nozzle trajectory. The real-time layer thickness estimation, which was used to achieve real-time control, was done using a semi-empirical equation, which took into account factors such as concrete mix design, rebound, and the distance of the nozzle to the wall, among others.

In [9], a system was implemented that could generate 3-D models of the surface, plan trajectories, and perform motion tracking control. A scanning device was used to create 3-D models, which were then used to generate spraying trajectories. Motion control was performed by an algorithm based on dynamic PID (Proportional Derivative Integral). The adapted machine was tested on a simulated tunnel and the average error of the actual trajectory was about 50 mm when compared to the target trajectory. This test did not actually involve projecting concrete, which means it did not exactly replicate real shotcrete conditions.

In [2], a commercial shotcrete system (HPS3016C of China Railway Construction) was successfully adapted to operate autonomously on a tunnel. Similarly to [1], a laser scanner was used to obtain a 3-D model of the surface to be sprayed. The spraying time at different points was determined by the volume to be filled. After a region of the tunnel was sprayed, it was scanned again and, if the placed layer thickness was less than the desired thickness, the region was shotcreted again. This scan-spray-scan strategy meant that there was layer

thickness control, although not in real-time. An experiment was conducted, with the achieved average layer thickness error being less than what is typical obtained in manual operations.

2.2.2. Perception

In manual shotcrete applications the operator assesses the surrounding environment visually, so that he can properly place the nozzle relative to the wall, and estimate the actual layer thickness in real-time [2]. To completely replace the operator, and thus achieve full autonomy, one needs to use perception technology [7], [10], [2]. Scanning the surrounding environment is fundamental to build a 3-D model which can then be used to generate a spraying trajectory [9]. Furthermore, if real-time layer thickness control is desired, it might be done with environment perception as well, although this is made difficult by the dust filled environment caused by the shotcrete process [11]. A 3-D model of the surface to be sprayed can also be used to estimate the volume of concrete that will be placed, by comparing the volume enclosed by the actual surface (pre-shotcrete) and the target surface (post-shotcrete). However, the concrete volume that needs to be projected is larger than this volume, because some material is wasted due to rebound. The remainder of this section describes previous work regarding the different perception technologies used to automate the shotcrete process.

In [7], a laser measurement instrument was used to scan a tunnel. The goal of the authors was to create a fully autonomous shotcrete system. The scanning served two purposes: estimate the volume of concrete that had to be placed, and obtain data to generate a spraying trajectory. The data relative to the pre-shotcrete surface was stored as a spreadsheet, which included angle, distance, and coordinate projection in vertical and horizontal directions to create a spraying trajectory.

In [11] an infrared laser which could scan a section of 180° of a tunnel surface was used. The objective of the work was to obtain a shotcrete system that could be used in manual mode, semi-autonomous mode, and fully autonomous mode. The scanner data was relevant for the last two modes. In the semi-autonomous mode, a virtual plane is created, congruent to the scanned surface. This virtual plane is situated at a certain distance from the actual surface plane. Then, the nozzle is computer controlled so as to only move in this plane and perpendicular to it. This ensures that the designated nozzle-to-wall distance is always maintained, and that the orientation is optimal, so as to minimize rebound. The operator must still control the path (within the aforementioned plane) and the speed of the nozzle along that path. In the fully autonomous mode, the operator is completely replaced. In that mode, similarly to the semi-autonomous mode, the nozzle can only move in the virtual plane, with perpendicular orientation to it, and at a certain designated distance to the tunnel wall, however, the path generation and control are computer-controlled. The surfaces that need to be shotcreted are often quite heterogeneous, containing a somewhat random contour [2] as is shown in Figure 2.3. This happens because these tunnels are often formed by use of explosives [1], [2]. Because the virtual plane in which the nozzle is allowed to move is congruent to the tunnel surface, the nozzle pose adapts to the contour of the surface. This

means that the optimal distance and orientation is maintained regardless of the heterogeneity of the wall surface.

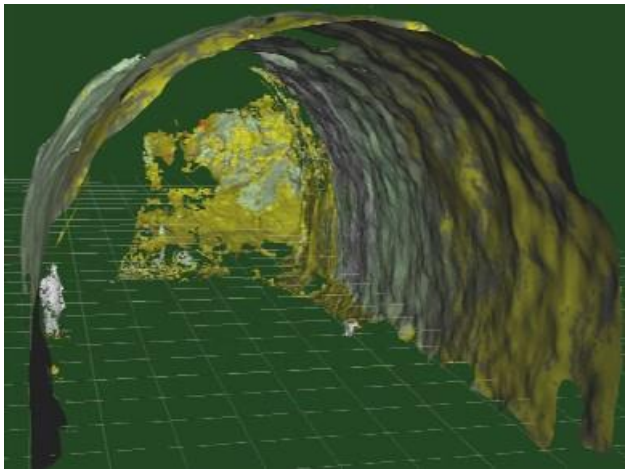


Figure 2.3: Three dimensional model of a tunnel surface (pre-shotcrete) (taken from [1]).

In [1], a LiDAR scanner was used to generate models of the pre and post-shotcrete surfaces in a tunnel environment. The goal of the developed work was to obtain a fully-automated shotcrete robot, and so, the purpose of the first scan was to provide the control system with tunnel surface data in order to generate a spraying trajectory. The second scan was used to assess the thickness of the applied concrete. This data was also meant to tune a semi-empirical real-time thickness estimator model which the authors developed.

With the goal of adapting a commercial shotcrete system to perform fully autonomously, [9] used a 2-D scanner mounted on the arm of the shotcrete machine (8 degree of freedom arm) as shown in Figure 2.4. The idea behind using a 2-D scanner to obtain a 3-D model, was that the scanner could be used to generate 3-D models if the instrument's pan and tilt were electronically controlled. This allowed the authors to avoid using 3-D scanners, which, as stated by the authors tend to be bulky and expensive. Other perception technologies, like binocular stereo cameras and total stations, were considered but rejected. The first one because it requires good lighting conditions and a clear view of the surroundings, and the second one because it only allows a limited number of measurements. After the environment was scanned and the point cloud was obtained, the non-surface point clouds which corresponded to walking construction workers and parked vehicles were removed. This was achieved using a clustering algorithm which segments the given data-points into groups. The cloud points which corresponded to the tunnel surface formed the largest cluster. The other clusters were then removed to obtain only the relevant cloud points.

Figure 2.4: The 2-D scanner mounted on the robotic arm used to generate 3-D models (taken from [9]).

In [2], a laser scanner was used to obtain a 3-D model of the surface to be sprayed. To estimate the volume of concrete to be placed, the pre-shotcrete and target surfaces were superimposed, as shown in Figure 2.5. Then, the 2.5-D growth method was used. This method starts with the division of the target surface model into a discrete grid, then each grid cell becomes the base of a prism. Each prism is enlarged outward until it reaches the pre-shotcrete surface. The summation of the volumes of the prisms results approximately in the volume of concrete to be placed.

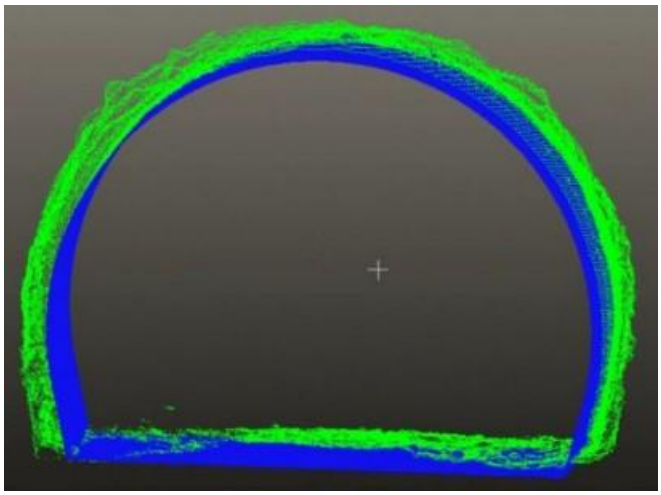


Figure 2.5: The green surface represents the actual surface before shotcrete application. The blue surface represents the goal surface (taken from [2]).

2.2.3. Trajectory generation

To achieve full autonomy, after a 3-D model is obtained, a nozzle path needs to be automatically created by a computer program. To form the trajectory, the computer program needs data regarding the configuration of the robotic manipulator, the 3-D model of the surface, and the desired nozzle-to-wall distance. The configuration of the robotic manipulator is essential to determine the direct and inverse kinematics.

The computation of the inverse kinematics of a manipulator allows to place the end tool on a desired pose. With the goal of creating a fully-autonomous shotcrete system, [11] started by introducing 3 constraints in order to reduce the number of degrees of freedom of the manipulator from 8 to 6. Then, to solve the inverse kinematics, the Newton-Raphson method

was used. The authors analysed the influence of different parameters such as concrete conveying capacity, spraying distance and nozzle moving velocity on layer thickness. However, the details of the method used to generate the trajectory were not revealed.

To generate spraying trajectories, [1] started by creating a 3-D model of the tunnel surface by use of a 3-D LiDAR. They solved the inverse kinematics using the decoupling technique, and then generated a trajectory by defining several target poses for the nozzle. Each target pose was obtained by a computer program, by computing the surface model normal vectors at different locations, although the authors did not provide further details about this process. The localization of the different targets formed a squared-zigzag trajectory, as shown on Figure 2.6. Then, the spraying was executed from the bottom of the tunnel to the top, such that the nozzle started with horizontal orientation and ended with a vertical orientation. To create the target poses and generate the spray path, several parameters needed to be given to the computer program, such as the initial target position, the width of each horizontal line (limited by the manipulator dimensions), the distance between the horizontal lines of the trajectory, the number of targets for each vertical line, and the number of targets for each horizontal line.

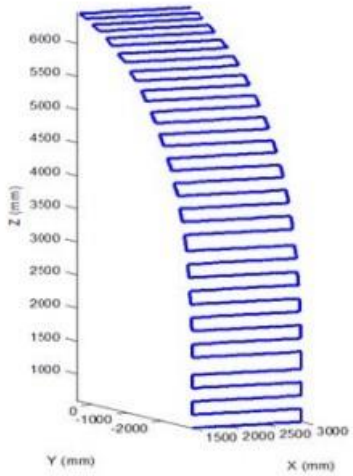
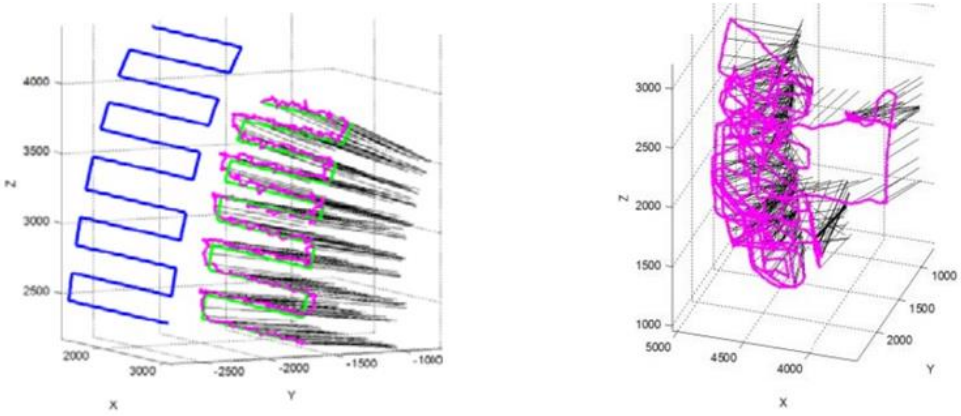


Figure 2.6: Squared-zigzag spray path. The nozzle starts at the bottom with a horizontal orientation and ends at the top with a vertical orientation (taken from [1]).

Only 2 targets are needed to form a straight path, however, in this case, if only 2 targets were used to create each horizontal and vertical line, then the orientation of the nozzle would not adjust to the contour of the tunnel wall as it traverses along the path. Using several targets for each path line allows the nozzle to maintain proper distance and orientation relative to the surface. The authors compared this system with a human-controlled system, obtaining the trajectories shown in Figure 2.7.



a) **b)**

Figure 2.7: a) Computer generated path. b) A spray path obtained manually (taken from [1]).

When shotcrete is done with human control, the operator tends to use mostly the last joint, which is a revolution one, because it is easier than controlling several joints simultaneously resulting in a semi-spherical geometry of the spraying [1]. Given that maintaining perpendicular orientation of the nozzle relative to the wall is important to reduce rebound [1], it can be concluded that relaying on the rotation of the last joint to spray will result in higher rebound, given that the nozzle will be mostly at non-perpendicular orientations.

In [9], a point cloud of a tunnel surface was obtained using a laser scanner. To guarantee that the nozzle remained perpendicular to the tunnel surface during the shotcrete process, the authors started by obtaining the surface normal vectors, which was done by dividing the point cloud in several groups, using cut planes, as shown on Figure 2.8. To generate positions, the cloud points close to a cut plane were paired with the closest points belonging to the adjacent group of points, as shown on Figure 2.9.

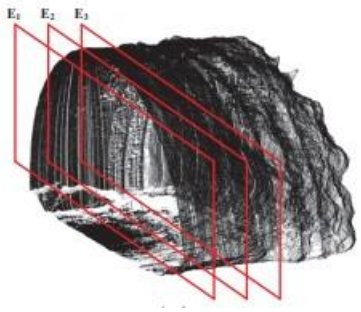


Figure 2.8: Tunnel point cloud and cut-planes (taken from [9]).

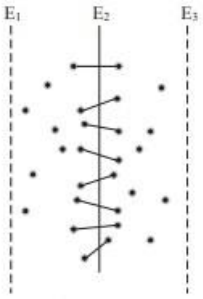


Figure 2.9: Each point close to a cut plane is paired with the closest point from the adjacent group which is also close to the same cut plane (taken from [9]).

The interception of the line segment, connecting every pair of matched points and the cut plane, was used to generate positions for a robotic path. Given the concave shape of the tunnel, the shape of the path resembles a parabola. To obtain the orientations, the surface normal directions were obtained by applying plane fitting to the point cloud.

2.2.4. Control

To create concrete layers with a desired thickness, one needs to control the TCP (tool centre point) speed and/or the concrete flow. If real-time control cannot be achieved, a scan-spray-scan approach can be used where after the spraying is finished the walls are scanned again. Then, if needed, more spraying is done. This strategy allows to better cope with the particles in the environment, but it's also more time-costly.

In [1], a real-time thickness estimator was developed which consisted of a semi-empirical function. After the spraying process, the authors scanned the surface again to tune the estimator. The function is meant to be used in real-time, and its results are used to define the manipulator's speed accordingly. This strategy, if successful, allows the placement of layers with the proper thickness, making it unnecessary to respray after the first spraying round. The real-time layer estimator is a function which considers several factors, such as the concrete flow given by the machine's pump, the mix design, the rebound, the nozzle orientation, and the wall position, among others. However, it does not take into account the actual placed layer thickness, so a post-shotcrete scan is still needed to ensure quality assessment.

3. Path Generation and Execution for Shotcrete

The shotcrete machine must automatically generate and follow a path, for the process to be fully automated. The robot used in this project is an IRB 6700 with 2.45 m reach and 300 kg of maximum payload. The choice of the robot and the platform on which it stands, were done by other members of the RoboShot@FRC project, taking into consideration, among others, the test tunnel being used and the payload needed for the shotcrete operation. The simulation version was equipped with an ABB Omnicore controller and RobotWare 7.5.2. The development and testing environment includes a tunnel mesh which was obtained by LiDAR scanning a real tunnel, railway tracks, and a platform where the robot is placed, as shown in Figure 3.1. The developed programs allow generating spray trajectories and executing them in RobotStudio 2022.3. The ability to generate several trajectories and then have the robot move along the tracks to execute the subsequent trajectories was also implemented. This means the tracks used in RobotStudio function as a linear conveyor, although the motion over the conveyor is to take place between projections.

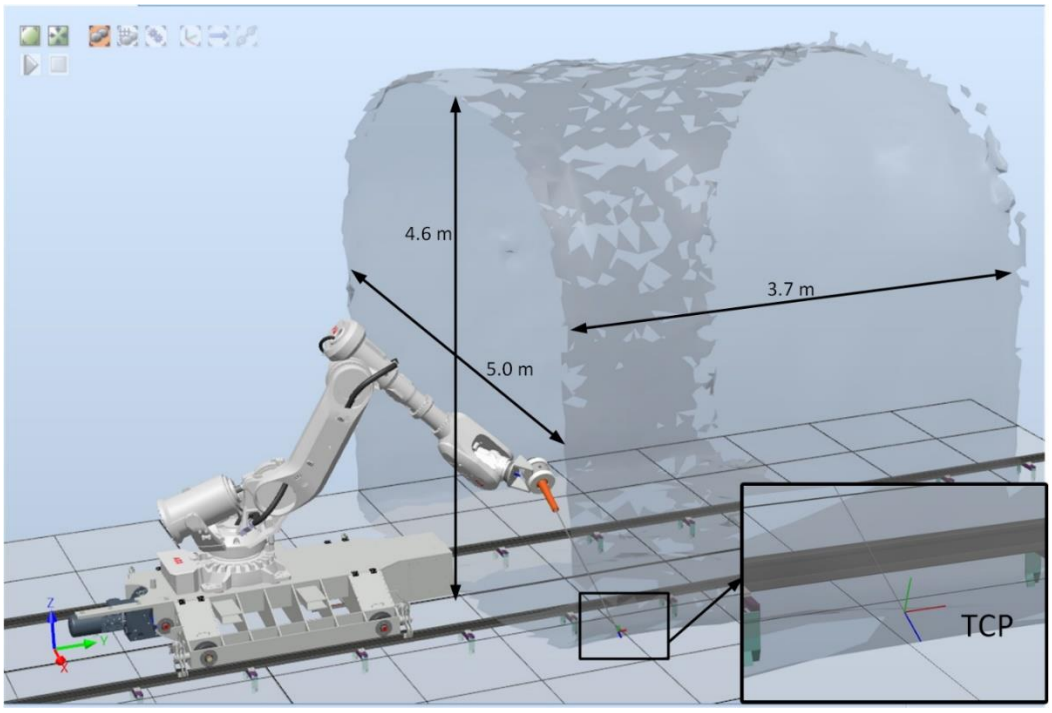


Figure 3.1: Main setup used in RobotStudio showing the mesh dimensions. The TCP z direction (in blue) corresponds to the projection attack direction.

This chapter describes how the path generation and execution works, which follows the pipeline summarized in Figure 3.2. It starts by, in 1), generating a list of cartesian poses given the mesh file, the robot model (including the projection head) and its relative pose, and the specified path characteristics. In 2), depending on the type of path being generated, the path is adapted to include circular motion along the path. Next, in 3), the path is adjusted to account for possible unreachable poses, due to the surface variations of the tunnel,

followed by further adjustments, in 4), an algorithm is used to avoid collisions between the robot, the tracks and the base structure. In 5) a search algorithm is employed to guarantee the path is smooth and along the surface between consecutive targets. Finally, in 6), the trajectory is executed by the robot by having a TCP/IP-based (Transmission Control Protocol/Internet Protocol) communication between the control program and the robot controller.

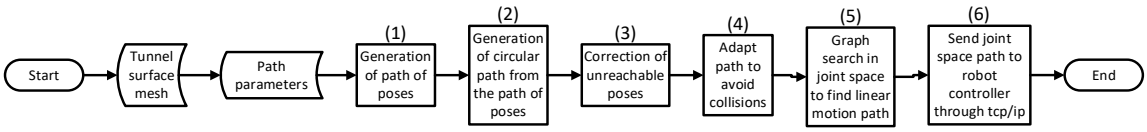


Figure 3.2: Pipeline of the path generation algorithm.

The development of the generation and control software was done in Python[2] 3.10.9 using Vedo[3] 2023.4.6+dev5, which allows creating and manipulating point clouds and meshes. RoboDK[4] 5.5.2 and RoboDK's Python module[5] 5.6.0 were also used to compute inverse and forward kinematics, as well as to evaluate the reachability of the motions. Development was done using Visual Studio Code[6] 1.75.0.

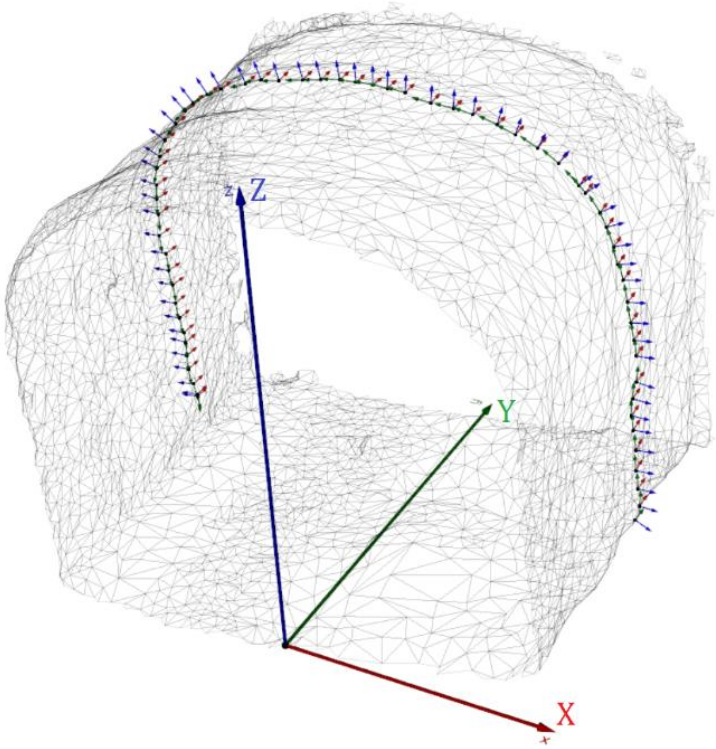


Figure 3.3: The z axis of each target is normal to the mesh.

[2] https://www.python.org/
[3] https://vedo.embl.es/
[4] https://robodk.com/
[5] https://robodk.com/doc/en/PythonAPI/index.html
[6] https://code.visualstudio.com/

The TCP (Tool Centre point) is located on the axis that passes through the nozzle, with a displacement from the nozzle tip (away from the robot) equal to the desired nozzle-to-wall concrete projection distance as shown in Figure 3.1. Therefore, the cartesian path for the TCP to follow must be composed of targets whose position lie on the tunnel surface, as shown in Figure 3.3. As mentioned previously, the nozzle attack direction should be normal to the mesh surface to minimize rebound. This restriction is only relaxed when reaching a particular target requires changing its orientation (as detailed in Section 3.4), or if it would lead to a collision (as detailed in Section 3.5). Figure 3.3 shows the mesh of the tunnel used for most of the tests and results presented in the remainder of this chapter, and the World Reference Frame (WRF) associated with it.

The developed algorithm allows generating 4 types of paths, namely straight, straight with circles, squared-zigzag and squared-zigzag with circles, as shown in Figure 3.4. In this figure, the arrows indicate the direction of motion of the TCP (the orientation of each pose is not depicted to improve clarity). The squared-zigzag path type is inspired on the path developed in [1], which resembles a path typically performed by the operators, while the straight type path is a simpler path type amounting to a squared-zigzag path where the distance of horizontal lines is zero. The circular versions of these path types were developed because it was a requirement to have circular trajectories. This requirement has the aim of creating more homogenous concrete layers, given the periodic nature of the concrete flow.

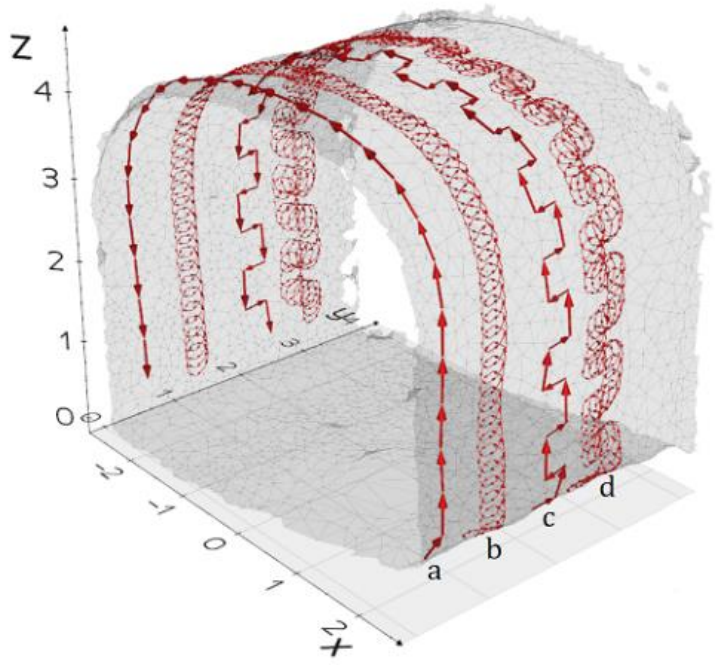


Figure 3.4: Different types of paths that can be generated. From left to right: a) straight path; b) straight path with circles; c) squared-zigzag path; and d) squared-zigzag with circles.

To form a straight path with circles, a straight path is first created and, to form a squared-zigzag with circles path, a squared-zigzag path is created first as well. The algorithm for

creating the underlying path types (straight and squared-zigzag) is the same, with the only difference being its input arguments.

The algorithm to form an underlying path is divided in two parts: first a path of positions is created (ordered list of 3-D points); then, an orientation (rotation matrix) is associated to each position to obtain an actual cartesian path. The mesh shown in Figure 3.3 and Figure 3.4 was obtained by other researchers within the RoboShot@FRC, and resulted from LiDAR scanning of a real tunnel [16]. To remove isolated faces, reduce noise and improve resolution (homogeneity), the mesh was pre-processed. Operations of decimation, subdivision and a Laplacian filter were used to obtain the final mesh shown in Figure 3.5, used as input in the remainder of the document. To test the path generation algorithm, both the original and processed meshes were used, resulting in similar paths. However, for shotcrete simulation only the processed mesh was used, to guarantee a more uniform simulation across the surface tunnel, as explained later in more detail.

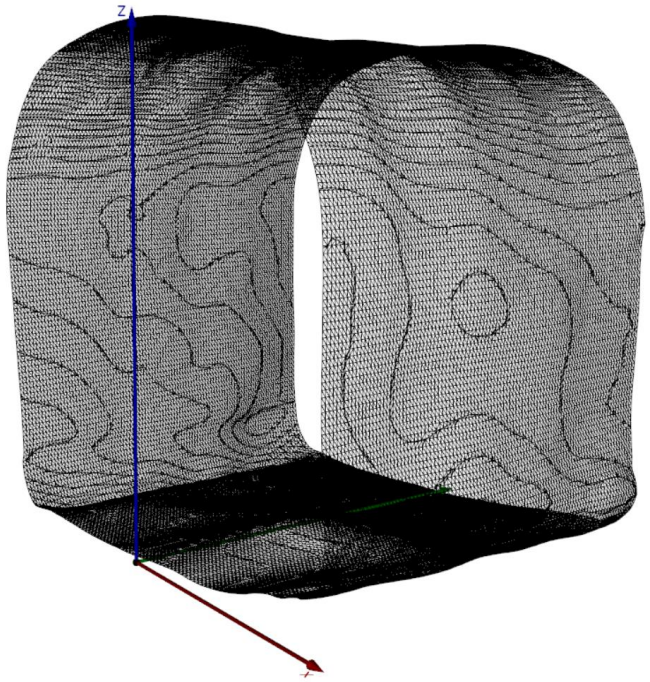


Figure 3.5: Mesh after pre-processing. The average area of its faces is 8.9 cm^2 with standard deviation of 3.6 cm^2 . Given it was obtained by scanning a real tunnel, its surface is irregular, containing visible deformations.

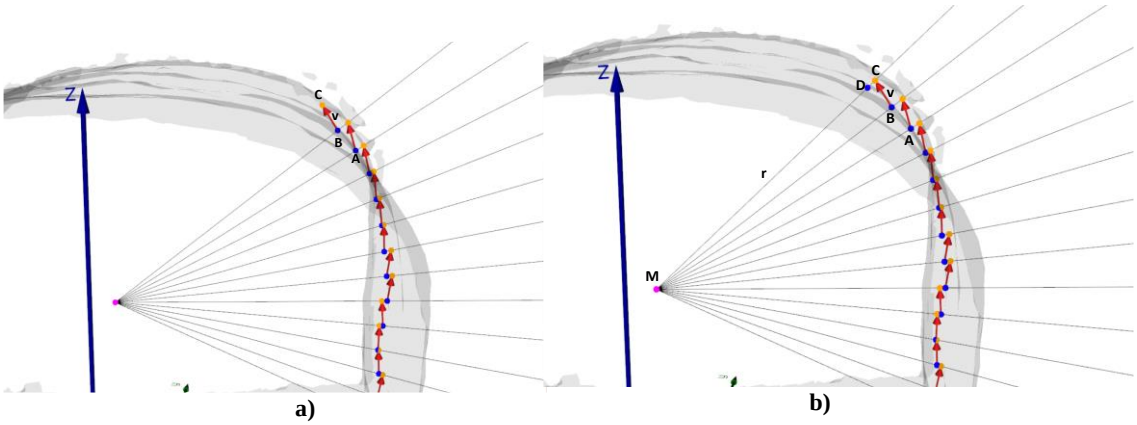
3.1. Obtaining an ordered list of positions to create an underlying path

The algorithm to obtain the path of positions (ordered list of 3-D points) receives 4 arguments for creating an underlying path (either squared-zigzag or straight):

1. Mesh.
2. Initial point.
3. Vertical vector.
4. Horizontal vector.

The mesh surface is the object on top of which the path will be created, so it is a fundamental input. The initial point corresponds to the first point of the path, while the vertical and horizontal vectors are the guiding vectors that are used to build the path. At each iteration, they allow creating the next point from the previous one, starting from the initial point. However, because the mesh surface is curved, these vectors need to be updated as the path is being created, so that they follow its curvature.

Figure 3.6 shows a snapshot of one iteration of the positions path generation process. On the left, points B and A are, respectively, the last entry and the penultimate entry of the ordered positions list (positions path); v is a vector, defined by $v = B - A$; C is the aproximation of the next positions list entry, and is defined by $C = B + v$ (last list entry plus a vector). On the right, D is the projection of C onto the mesh which is done by obtaining the interception of the ray **r** with the mesh. The ray **r** starts at point M and passes through C. In the first iteration of the algorithm, vector v is equal to the third argument of the algorithm, the vertical vector. At every iteration, it is updated as the difference between the last and penultimate positions list entries, as exemplified in Figure 3.6. To make sure the ray always intercepts the mesh, it is important that the input mesh does not have any holes on its surface.



a) b)

Figure 3.6: a) Every orange point is projected to the mesh resulting in a corresponding blue point. The blue points are the ones that form the ordered positions list. b) Point D is created by the interception of ray r (defined by M and C) with the mesh.

Point M is the section centroid and is obtained by intercepting the mesh with a plane, as shown in Figure 3.7, and computing the average value of the interception points. The plane is normal to the y axis and passes through the initial point (one of the input arguments of the algorithm).

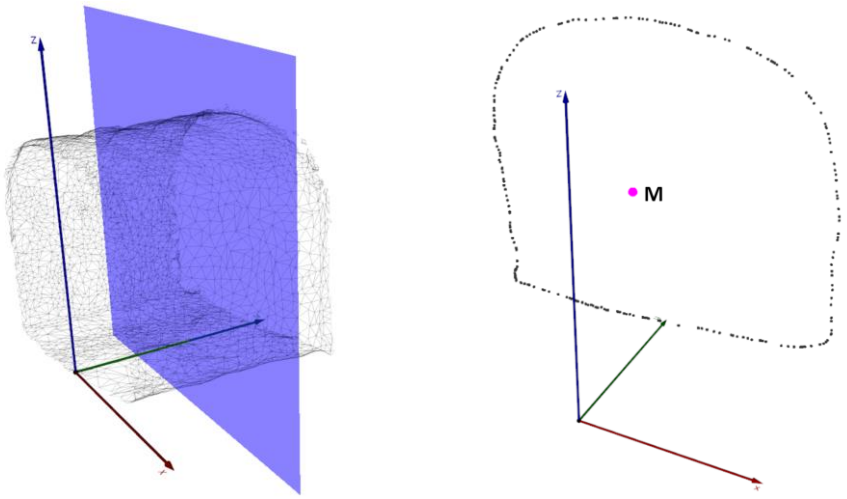


Figure 3.7: a) The interception of a plane with the mesh results in a set of points shown in b, with centroid M.

When the horizontal vector (given as an input argument) has at least one non-zero component, a squared-zigzag-path is created in a similar way to what was just explained. The only difference is that there are two vectors that need to be updated and used when generating new points, one after the other. Figure 3.8 shows both path types, whereas the paths are represented only by points since, at this stage of the algorithm, an orientation has not yet been assigned to each position.

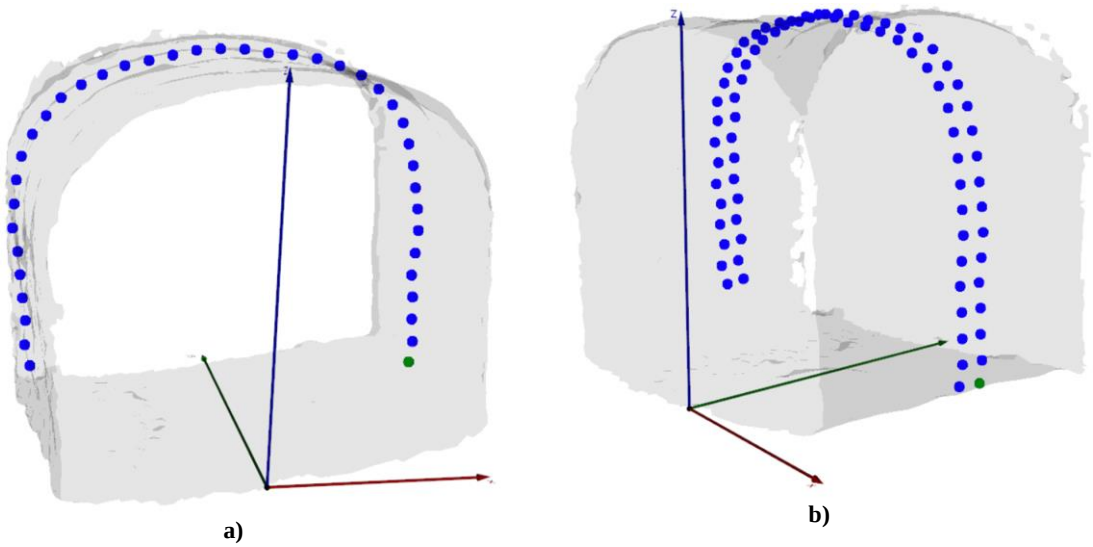


Figure 3.8: Positions path (with initial point shown in green) for the straight-type case and the squared-zigzag case, in a) and b), respectively.

3.2. Assigning an orientation to each position to obtain a cartesian path

To obtain a cartesian path, one needs to assign an orientation to each position of the ordered positions list. As mentioned previously, the nozzle should point perpendicularly to the surface to minimize rebound. This means that each target should have its z axis normal to the surface. This restriction alone is not enough to determine orientations, because there are infinite orientations that a target may have, that result in its z axis being perpendicular to a surface. This problem is solved by arbitrating a final orientation for each target resulting in an optimization problem summarized by:

$$\max(\hat{x} \cdot \hat{g}) \; constrained \; to \; \hat{x} \cdot \hat{z} = 0 \; and \; \hat{z} \; normal \; to \; the \; surface$$

Where $\hat{x}$ is x axis unit vector; $\hat{g}$ is a user specified vector that represents a reference direction and $\hat{z}$ is the orientation matrix's z axis unit vector. The remaining orientation matrix column, y axis unit vector is determined by obtaining the other two:

$$\hat{y} = \hat{z} \times \hat{x}$$

The algorithm for assigning an orientation to each position takes four input arguments:

1. Ordered positions list.
2. Reference direction.
3. Radius for sampling.
4. Mesh.

For every position (all positions lie on the mesh surface), its corresponding orientation z unit vector is computed as the average of the mesh normal directions at all mesh vertices within the specified sampling radius (given as the third input argument). If the radius is so small, that no mesh vertices are found within it, then the normal direction is obtained by computing the average of the three mesh vertices closest to the position under consideration. After obtaining the z axis for a given target, one needs to compute its x axis direction, which is done by solving the previously explained optimization problem. The solution is obtained with an iterative algorithm that creates an initial x axis unit vector, $\hat{x}_1$, which is perpendicular to the corresponding z unit vector, $\hat{z}$ (previously computed). Vector $\hat{x}_1$ is first rotated around $\hat{z}$ with an angular step θ, resulting in $\hat{x}_2$ ($\hat{x}_1, \hat{x}_2$ and, in general, $\hat{x}_i$ are always perpendicular to $\hat{z}$). If $\hat{x}_2$ is further (in terms of angular distance) from the specified reference direction, $\hat{g}$, on the next iterations the angular step becomes $-\theta$. The angular distance between $\hat{x}_i$ and $\hat{g}$ is computed using the dot product between them. At every subsequent iteration, $\hat{x}_i$ is obtained by rotating $\hat{x}_{i-1}$ by an angular step (of either θ or $-\theta$) around $\hat{z}$. Once the dot product between $\hat{g}$ and $\hat{x}_i$ is smaller than the dot product between $\hat{g}$ and $\hat{x}_{i-1}$, the algorithm stops and establishes $\hat{x}_{i-1}$as the x unit vector of the corresponding target. The rotation of $\hat{x}_i$ around $\hat{z}$ is done using Rodrigues' Formula [26] . This unit vector is not as close as possible to the reference direction due to using an angular step that is not as small as possible (a trade-off is done because smaller angular steps make the algorithm more precise, but also slower).

To clarify, Figure 3.9 shows Figure 3.8's $\hat{z}$, $\hat{g}$ and $\hat{x}_1$ vectors, several instances of $\hat{x}_i$, and the final pose for a particular target whose position is represented by the point in magenta.

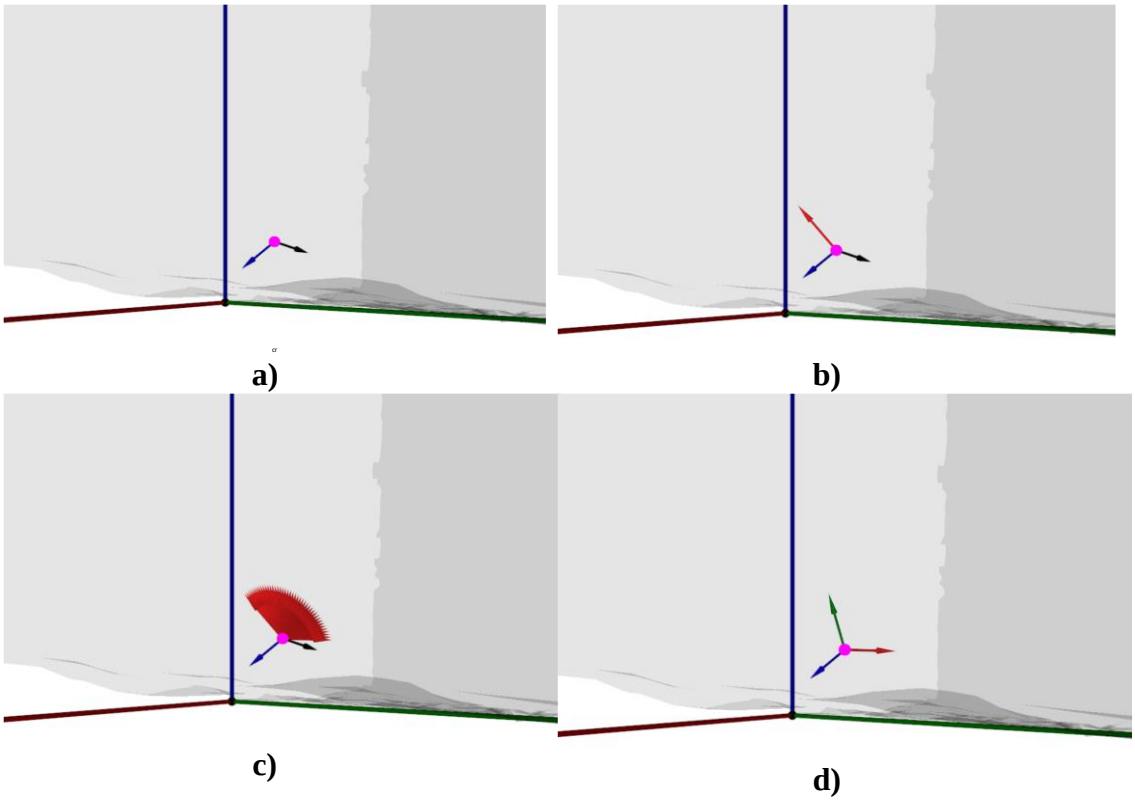


Figure 3.9: Computing the target orientation from its position and the tunnel surface: a) shows the position in magenta, the target z axis, $\hat{z}$, in blue, and the reference direction for the x axis, $\hat{g}$, in black; b) the initial x axis direction, $\hat{x}_1$, was added in red; c) shows the evolution of the $\hat{x}_i$ axis in red over several iterations, computed by rotating $\hat{x}_{i-1}$ by an angular step of 3° around $\hat{z}$; d) shows the final cartesian pose, including the y axis direction in green.

While the reference direction for the first target is an input argument, the reference direction for each subsequent target is the x axis direction obtained for the target created immediately before. This is done to minimize the joint distance the robot must "travel" to go from one target to the next.

This algorithm is applied to every position of the ordered positions list, obtaining a list of targets where each target represents the TCP for a given location, that is, a cartesian path. This information is stored using an ordered list of 3-D transformation matrices in homogeneous coordinates (hereon denoted as homogeneous transformation matrices).

3.3. Creating circular paths from underlying paths

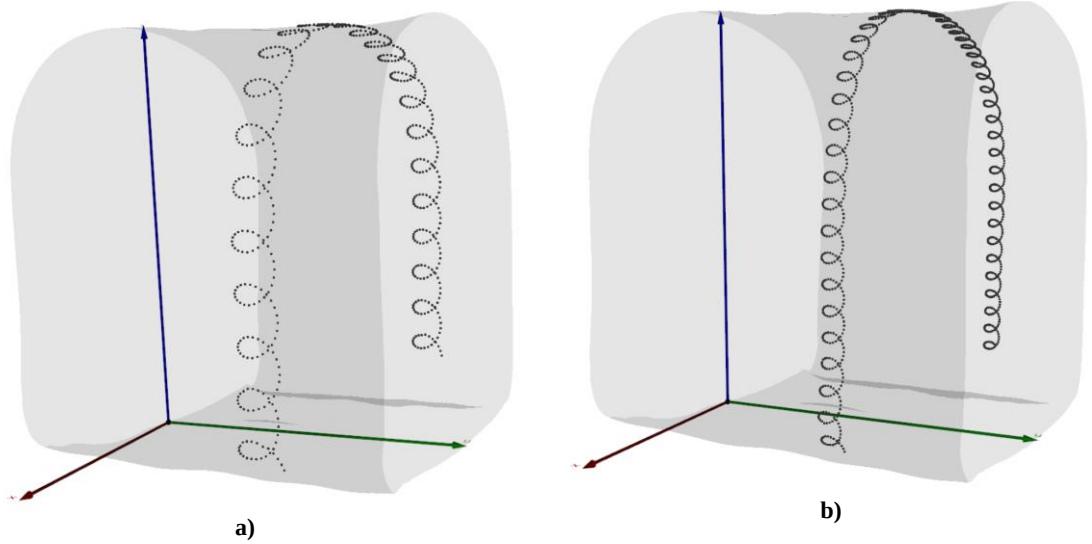
As stated previously, current shotcrete equipment, which is remotely controlled by an operator, typically includes a continuous rotation of the projection head. This rotation helps reduce the impact of the periodic variation of the concrete flow on the projection result, by

spreading the projection over a large area, and by working as a low pass filter for the flow. For the robot-based projection to be able to have a similar behaviour, but without compromising the precision of the system regarding the direction projection, it was decided to allow to superimpose a circular motion to the path generated previously. These new paths are denoted here as a straight path with circles and as squared-zigzag path with circles.

The algorithm to create circular paths from underlying paths is the same for both types of underlying paths, straight and squared-zigzag. Its input arguments are:

1. Underlying path (ordered list of homogeneous transformation matrices with position and rotation).
2. Radius (the radius of the circles).
3. Spatial frequency (number of circles per linear meter of the underlying path).
4. Number of points to approximate the circles (approximated by N-point polygons).

The first input is an ordered list of homogeneous transformation matrices with position and rotation and is the output of the algorithm explained in section 3.2. The second input is the desired radius for the circles; the third input determines the number of circles created per linear distance along the mesh surface and the fourth input is the number of targets used to approximate the circles. Figure 3.10 gives examples of two straight paths with circles with different parameters.



a) **b)**

Figure 3.10: a) Straight path with circles: radius of 0.2 m; spatial frequency: 2.0 circles/m; number of points per circle:24; b) Straight path with circles: radius of 0.1 m; spatial frequency: 4.0 circles/m; number of points per circle:24. Points represent positions of targets.

Similarly, to the algorithm used to obtain the underlying paths, the algorithm for creating circular paths first creates an ordered list of positions, and then assigns an orientation to each position. While the algorithm to assign an orientation to each position is the same as

explained in Section 3.2, the algorithm to create an ordered list of positions is quite different, as is described below.

The underlying paths work as the scaffolding for the creation of the circular paths. For every set of consecutive positions of the underlying path, $\{u_j, u_{j+1}\}$, a new set of points is created by the following system of equations:

$$\begin{cases} q_t = u_j + v \cdot l \cdot k + \varepsilon \\ k \in \left[0, \quad \frac{|u_{j+1} - u_j|}{l} - 1\right] \cap \mathbb{Z} \\ v = u_{j+1} - u_j \\ \varepsilon = r \cdot \cos(f \cdot t) \cdot \hat{d}_j^x + r \cdot \sin(f \cdot t) \cdot \hat{d}_j^y \end{cases}$$

Where q_t denotes the t^{th} 3-D position of the circular path; v is a vector defined by the set of points $\{u_j, u_{j+1}\}$ of the underlying path; l is a scalar that corresponds to a linear step; and k is a an integer that, along with l, gives v an appropriate magnitude. The last equation parameterizes a circle with radius r and frequency f, using the unit vectors $\hat{d}_j^x$ and $\hat{d}_j^y$, the x and y axes unit vectors of the j^{th} pose of the underlying pose path, respectively. The equations above are applied in a piecewise manner to every pair of consecutive positions of the underlying path, with $t \in \mathbb{Z}$ serving as a parameter that starts at zero and increments one unit at every iteration (every time a new position is created). The position that is added to the circular path is not actually q_t, but rather the projection of q_t onto the mesh surface, similarly to the algorithm for the generation of the underlying path. To clarify, the j^{th} pose is a homogeneous transformation matrix:

$$H_j = \begin{bmatrix} \hat{d}_j^x & \hat{d}_j^y & \hat{d}_j^z & u_j \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The radius, r, of the circle parameterized above, is an input argument of the algorithm, while the frequency (f) of the trigonometric functions and l are calculated from the spatial frequency and number of points (both are input arguments) as follows:

$$\begin{cases} f = \frac{2\pi}{N} \ [radians] \\ l = \frac{|u_{j+1} - u_j|}{|u_{j+1} - u_j| \cdot \lambda \cdot N} = \frac{1}{\lambda \cdot N} \ [meter] \end{cases}$$

Where λ and N are, respectively, the spatial frequency and the number of points to approximate each circle, both input arguments of the algorithm. One way to picture the circular (positions) path algorithm, is to imagine a vector with magnitude equal to the radius (r) positioned at the underlying path's initial position in the first iteration. Then, at every subsequent iteration, its tail moves a linear distance of $l \cdot k$ meters along the underlying path, while the vector itself rotates f radians around the surface normal, as shown in Figure 3.11.

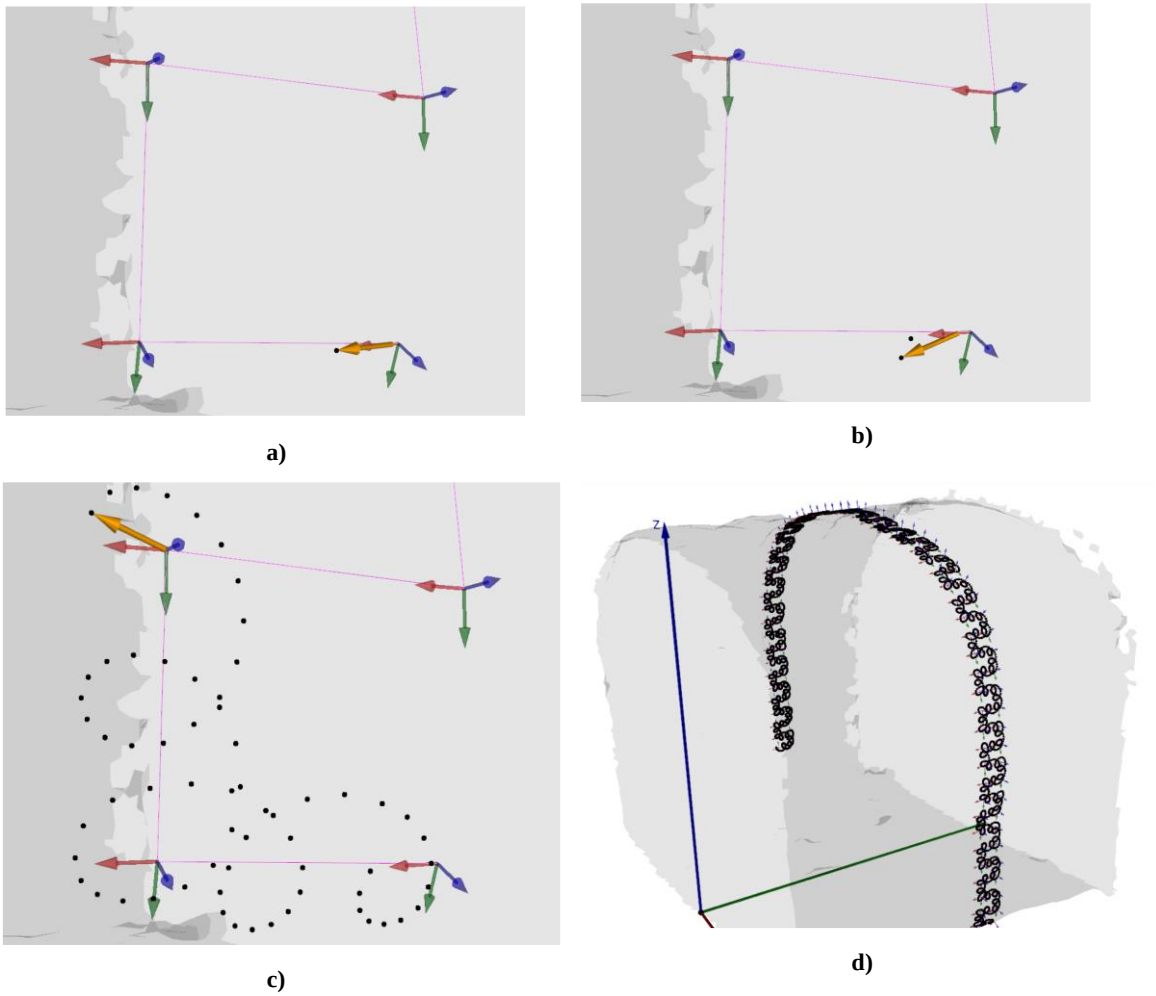


Figure 3.11: a) First four targets of the underlying path, with magenta lines connecting them for illustration purposes, and the first position of the circular path (black point); b) Second position of the circular path (orange vector tail moves a distance of $l \cdot k$ linearly along the magenta line, and rotates around the surface normal f radians); c) The two first linear segments of the underlying path are traversed by the orange vector; d) Full circular (positions) path and the underlying path's targets are shown.

As mentioned previously, to obtain a circular cartesian path, an orientation is assigned to every position generated by the circular path algorithm using the algorithm described in Section 3.2, obtaining a final path as shown in Figure 3.12 a). This squared-zigzag with circles path has the following characteristics:

- Initial position of $[2.48 \quad 2.5 \quad 0.50]$ meters relative to the WRF (parameter of its underlying path).
- Vertical distance of 0.5 meters (parameter of its underlying path).
- Horizontal distance of 0.5 meters (parameter of its underlying path).
- Radius of the circles equal to 0.14 meters.
- Circles approximated by 16-point polygons.
- Spatial frequency of 8 circles per linear meter.

The path shown in Figure 3.12 b) is a straight path with circles, which has the same characteristics of the path shown in Figure 3.12 a), except the horizontal distance which is equal to 0 meters.

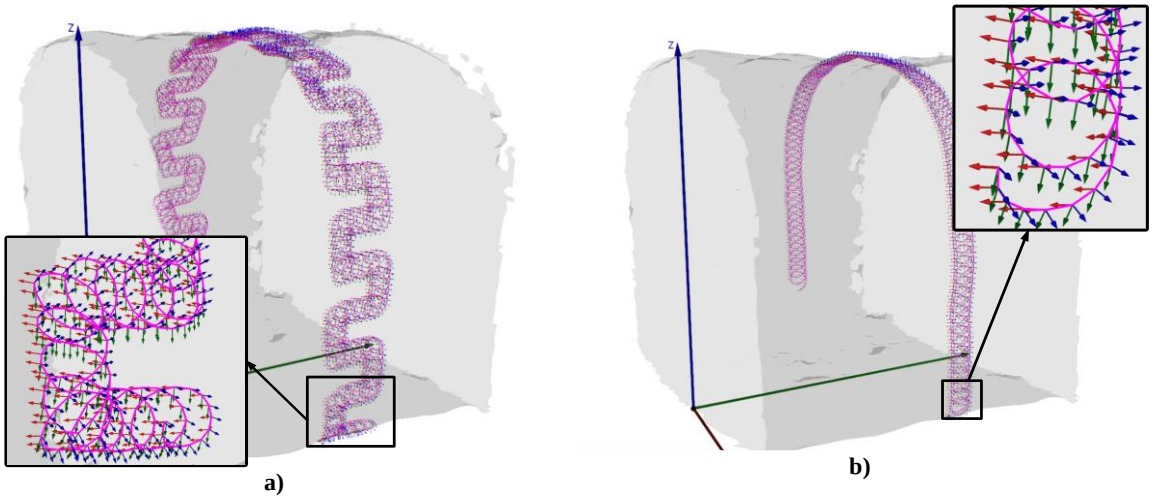


Figure 3.12: Targets shown with consecutive targets positions connected with magenta lines: a) squared-zigzag path with circles, including zoomed view; b) straight with circles path type, including zoomed view.

3.4. Correction of unreachable poses

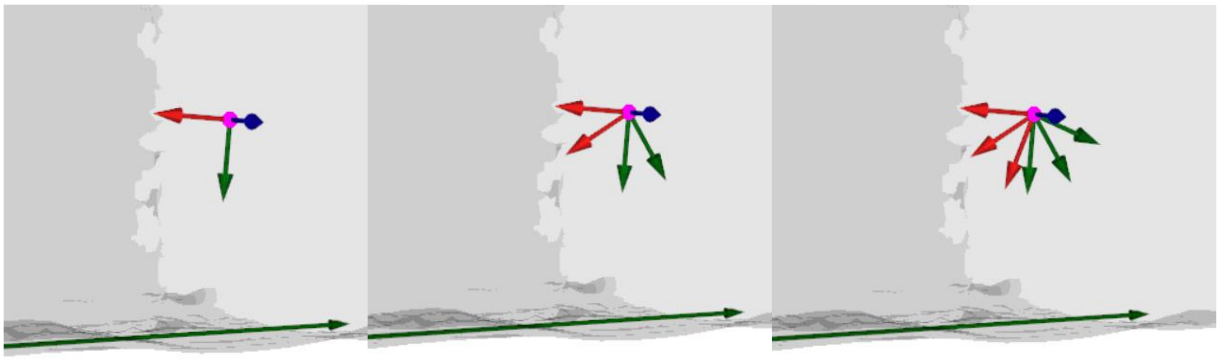
The algorithms explained in Section 3.1, Section 3.2 and Section 3.3 allow generating a cartesian path that best adjusts to the given mesh, with the given parameters, but does not check whether the robot being used is able to perform that path. This section explains how the path obtained using the algorithms described earlier, is adjusted so that all targets are reachable for the given robot model.

The algorithm for correction of unreachable poses iterates over every pose of the given path, taking into account the target pose, the robot model being used and the robot base pose in the WRF. When a given target is unreachable, i.e., the robot cannot achieve a certain pose, a search is done to find a reachable target pose with the same position but with different orientation. Whenever a target pose is not reachable, the search is first done only in roll space, which means rotating the target TCP pose around its z axis, with a certain angular step, as shown in Figure 3.13. This is the first adjustment to test, because it allows respecting the perpendicularity to the surface restriction to minimize rebound. If changing the roll is not enough to obtain a reachable pose, a search is done in pitch and yaw (rotations around the y and x TCP axis, respectively). To know if the targets are reachable, the RoboDK's Python module was used, specifically the SolveIK method.

The algorithm for correction of unreachable poses takes as inputs arguments the following:

1. Robot model with nozzle TCP and robot location.
2. Cartesian path (ordered list of homogeneous transformation matrices).
3. Angular step for the search in roll, pitch and yaw.

The output of the algorithm is a cartesian path where each target has the same position as in the input path, but the orientations are such that all targets are reachable. For simplicity's sake, and to make the search faster, the search is only done in terms of the orientations of the targets. If the position of a target is such that the robot cannot reach it, regardless of its orientation, then that information is returned, and the algorithm continues. In this situation, the generated path is not feasible with the given input arguments but, nonetheless, can be used for shotcrete simulation (for instance, it could be feasible with a different robot model, or with a different robot location).



a) **b)** **c)**

Figure 3.13: Search in roll space. a) Original unreachable pose; b) Rotation of $\pi/10$ radians around target's z unit vector (blue arrow); c) Rotation of $\pi/5$ around z unit vector to obtain feasible pose.

3.5. Collision avoidance

As shown previously, the robot sits on a platform which is mounted on rails. In the real application no obstacles are expected to be in the vicinity of the robot, but there could be collisions with the platform or with the rails as exemplified in Figure 3.14. This section explains the algorithm developed to prevent these collisions. Given that the platform and the rails geometry is known beforehand, rather than detecting the collisions in real-time, they are anticipated and avoided altogether by adjusting the cartesian path.

Given it is assumed that the only possible collisions are with the tracks or the car, and to make the algorithm faster, not all targets are checked for collisions. Instead of checking every pose, only those with a position z value below a certain threshold (denoted as the test threshold) are tested for collisions.

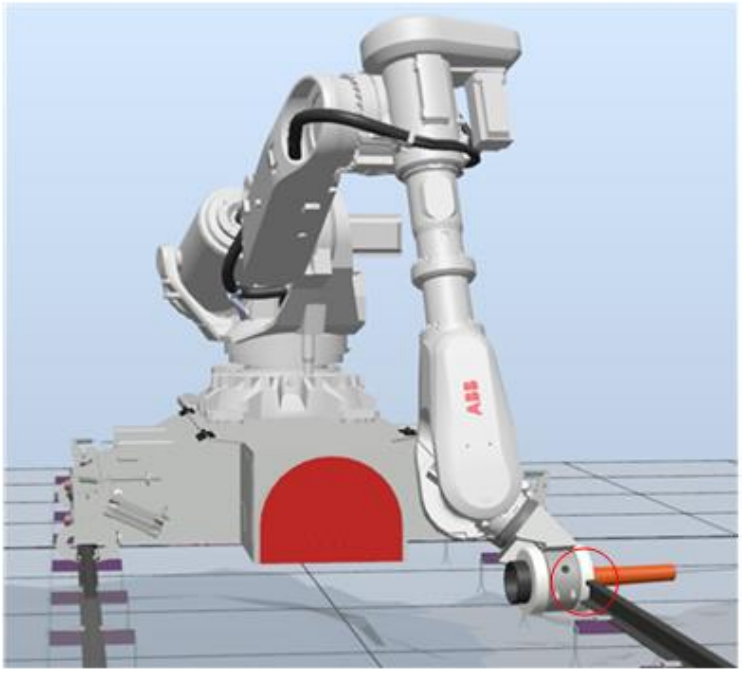


Figure 3.14: Collisions between the robot and the rails and/or car, such as the one shown here, need to be avoided.

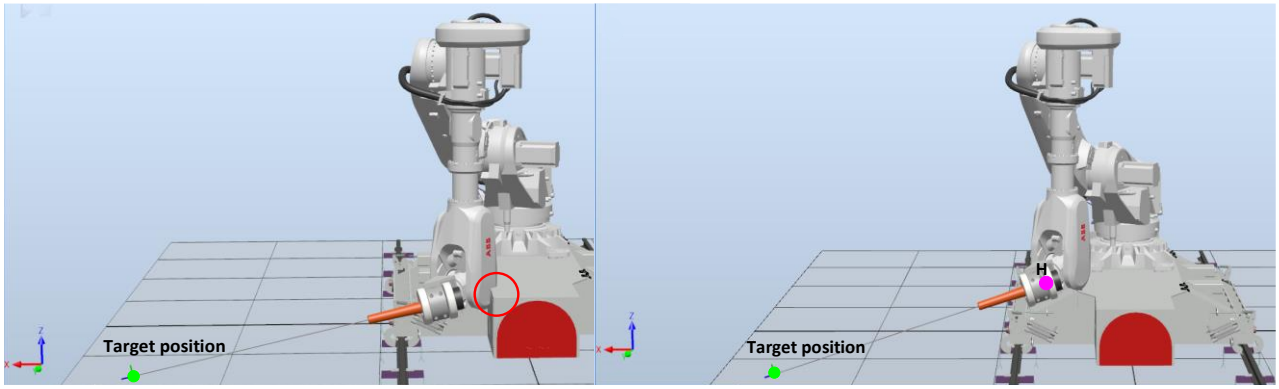
The algorithm to avoid collisions has the following input arguments:

1. Robot model with nozzle TCP and robot location.
2. Cartesian path (list of ordered homogeneous transformation matrices).
3. Threshold to test for collisions.
4. Threshold to signal a collision.

For each generated target and configuration, the robot fist must assume a given position. Given the position of the robot and the type of paths that are generated, the fist is the robot part expected to reach an overall lower position z value. Therefore, the collision avoidance algorithm focuses on guaranteeing that the fist never assumes a position too close to the ground level. If a position's z value (height, in the WRF) is bellow a given threshold (fourth input argument), it is considered to generate a collision and, in that situation, the perpendicularity constraint of the TCP orientation is relaxed, and recomputed as

$$\hat{z} = \frac{target_{pos} - H}{\left| target_{pos} - H \right|}$$

Where $\hat{z}$ denotes the recomputed z axis unit vector of the TCP orientation for a given cartesian target, whose original pose was such that the fist endpoint was considered to be in a collision; $target_{pos}$ is the position of the target; and H is a position whose x and y values are the same as the original fist position, but with a large enough offset in the z value to avoid collisions. Figure 3.15 shows an example of a target before (a)) and after (b)) the adjustment to avoid collisions. To make the algorithm faster, to test for collisions, only targets below a certain threshold (third input argument) are considered.



a) b)

Figure 3.15: a) Targets near the floor might lead to a collision between the robot and the car (collision highlighted by the red circle); b) Target orientation redefined such that the robot end fist position, H (magenta dot), has a higher z value.

3.6. Graph search in joint space to find path feasible by linear motions

For the shotcrete projection to be uniform, the path must be smooth, avoiding sudden directions and velocity changes at the Tool Centre Point (TCP), i.e., the path point at the tunnel surface. This continuity and smoothness can be partially achieved by assuming linear (or circular) motions between consecutive target poses. As such, a graph-based algorithm was implemented, considering, for the robot being used, the inverse kinematics at each pose of the generated cartesian path. The possible inverse kinematics solutions for each pose, given as a list of joint vectors, are stored, and the lists are all concatenated to form a list of lists of joint vectors. A joint vector is composed by the 6 scalars that determine the position of each of the robot's joints, considering a 6-axis robot. A root node is defined as a special starting node, with no pose or configuration associated with it, serving only as a starting node for the graph (search). All nodes have a corresponding father node (except the root node), an associated joint vector (except the root node), and possible children nodes. The joint vectors that correspond to the same pose form a level in the search graph, as shown in Figure 3.16. This means that there is a target TCP pose associated to each graph level (excluding the root node level), and that the number of nodes on a given level is equal to the number of different joint-space solutions that result in the same pose for the TCP. The goal of the graph search is to find a connecting path from the first level to the last one, i.e., determine a linear-based motion sequence that goes from the first pose to last one. During the search, a child node can only be added to the graph path list if a linear robot motion between its father node and itself is possible. This ensures that the perpendicularity constraint and the wall-to-tunnel surface distance is maintained between targets. The graph path list consists of an ordered list of nodes, where the robotic linear motion between each pair of consecutive nodes is possible, with the final path being the solution returned by the graph search. The implemented graph search algorithm is a depth-first search with a heuristic for child ordering at each node expansion. The nodes generated by a father node are explored in an order given by one of two heuristics. The main goal of the heuristics is to reduce the search time. The closer the values of consecutive joint targets, the more likely it is for them

to be reachable with a continuous linear or circular motion, and the less the robot will travel. The first heuristic was denoted as a norm-based, given it orders the child nodes starting from the one with smallest Euclidean distance to the father node. The second heuristic was denoted as a max-diff heuristic, since it orders the children nodes from the one with the smallest maximum difference in any joint value to the one with largest maximum difference.

When running the program to generate the joint targets list, either heuristic can be used to minimize the graph search time. The feasibility of each linear motion is tested by setting the robot's joints according to the value represented by the father node, and testing if a linear motion to the joint vector associated to the child node being explored is possible (this test is done using RoboDK's Python module, specifically the MoveL method). If the linear motion is not possible, the next (according to the used heuristic) child node gets explored. If all possible children of a given father node have been evaluated with no success, then backtracking occurs. If a linear motion between a father node and a child node is possible, then the child node gets added to the graph path list and the current graph level gets incremented, proceeding to the next depth until a solution is reached. As such, a solution is a path in joint space that is feasible exclusively by linear motions. Given that the maximum search depth (equal to the number of poses of the cartesian path) is limited, and that the branching factor is finite, the algorithm is guaranteed to be complete, i.e., if a solution exists, and enough time is given, the algorithm will find and return that solution. If no solution exists, given enough time, the algorithm will return that information. The pseudocode of the graph search algorithm is shown in appendix A.

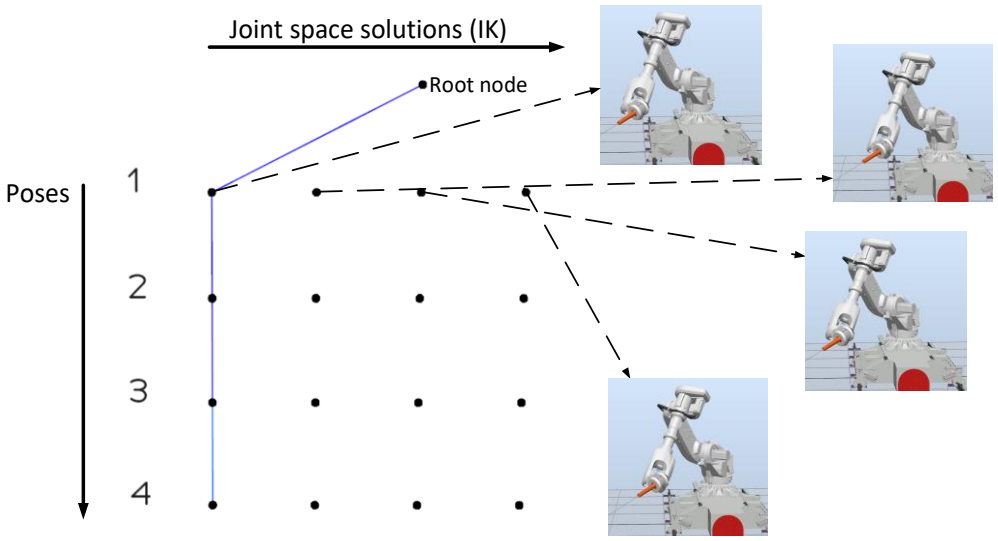


Figure 3.16: In this example, excluding the root node level, each of the nodes in the first four levels of the graph has an associated cartesian pose, and each of those cartesian poses has 4 possible joint space configurations (solutions). The dashed lines show the correspondence between each of the 4 configurations for the first robot pose in the search graph.

3.7.Path execution

The tracks shown in Figure 3.1 serve as a conveyor, allowing the robot to move along the tunnel, applying concrete to the whole tunnel surface. After generating a list of joint vectors (joint space path) for the chosen path type, a connection is established with the robot controller using TCP/IP, with the path generation application being the server and the robot application the client, as shown in Figure 3.17. Each trajectory joint target gets an associated motion type, which can correspond to a joint or linear motion. Currently, each trajectory starting point, associated with a given tunnel section, is reached using a joint motion, with the spraying turned off, while the remaining targets are reached using linear motion, thus guaranteeing that the trajectory remains approximately normal to the tunnel surface in between targets while spraying. The application communicates continuously with the robot controller, sending motion orders in batches, and receiving the corresponding acknowledgements. Given the use of TCP/IP, which is deterministic, no error verification was implemented so far. The shotcrete pump start signal is currently activated when the robot reaches the first target.

To reduce the amount of data sent with each transmission, the trajectory is partitioned and sent in smaller sets, using the TCP/IP-based connection socket, with a synchronized bidirectional communication between the application and the robot controller. Furthermore, to be able to execute linear motions in these controllers, one needs a cartesian target with an associated joint configuration. As such, each joint target sent to the robot is converted to a cartesian target with the corresponding configuration in the controllers, before being used in a motion instruction.

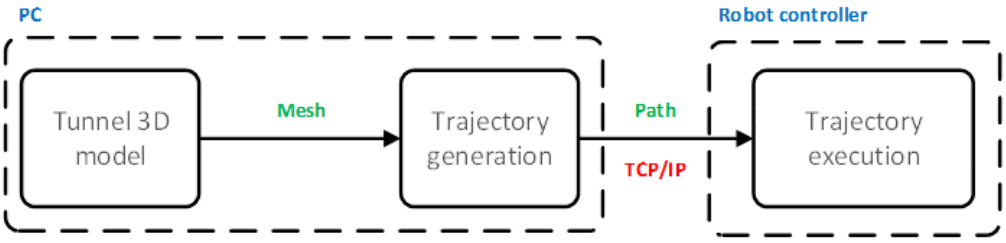


Figure 3.17: The path data are sent to the robot controller via TCP/IP.

4. Shotcrete Simulation

The previous chapter explained the different types of paths that can be generated. To assess which path type and path parameters are optimal for a given tunnel, and to allow an easier assessment of the generated paths, a program was developed to simulate the shotcrete process. The ultimate goal of such a program is to be able to optimize the trajectory parameters (path parameters and TCP speed), which means determining which parameter combination leads to a concrete layer with a desired average thickness and minimum thickness variation (maximum homogeneity). The mesh used is the processed mesh described in Chapter 3, which has a resolution that allows a good compromise between simulation time and accuracy. The shotcrete model used in the developed shotcrete simulation system implemented is based on [27].

4.1. Inputs and path processing

The shotcrete simulation program receives as input arguments the following:

1. Cartesian path (ordered list of homogeneous transformation matrices).
2. Tunnel mesh.
3. Timestep for the simulation.
4. TCP speed.

The cartesian path is created by the program explained in Chapter 3. The tunnel mesh is the same as the one used to generate the path. The timestep is a parameter given by the user and corresponds to the time resolution of the simulation. The TCP speed is also chosen by the user and is crucial to calculate the amount of concrete sprayed at every iteration. The combination of the path parameters and the TCP speed constitute the trajectory parameters.

The input path represents a discrete version of an idealized continuous path. The first step of the simulation program is to sample the input path to obtain a new (discrete) path where the distance between each pair of consecutive targets (measured along the arc length of the input path) is equal to the product of the time step and the TCP speed, as shown in Figure 4.1. Each new target is assigned an orientation, approximately normal to the surface. To understand the principle behind this approach, it might be helpful to imagine a point traversing the input path, and creating targets every time it travels the distance corresponding to the product of the time step and the TCP speed (which corresponds to a distance step).

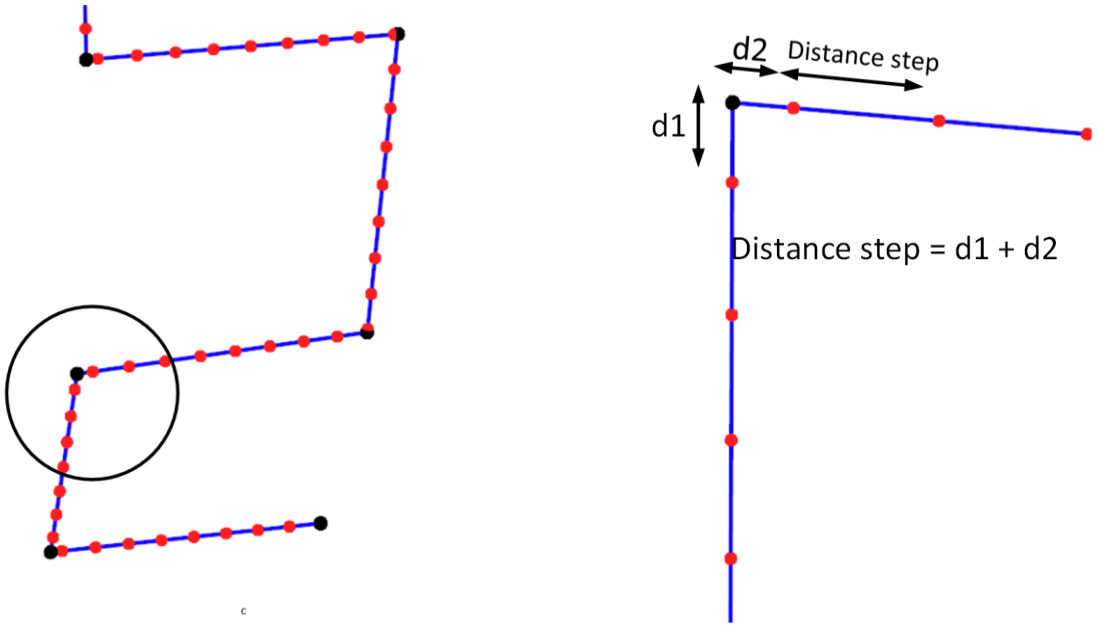


Figure 4.1: a) The original path (black dots) is sampled to produce a new path (red dots); b) Zoomed view of the circled part in (a). The distance between every pair of consecutive red dots (measured along the blue lines) is constant and equal to the product between the time step and TCP speed (distance step).

Depending on the distance step, the processed path may have more or less targets than the original path, as exemplified by Figure 4.1 and Figure 4.2.

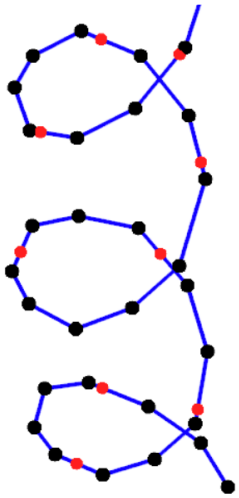


Figure 4.2: If the distance step is too large, the new path (red dots) will have fewer targets then the original path (black dots), resulting in a faster but less precise simulation.

4.2. Concrete flow

The concrete flow, expressed as volume of concrete per time unit, is not constant in the equipment used in the RoboShot@FRC project [28], given the working principle of the

double piston pumps used for shotcrete. As stated in [13] and [29], and as shown in Figure 4.3 and Figure 4.4, respectively, the concrete pump pressure varies periodically over time. . As such, the flow was modelled as a periodic square function, as shown in Figure 4.5.

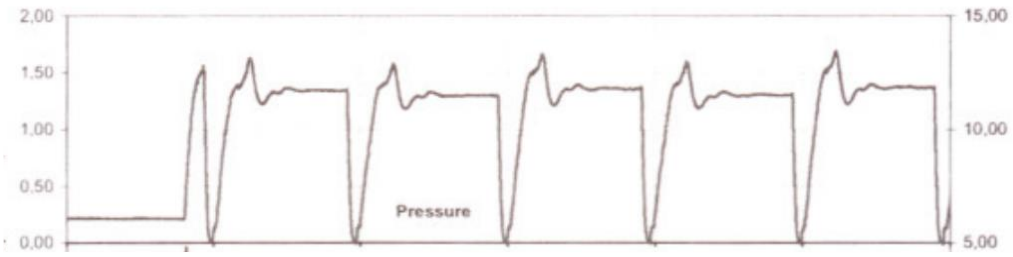


Figure 4.3: Concrete pressure (taken from [13]).

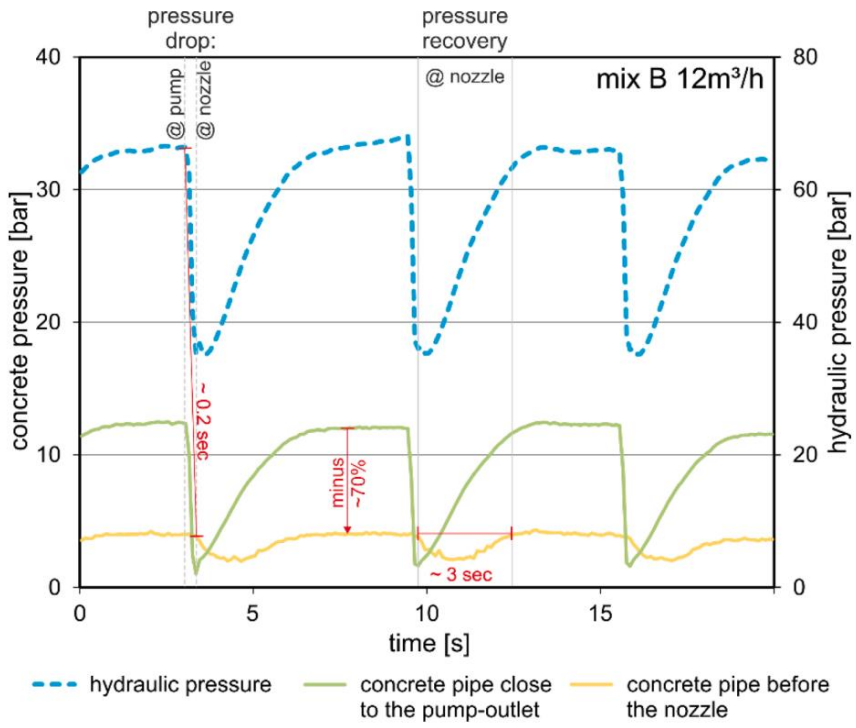


Figure 4.4: Concrete pressure curves (taken from [29]).

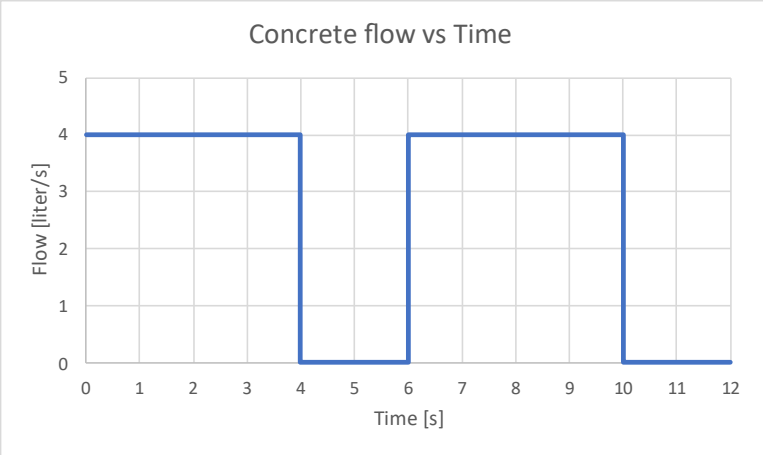


Figure 4.5: The concrete flow is modelled as a periodic square function.

The periodic square function, Q, has a duty-cycle equal to $\frac{2}{3}$ with an average flow of 2.67 liters per second (equal to 9.6 m^3 per hour).

At every iteration the volume of concrete is computed as the definite integral of the concrete flow function. The volume, in liters, sprayed at the i^{th} iteration is:

$$V_i = \int_{T \cdot i - T}^{T \cdot i} Q(t)\, dt,$$

where Q is the periodic square function, and T is the timestep.

4.3.Rebound modelling

The rebound was modelled according to Figure 4.6, as described in [30]. According to authors, the factors that affect rebound are:

1. Nozzle angle to the substrate.
2. Accelerator dose.
3. Nozzle distance to the substrate.
4. Area of application in the tunnel.

The total rebound percentage is the sum of the rebound percentages caused by each of the aforementioned factors. In the RoboShot@FRC project, the accelerator dose used was between 6% and 8%, which means this factor should not contribute at all to rebound. The nozzle-to-wall distance is constant and equal to 1.5 meters, which means it should not contribute to the rebound as well. However, the nozzle angle to substrate, and area of application need to be taken into consideration. Therefore, the created rebound model considers these two factors.

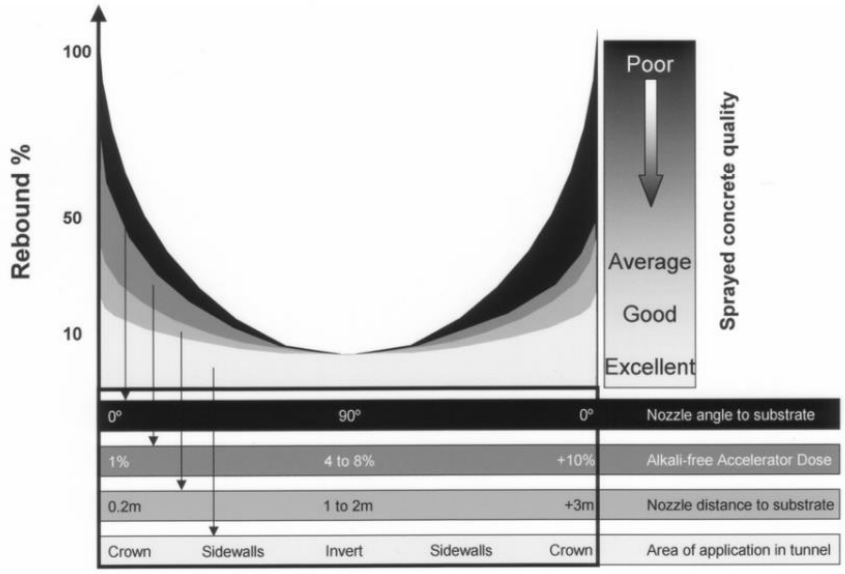


Figure 4.6: Influence of several factor on rebound percentage (taken from [30]).

The rebound is computed at every iteration for all faces that are intercepted by the spray "cone". The spray cone, shown in Figure 4.7, has its orientation and vertex position equal to the nozzle direction and nozzle tip position, respectively. The cone aperture has 18.9° and the spraying distance is 1.5 m from the surface. Figure 4.7 shows the projection cone, the nozzle position, the ray defined by the nozzle position and the position of a generic face, and the normal direction of that generic face for a given iteration. For the sake of this report, "nozzle position" corresponds to the position of the nozzle tip.

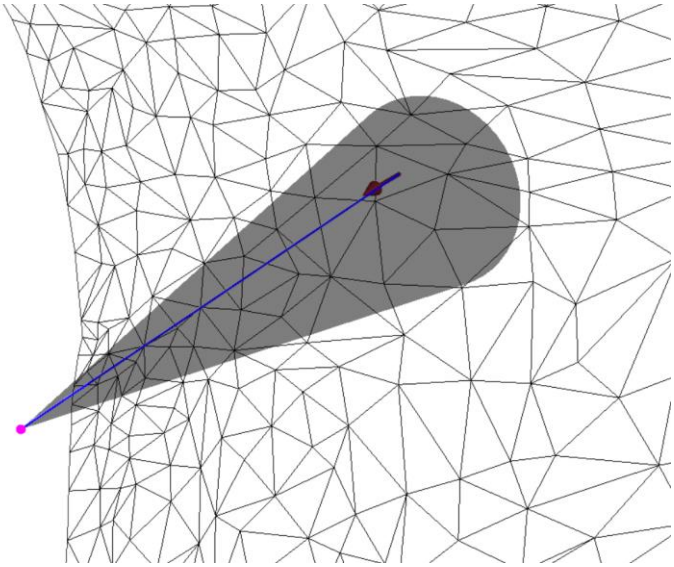


Figure 4.7: The magenta dot, the blue line, the red arrow, and the grey cone represent, respectively, the position of the nozzle, the ray defined by the nozzle position and the position of a generic face, the normal direction of that generic face, and the concrete spray cone. The mesh resolution is deliberately low to help visualize individual faces.

Let β be defined as the angle between between a face's normal direction and the vector with origin at the nozzle position that points to that face (nozzle-to-face vector). Regarding Figure 4.6, a "nozzle angle to substrate" of 90° is equivalent to a β of 0° and vice-versa. According to Figure 4.6 when β is zero, this factor does not cause any rebound (rebound is still non-zero but it is caused by other factors). The relationship between β and the rebound (caused by β) can be approximated by a second-degree polynomial given the parabolic shape of the function shown in black in Figure 4.6. With this in mind, the following function was devised to represent the rebound on a face with an angle, β, between its normal direction and the nozzle-to-face vector:

$$rebound_{\beta}(\beta) = 0.25 * \left(\frac{\beta}{0.5 * \pi}\right)^2$$

Which corresponds to the contribution of angle β to rebound.

The function that relates the zone of the tunnel a face is placed, to the contribution of such placement to the rebound can also be approximated by a second-degree polynomial (see function graph in white in Figure 4.6). This rebound contribution is non constant due to the effect of gravity, causing the most rebound on the faces located at the crown, as expected. To assess how this affects a given face, its normal direction is used. For a face at the invert, its normal direction should be close to [0 0 1] corresponding to a value of rebound contribution close to 0.1. For a face located at the crown, its normal direction should be [0 0 -1], approximately, contributing to a rebound close to 0.25. To capture this effect, the following function was devised, relating the normal direction, n, of a generic face to a value of rebound contribution:

$$rebound_{n}(n) = 0.1 + 0.15 * (0.5 * n \cdot [0 \quad 0 \quad -1] + 0.5)^2$$

The total amount of rebound is given by:

$$rebound(\beta, n) = rebound_{\beta}(\beta) + rebound_{n}(n)$$

$$\Leftrightarrow rebound(\beta, n) = 0.25 * \left(\frac{\beta}{0.5 * \pi}\right)^2 + 0.1 + 0.15 * (0.5 * n \cdot [0 \quad 0 \quad -1] + 0.5)^2$$

At every iteration, this formula is used to compute the rebound at all faces intercepted by the projection cone.

4.4.Shotcrete simulation algorithm

The shotcrete simulation algorithm is mostly based on the algorithm described in [27]. The deposited concrete thickness (or height) in each face is stored in a list, to form a heightmap. At every iteration of the simulation algorithm, the thickness of each face, which represents the amount of concrete adhering to that face, is updated. If a face receives a non-zero amount of concrete its thickness is increased, otherwise it stays the same.

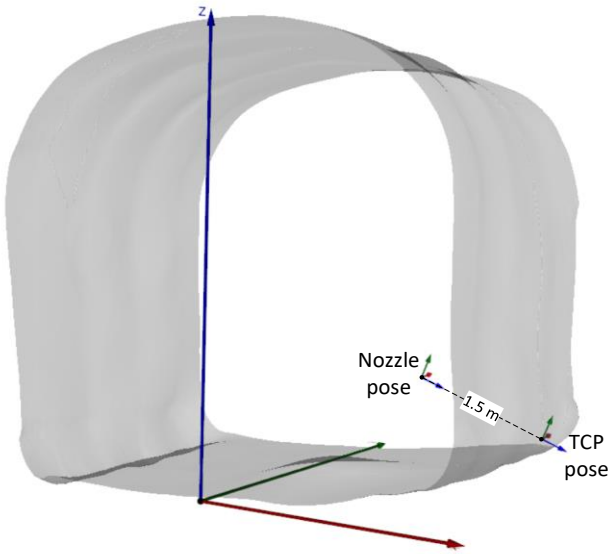


Figure 4.8: The nozzle pose is obtained by translating the TCP pose along the opposite direction given by the z unit vector, a magnitude equal to the designated nozzle-to-wall distance.

The nozzle pose at the i^{th} iteration, exemplified in Figure 4.8, is defined as the following homogeneous transformation matrix:

$$N_i = \begin{bmatrix} \hat{x}_i & \hat{y}_i & \hat{z}_i & p_i - h \cdot \hat{z}_i \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where $\hat{x}_i$, $\hat{y}_i$ and $\hat{z}_i$ are, respectively, the unit vectors that define the orientation of the i^{th} target; p_i is the position of the i^{th} target; and h is the designated nozzle-to-wall distance, which, for the RoboShot@FRC project was set at 1.5 meters, as stated previously.

To figure out how the concrete that is shotcreted is spatially distributed, the authors in [27] sprayed a planar surface at a distance of 1.5 m for an undisclosed amount of time, obtaining a lump of concrete that resembles a solid of revolution. This lump is represented by the solid shown in Figure 4.10, and is called “reference spray deposit”. Given the reference spray deposit is a solid of revolution, the information to recreate it can be summarized in a function that assigns, to any value of radial distance, a value of height (thickness). Such a function can also be created so that the domain values are not, explicitly, radial distances but for example the angles formed by the nozzle direction and the vector defined by the subtraction between the position of a face of the mesh and the nozzle position (nozzle-to-face vector) as exemplified in Figure 4.9. A graphical representation of such a function was given in [27], shown in Figure 4.11.

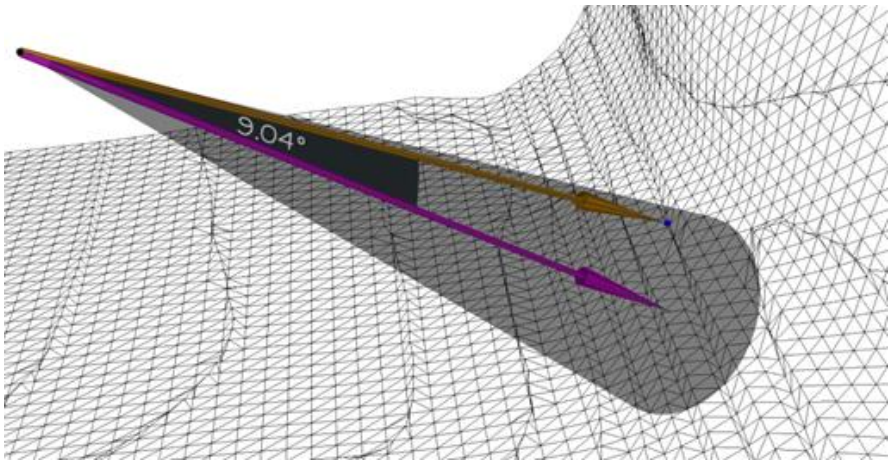


Figure 4.9: Example, showing, the spray cone, the position of a mesh face (blue dot), the nozzle position (black dot), the nozzle direction (magenta arrow) and the nozzle-to-face vector (orange arrow), for an angle of 9.04°.

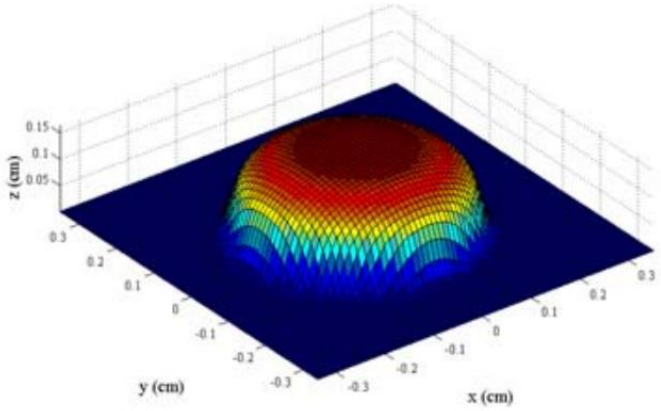


Figure 4.10: Reference spray deposit (taken from [29]).

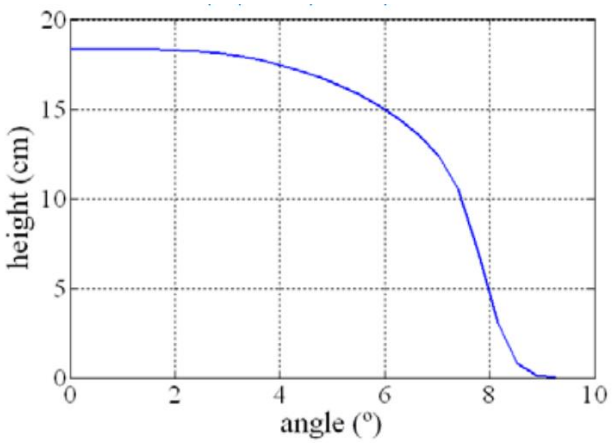


Figure 4.11: Empirical relation between the angle to the nozzle direction and the concrete thickness, for a given amount of spray time (taken from [27]).

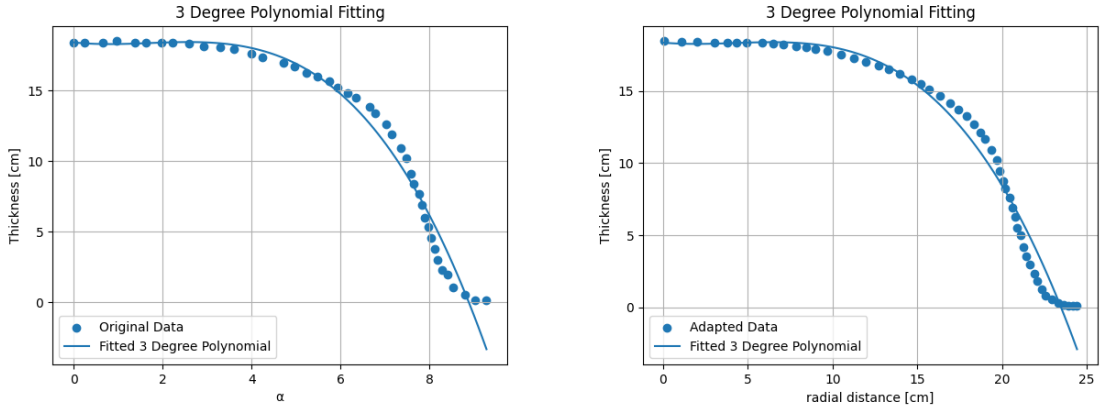
The function shown in Figure 4.11 was approximated using a polynomial interpolation, resulting in the following 3rd degree polynomial, where the angle α is the angle defined by the nozzle direction and the nozzle-to-face vector:

$$thickness(\alpha) = -0.05186 \cdot \alpha^3 + 0.2640 \cdot \alpha^2 - 0.3262 \cdot \alpha + 18.40\ [cm]$$

The polynomial that relates the radial distance, x, to the thickness was also obtained so that the volume of the reference spray deposit shown in Figure 4.10 could be computed:

$$thickness(x) = -0.002663 \cdot x^3 + 0.03354 \cdot x^2 - 0.1008 \cdot x + 18.35\ [cm],$$

where x is the distance from the center of the base of the reference spray deposit to a given point on the plane, that is, the radial distance. The datapoints on which the polynomial interpolation was based, as well as the obtained polynomials, are plotted in Figure 4.12.



a) **b)**

Figure 4.12: a) Original datapoints and 3rd degree polynomial interpolation for the relationship between angle α (in degrees) and the concrete thickness. b) Relationship between the radial distance and the thickness.

The reference spray deposit was obtained by spraying a planar surface at 1.5 m, with the nozzle direction perpendicular to the surface, however, the actual mesh is not a plane so applying the function shown in Figure 4.11 is not enough to compute the amount of concrete that reaches every mesh face. All the concrete that reaches the mesh must have passed through the cone's base plane (where either one of the functions shown in Figure 4.12 can be applied) and there is conservation of volume so it is possible to overcome this problem. The authors of [27] suggest projecting (in the mathematical sense) each face polygon that is intercepted by the spray cone into the plane defined by the cone's base as shown in Figure 4.13.

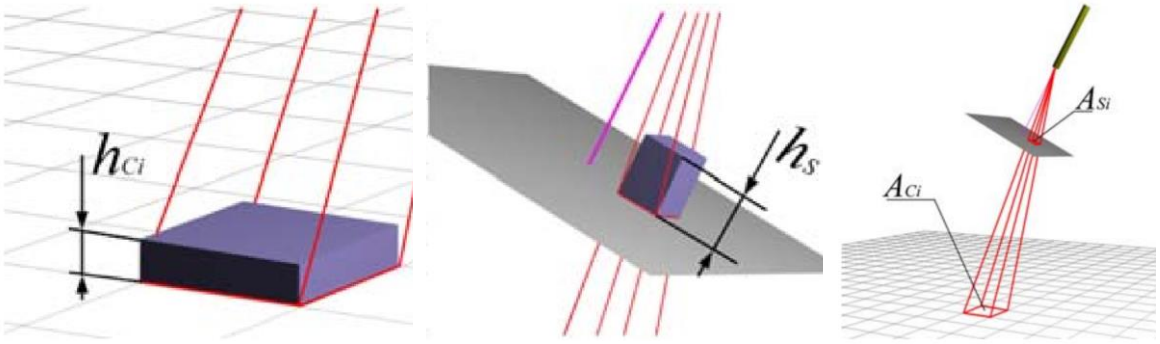


Figure 4.13: Volume that reaches a surface face (a) is equal to the volume that passed through the projection of that face onto the plane of the cone's base (b). c) Ratio between area of a face and its projection area allows computing thickness deposited at that face (taken from [27]).

The volume of sprayed concrete must be conserved, given that there is no compression, and assuming enough output velocity to assume a linear material trajectory. Therefore, the concrete volume that reaches a given face C, of the mesh, must be equal to the concrete volume that passed through its projection polygon, S, in the cone base plane as shown in Figure 4.13. Each face vertex is projected along the direction of the line defined by the vertex itself and the nozzle position (cone vertex) as shown in Figure 4.13 c).

Given that the volume is conserved, it results that:

$$V_C = V_S \Leftrightarrow A_C \cdot h_C = A_S \cdot h_S,$$

where V_C and V_S represent, respectively, the volume that reaches face C and the volume that passes through polygon S (projection of face C into the cone base plane); A_C and A_S represent, respectively, the area of face C and of polygon S; h_C and h_S represent, respectively, the concrete thickness of face C and the thickness that would result if the concrete were deposited on polygon S. The value h_S is the output of the 3rd degree polynomial, so we have:

$$h_C(\alpha) = \frac{A_S}{A_C} \cdot \ thickness(\alpha)\ [cm]$$

The expression for the concrete thickness added to a face C, at the i^{th} iteration, for a sprayed volume equal to the volume of the reference spray deposit, is given by:

$$\Delta h_{i,C}(\beta, n, \alpha) = \frac{A_S}{A_C} \cdot \Delta h_{i,S} \cdot \big(1 - rebound(\beta, n)\big)\ [cm]$$

$$\Leftrightarrow \Delta h_{i,C}(\beta, n, \alpha) = \frac{A_S}{A_C} \cdot thickness(\alpha) \cdot \big(1 - rebound(\beta, n)\big)\ [cm]$$

The reference spray deposit represents the shape of the concrete that resulted from spraying a flat surface for a certain amount of time. It is assumed that any spray deposit on a flat surface is equal to the reference spray deposit, represented in Figure 4.10 and summarized

in Figure 4.11., multiplied by a scalar, that relates the volume sprayed at that iteration of the simulation program and the volume of the reference spray deposit. This means that the polynomial used to approximate the reference spray deposit must be scaled by a factor which, for the i^{th} iteration, is defined by:

$$K(i) = \frac{V_i}{V_R}$$

Where V_i is the volume sprayed at the i[th] iteration and V_R is the volume of the reference spray deposit.

V_R is the volume of the reference spray deposit. It was calculated using *Wolfram Alpha* [31], which allows defining a function and return the volume of the solid resulting from the function revolution resulting, for this case, in a volume of 0.02118 m^3 (21.18 liters). This value is an approximation of the volume of the solid shown in Figure 4.10 and was computed by revolving $thickness(x)$ for $0 \leq x \leq 23.48$ (the value 23.48 corresponds to the function's root) around the y-axis and computing the resulting volume. The definition of the V_i variable has been given in Section 4.2, so it results that:

$$K(i,T,t) = \frac{\int_{T\cdot i-T}^{T\cdot i} Q(t)\,dt}{21.18}$$

Therefore, the complete formula for calculating the added thickness to a generic face C at the i[th] iteration is:

$$\Delta h_{i,C}(\beta,n,\alpha,i,T,t) = \frac{A_S}{A_C}\cdot thickness(\alpha)\cdot\big(1-rebound(\beta,n)\big)\cdot\ K(i,T,t)\ [cm]$$

The expressions for $thickness(\alpha)$, $rebound(\beta,n)$ and $K(i,T,t)$ have already been detailed above (adapted from [27]). The value A_C is the area of face C, while the value A_S is the area of S (the projection of C to the cone base plane), computed using the formula for the area of a triangle with vertices a, b and c:

$$A_{a,b,c} = \frac{1}{2}\cdot\left|\overrightarrow{ab}\times\overrightarrow{ac}\right|\ [m^2]$$

4.5.Mesh construction

The algorithm explained so far for the shotcrete simulation results in a list of scalar values, where each value represents the concrete thickness deposited at each mesh face, for a given set of trajectory and flow parameters. As the shotcrete simulation program runs, the concrete thickness throughout the mesh can be visualized by a colormap, as shown in Figure 4.14.

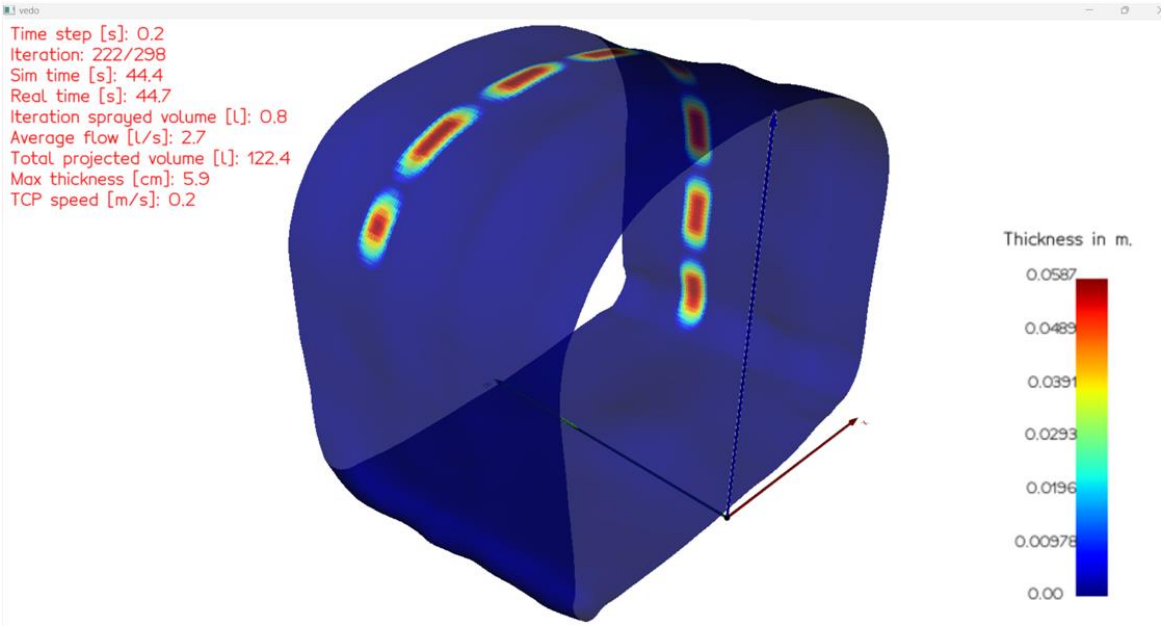


Figure 4.14: Mesh with a colormap for concrete thickness at iteration 222 out of 298. The periodic nature of the concrete flow is evidenced by the periodicity of the colors along the tunnel.

The algorithm to construct the resulting mesh (post-shotcrete mesh) takes as input arguments the following:

1. Original mesh.
2. Heightmap (value of thickness for each mesh face).

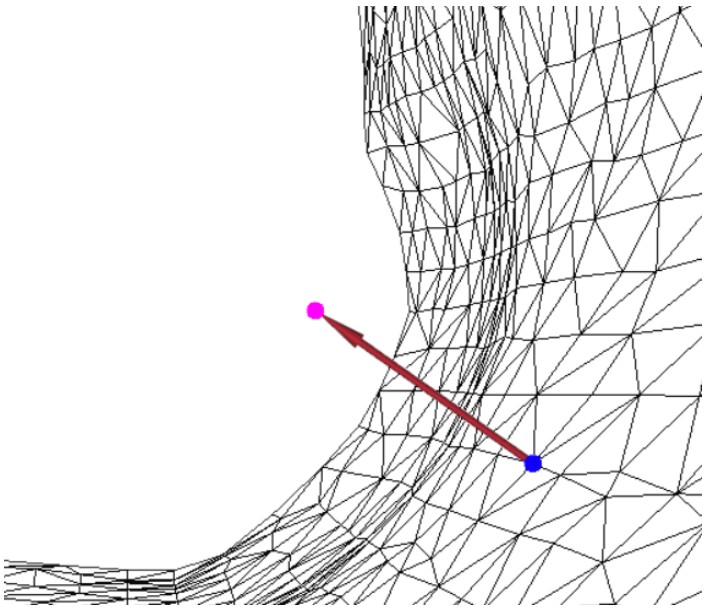


Figure 4.15: The magenta dot represents a vertex of the post-shotcrete mesh (to be created). The red arrow is perpendicular to the original mesh surface and has a magnitude equal to the average of the thicknesses associated with the faces surrounding the vertex in blue, which resulted from the algorithm described in Section 4.4.

The algorithm takes each original mesh vertex and its normal direction, and generates a new vertex given that direction and the average concrete thickness associated with the surrounding faces, as exemplified in Figure 4.15. The post-shotcrete mesh always has the same number of vertices as the original one, as shown in the example in Figure 4.16.

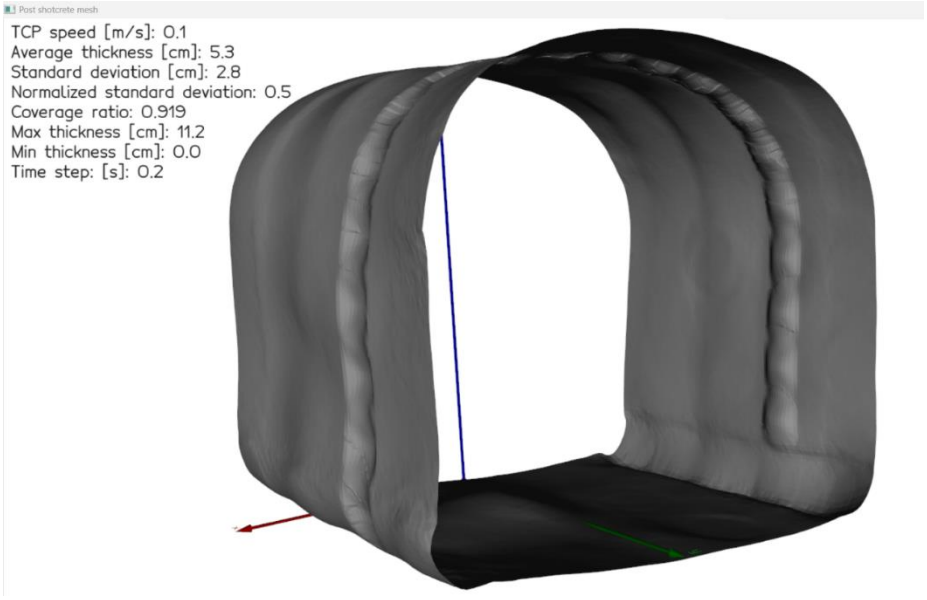


Figure 4.16: Post-shotcrete mesh shown here has the same number of vertices as the original one. The periodicity of the concrete flow resulted in a non-homogenous shotcrete deposition (result for non-optimized path parameters).

5. Tests and Results

This chapter explains how the path generation and shotcrete simulation algorithms are used to optimize the shotcrete process. The type of trajectory as well as the trajectory parameters influence the concrete layer deposited on the tunnel's inner surface. By varying the trajectory parameters (path parameters and TCP speed), and observing the output mesh and/or developing a cost function, it is possible to optimize the shotcrete process, as described below.

5.1. First section optimization

A program was developed that iteratively runs the path generation and shotcrete simulation programs. With every iteration a new set of trajectory parameters is used. The resulting heightmap, that is, the concrete thickness at each face of the mesh is stored. A program was also developed to be able to take the heightmap files as inputs and compute several metrics of the thickness such as the average, standard deviation, normalized standard deviation and the coverage ratio. For all the simulations a time step of 0.2 s was used. The goal is to figure out which trajectory parameters result in a layer with a higher homogeneity.

The straight type trajectories have 3 parameters:

1. TCP speed.
2. Circles' radius.
3. Circles' spatial frequency.

The speed of the TCP, the circles' radius, and their spatial frequency were varied according to Table 1, resulting in a total of 1000 combinations.

Table 1: Each straight with circles trajectory parameter was varied to test a total of 1000 combinations.

	Minimum	**Maximum**	**Step**
TCP speed	0.05 m/s	0.5 m/s	0.05 m/s
Radius	0.05 m	0.5 m	0.05 m
Frequency	1 circles/m	10 circles/m	1 circles/m

For the squared-zigzag trajectory type there are 2 additional parameters: the horizontal and vertical distances. For the squared-zigzag trajectory type, each parameter was varied according to Table 2, for a total of 1024 combinations.

Table 2: Each trajectory parameter was varied to test a total of 1024 combinations.

	Minimum	**Maximum**	**Step**
TCP speed	0.1 m/s	0.4 m/s	0.1 m/s
Radius	0.05 m	0.20 m	0.05 m
Frequency	2 circles/m	8 circles/m	2 circles/m
Horizontal distance	0.1 m	0.4 m	0.1 m
Vertical distance	0.1 m	0.4 m	0.1 m

For each trajectory type, the domain range and resolution were chosen as a compromise between robot feasibility and total time of computation.

Out of all faces that received a non-zero amount of concrete, the ones with largest and smallest positions y coordinate values are identified[7]. Those y values are the bounds of the shotcrete section and all faces within those bounds and above the ground level are used to compute the following metrics:

1. Average thickness.
2. Thickness standard deviation.
3. Normalized thickness standard deviation (ratio of metric 2 over metric 1).
4. Coverage ratio.

The average thickness is a metric that allows the user to identify which sets of trajectory parameters are viable for a desired deposition thickness value. Out of all sets of parameters that result in an average thickness value close to the desired value, the other 3 metrics may be used to select the appropriate trajectory parameters. The standard deviation and the normalized standard deviation measure the homogeneity of the deposited layer. The coverage ratio measures how much of the section's area received a non-zero amount of concrete, relative to the section's total area, which means it is also associated with the homogeneity of the surface.

The developed analysis program allows:

1. Computing the mentioned metrics for all shotcrete simulations (which were ran previously).
2. Selecting, among the shotcrete simulations, the one with a given metric closest to a given value. This selection allows filtering by parameters and/or metrics. The resulting mesh is displayed by using the corresponding heightmap, as shown in the example in Figure 5.1.
3. Plotting the simulations' parameters and metrics, ordered by a specified metric, to assess the impact of each parameter on the resulting metrics, as shown in the example given in Figure 5.2. Also allows filtering by parameters and/or metrics.
4. Generating a 3-D plot of points, where the location of each point represents the set of trajectory parameters, and the color represents the distance of a selected metric to a specified goal value as shown in Figure 5.3 (only available for the straight type paths, because the parameters space for the squared-zigzag path types has more than 3 dimensions). Also allows filtering by parameters and/or metrics.

[7] Given the tunnel direction and geometry, and the WRF, the y direction points along the tunnel for the developed simulations.

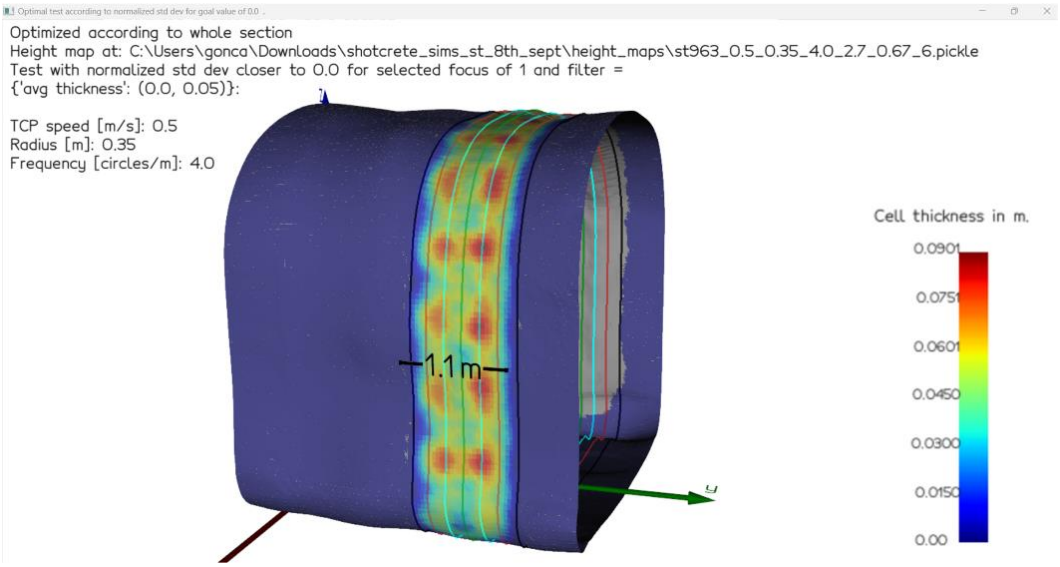


Figure 5.1: Results being shown for the heightmap with lowest normalized standard deviation and average thickness less than 5 cm, of the simulations performed for the straight with circles type paths. The bounds of the section are represented by the black lines which are 1.1 meters apart.

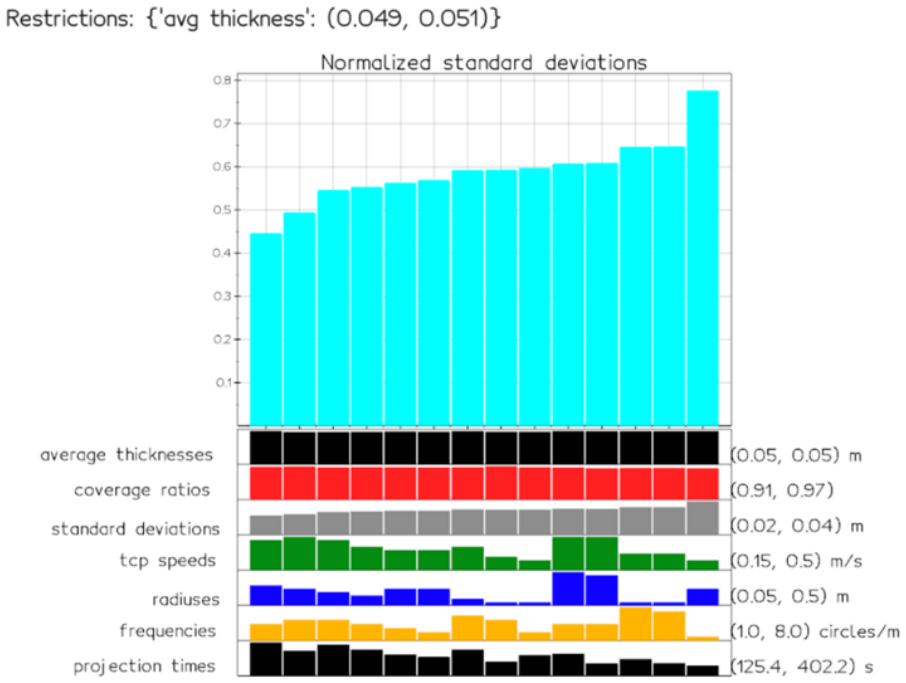


Figure 5.2: Results using the straight with circles type paths simulations, ordered by normalized standard deviations, with a filter for the average thickness. Each row represents one specific test with the parameters values as shown in that column.

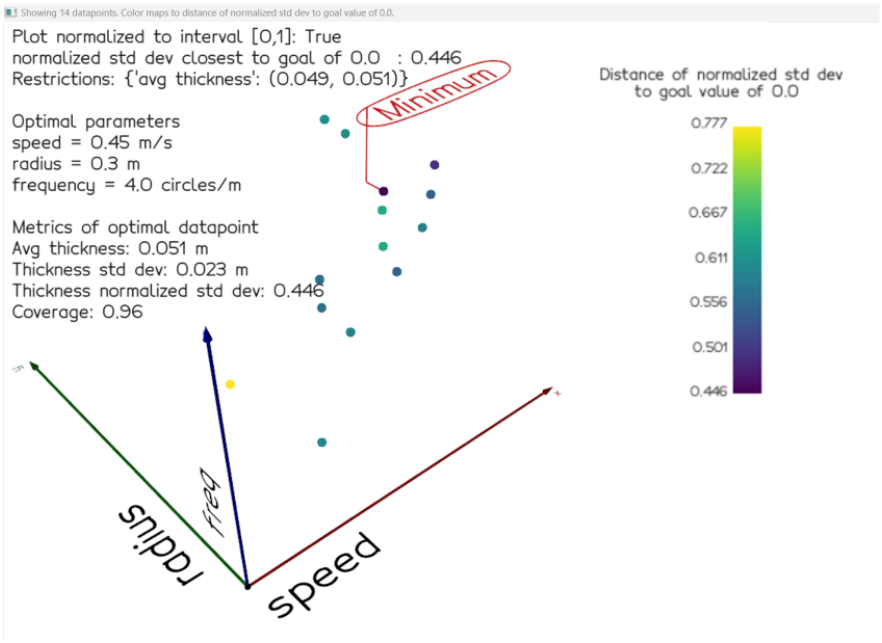


Figure 5.3: Point cloud where the 3-D position of each point represents a set of path parameters. The color of each point represents the distance of the associated heightmap's normalized standard deviation to a specified goal value of 0 m. Filter for average thickness applied.

A user can select the acceptable range of average thickness for the concrete layer, a minimum acceptable coverage ratio, and then use the set of parameters that minimize the standard deviation (or another metric such as the normalized standard deviation, coverage ratio or average thickness).

As an example, for a goal average thickness of 5 cm ± 0.5 cm and a coverage ratio of at least 0.9, the set of parameters that minimize the normalized standard deviation of the thicknesses for the straight with circles and for the squared-zigzag type trajectories are, respectively:

- TCP speed: 0.5 m/s.
- Circles' radius: 0.35 m.
- Circles' spatial frequency: 5.0 circles/m.

And,

- TCP speed: 0.4 m/s.
- Radius of the circles: 0.2 m.
- Spatial frequency of the circles: 2.0 circles/m
- Vertical distance: 0.1 m.
- Horizontal distance: 0.1 m.

The heightmap, as well as the post-shotcrete mesh resulting from these parameters, are shown in Figure 5.4 and Figure 5.5.

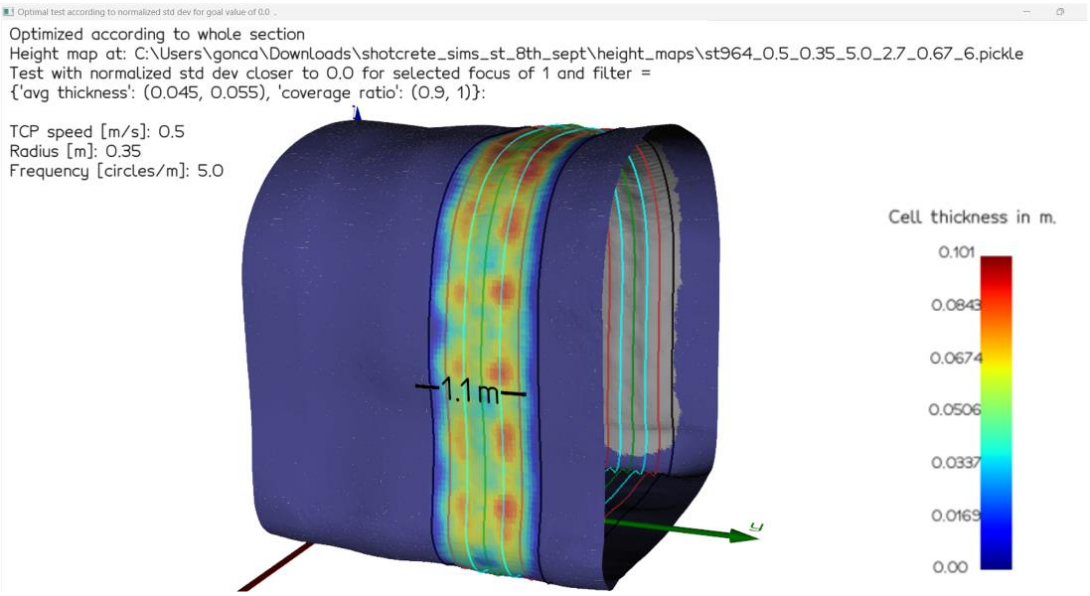


Figure 5.4: Color-mapped mesh corresponding to the simulation that results in the lowest normalized standard deviation of the thickness, for the straight with circles type trajectories, considering an average thickness of 5 cm ± 0.5 cm, and a coverage ratio larger than 0.9. Projecting a layer with this trajectory parameters should take, approximately, 292 s.

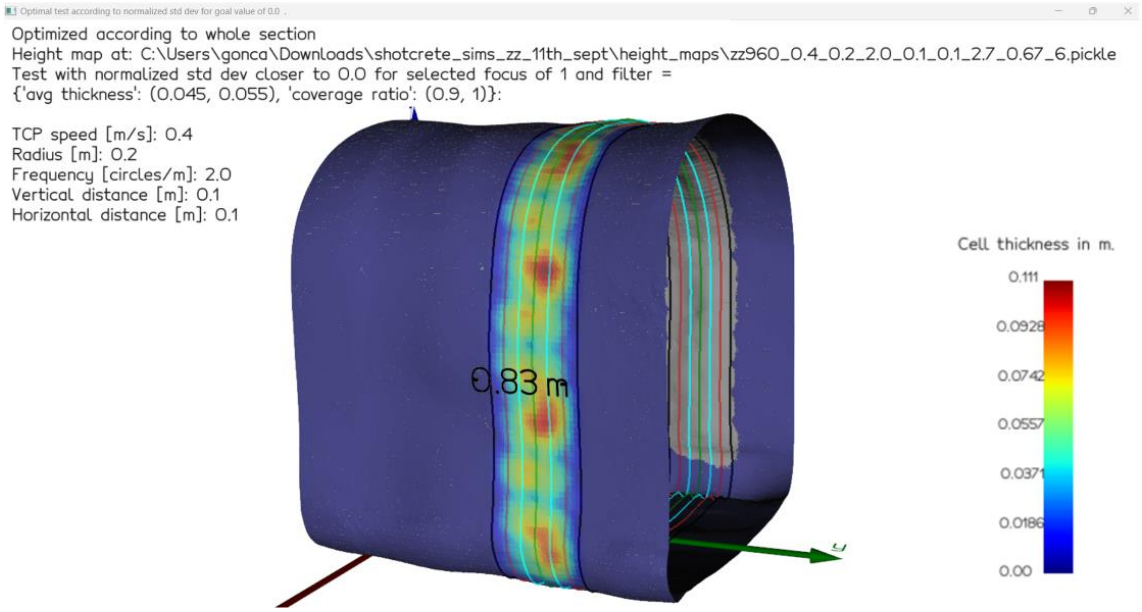


Figure 5.5: Color-mapped mesh corresponding to the simulation that results in the lowest normalized standard deviation of the thickness, for the squared-zigzag type trajectories, considering an average thickness of 5 cm ± 0.5 cm, and a coverage ratio larger than 0.9. Projecting a layer with this trajectory parameters should take, approximately, 202 s.

The metrics associated with the heightmaps represented in Figure 5.4 and Figure 5.5, as well as a function that maps the arc length distance along the middle section to the thickness along the section middle (thickness profile) are shown, respectively, in Figure 5.6 and Figure 5.7.

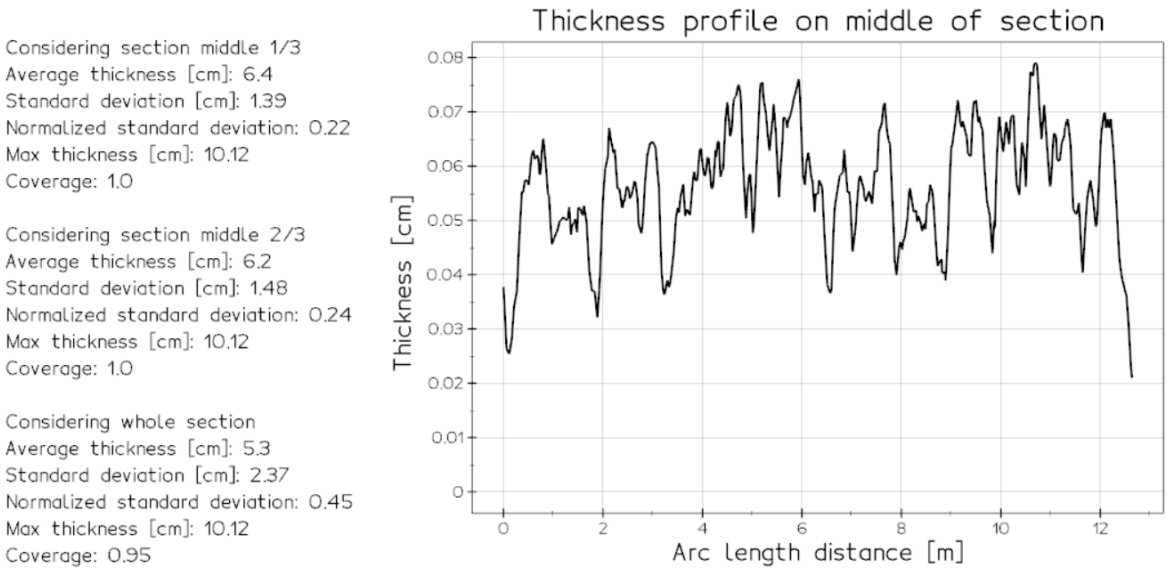


Figure 5.6: Heightmap metrics considering 1/3, 2/3 and the whole section and thickness profile on the middle of the section for the heightmap shown in Figure 5.4.

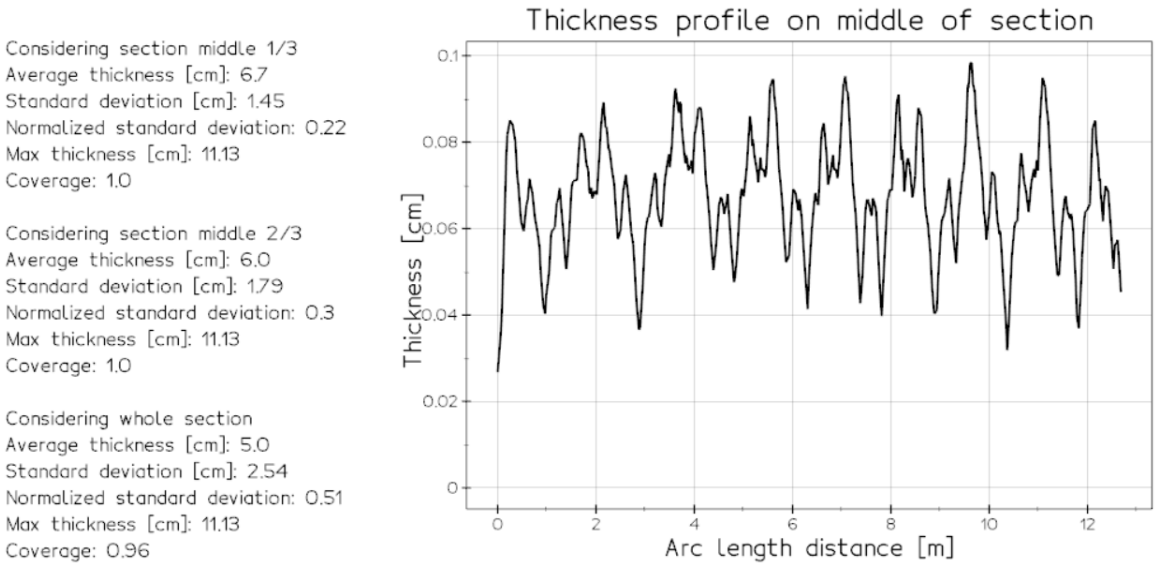


Figure 5.7: Heightmap metrics considering 1/3, 2/3 and the whole section and thickness profile on the middle of the section for the heightmap shown in Figure 5.5.

To clarify, the section middle 1/3, middle 2/3 and middle 3/3 (whole section) are made evident in Figure 5.8.

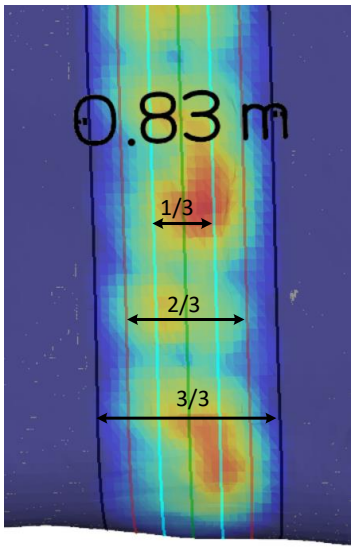


Figure 5.8: Green line marks the middle of the section (along which the thickness profile function is computed), cyan lines mark the boundaries of the centered 1/3 of the section, red lines mark the boundaries of the centered 2/3 of the section, and black lines mark the section boundaries (equivalent to 3/3 of the section).

The heightmaps represented by the color maps shown in Figure 5.4 and Figure 5.5 have normalized standard deviations of 0.45 and 0.51, respectively. Using the trajectory parameters shown in Figure 5.4 and Figure 5.5, each projection, would take in reality around 292 s and 202 s, respectively. Given each projection created sections with width 1.1 m and 0.83 m, respectively, the velocity of projecting sections along the tunnel direction is, respectively, 0.23 m/minute and 0.25 m/minute.

5.2. Optimization of multiple consecutive sections

After obtaining the optimal parameters for a single shotcrete section it is necessary to obtain the optimal increment along the tunnel direction (y axis direction) to place the subsequent section layer. This is done by using the optimal parameters, to shotcrete sections that are displaced from the initial section along the y direction. The superposition of the heightmap obtained in the first layer projection and the heightmap obtained in the second layer projection results in a combined heightmap. The normalized standard deviation of the combined heightmap is the cost function and the only degree of freedom of the optimization process is the increment of the second deposited layer along the y direction. To optimize the second layer placement, the resulting mesh from the first layer simulation is used as input which impacts the rebound values.

The optimal set of parameters for the current example, obtained for a straight with circles type trajectory is:

- TCP speed: 0.5 m/s.
- Radius of the circles: 0.35 m.
- Spatial frequency of the circles: 5.0 circles/m.

To optimize the placement of the second section, several simulations were performed using an increment in the negative y direction between 0.1 m and 1.10 m with a 0.05 m step. The optimal increment to project the second layer was found to be 0.80 m. The combined (first and second section) color-mapped mesh as well as the post-shotcrete mesh corresponding to projecting a second section, 0.80 m away in y from the first one, are shown in Figure 5.9.

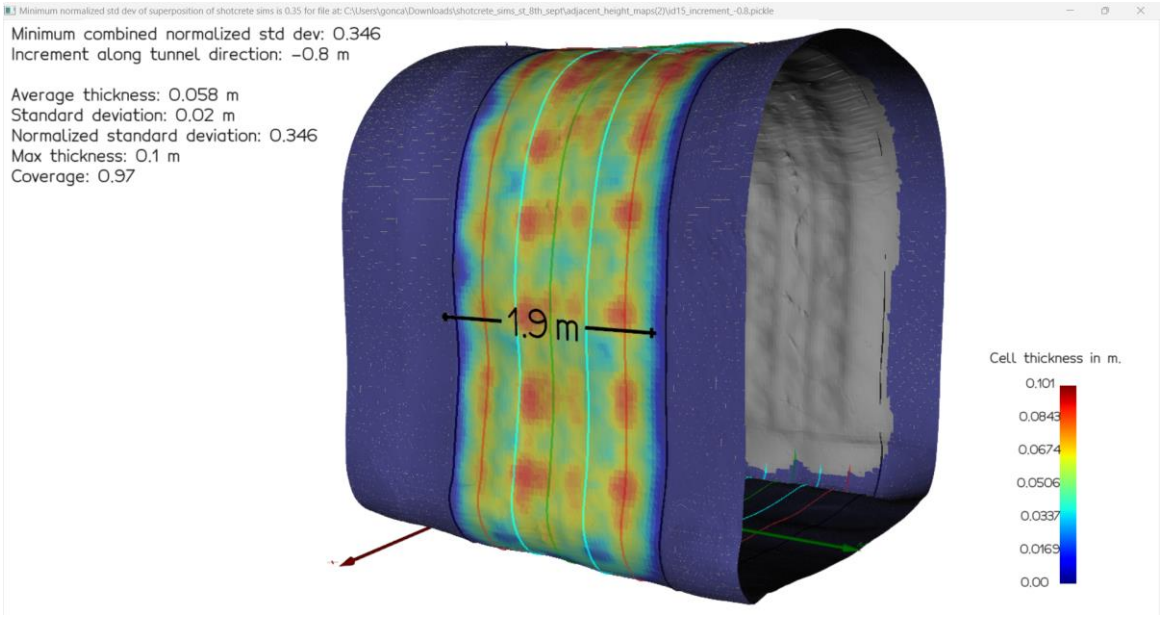


Figure 5.9: Color-mapped mesh, as well as post-shotcrete mesh (grey) after projecting a second layer with a 0.80 m offset in the (negative) y direction. The color map shown is the combined heightmap of the first and second section projections.

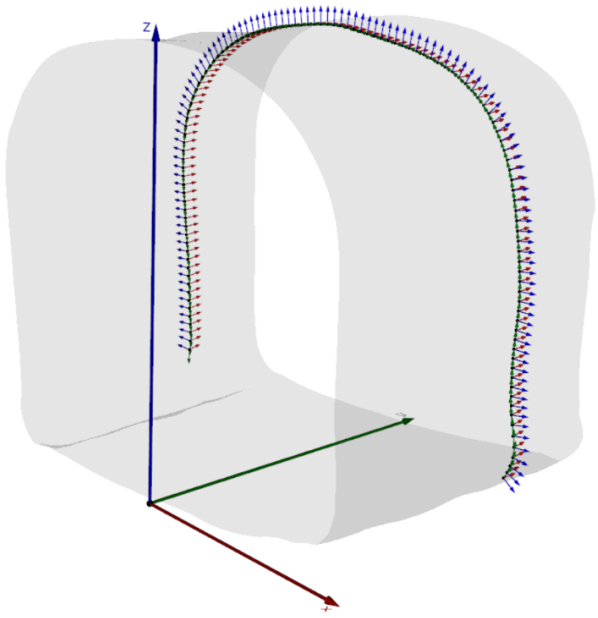


Figure 5.10: Simple path to test projecting with constant flow.

Given the fact that the layers obtained are not very homogenous, with the lowest single section normalized standard deviation being 0.45, it was investigated whether using a constant concrete flow would improve this metric. As such a simple trajectory without circles was created as shown in Figure 5.10. To have a constant concrete flow, the duty-cycle of the square function explained in Section 4.2 was set to one.

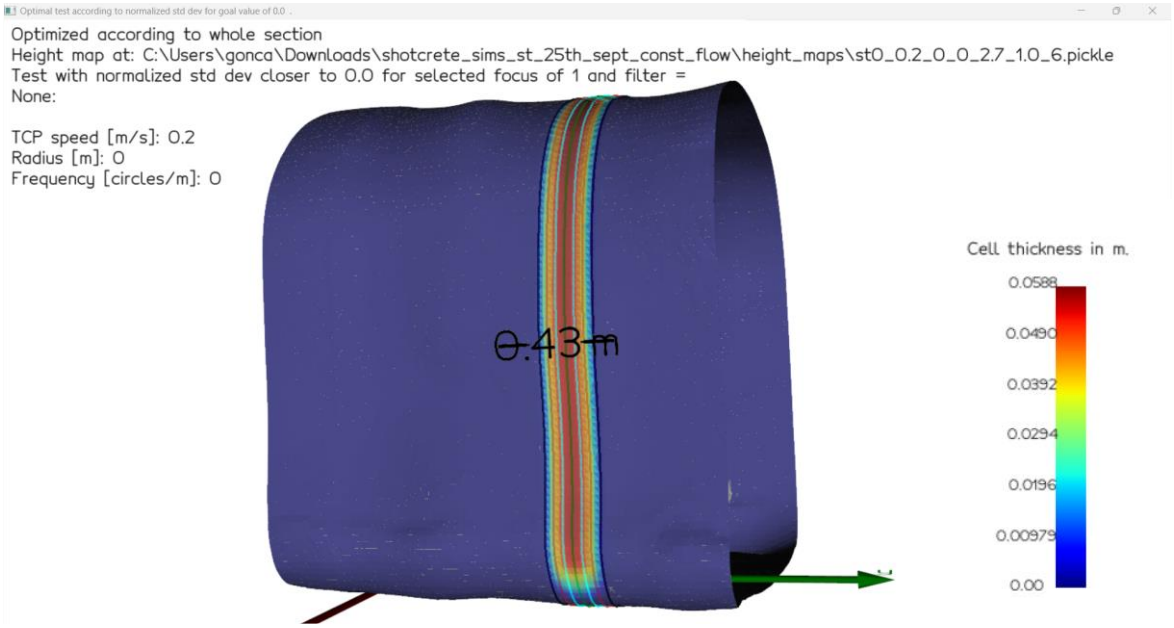


Figure 5.11: Color-mapped mesh for straight trajectory, without circles and TCP speed of 0.2 m/s, using constant flow of 4.0 liters/s.

The shotcrete simulation of the path shown in Figure 5.10, using a TCP speed of 0.2 m/s with a constant flow of 4 liters/s resulted in the color-mapped mesh shown in Figure 5.11 with metrics shown in Figure 5.12. As expected, with a constant concrete flow the deposited layer becomes quite homogeneous. In this case, the normalized standard deviation for the section is 0.32, while the lowest normalized standard deviation of the simulations for a single section with non-constant concrete flow was 0.45. This improvement means that it might be worthwhile investing in concrete pumps capable of outputting a more constant flow. The difference between thickness values in the middle and in the periphery of the layer can be mitigated by the correct placement of the subsequent section layer.

Using the same procedure, detailed before, several second section simulations were performed, using the second section increment along the y direction as the only degree of freedom and the combined heightmap's normalized standard deviation as the cost function to determine the optimal placement (along the tunnel direction) of subsequent sections.

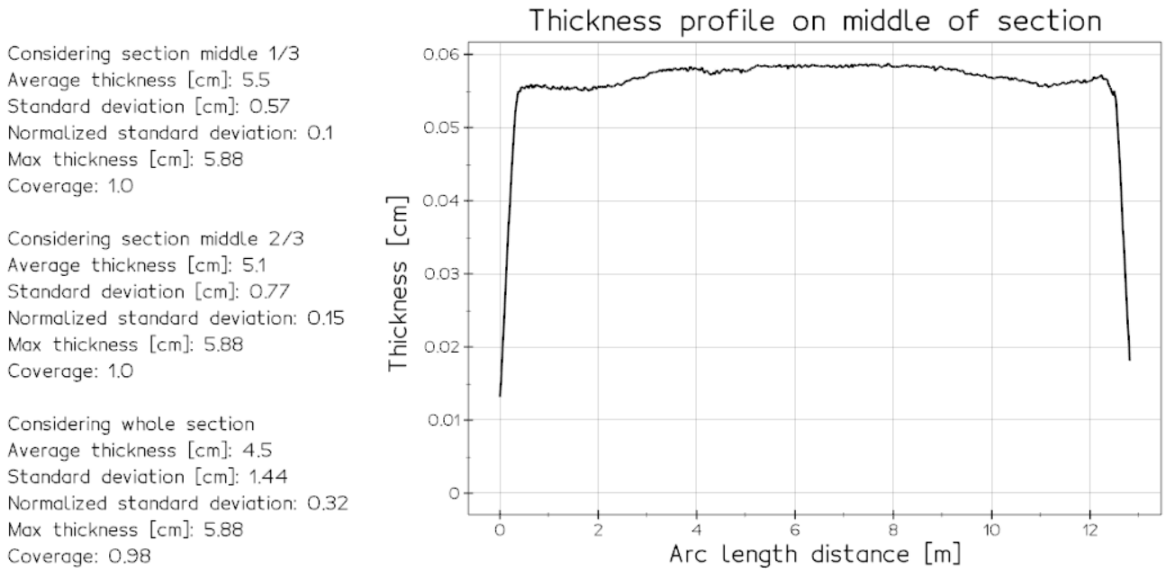


Figure 5.12: Metrics and thickness profile function for heightmap shown in Figure 5.11.

The second section simulations used as minimum and maximum increments, 5 cm and 45 cm, respectively, with a 5 cm step, to obtain an optimal increment of 35 cm. The combined heightmap and metrics are shown in Figure 5.13.

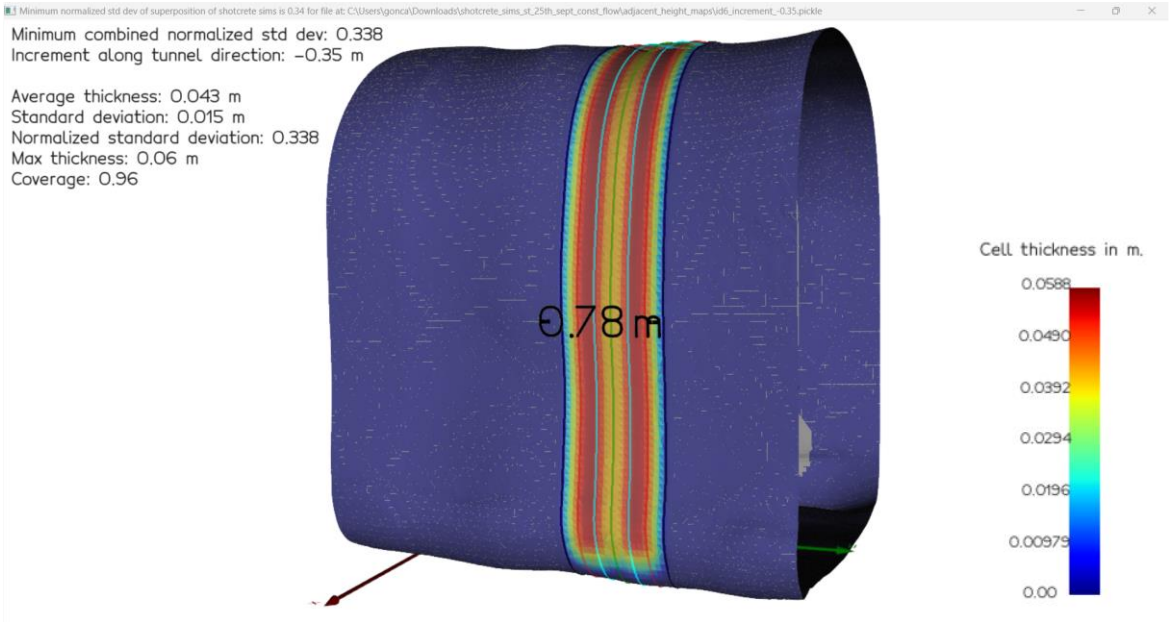


Figure 5.13: Combined heightmap resulting from placing a second section with an offset of 35 cm in the negative y direction, using constant flow of 4.0 liters/s.

Using the optimal increment of 0.35 m, a full tunnel projection simulation was done as shown in Figure 5.14.

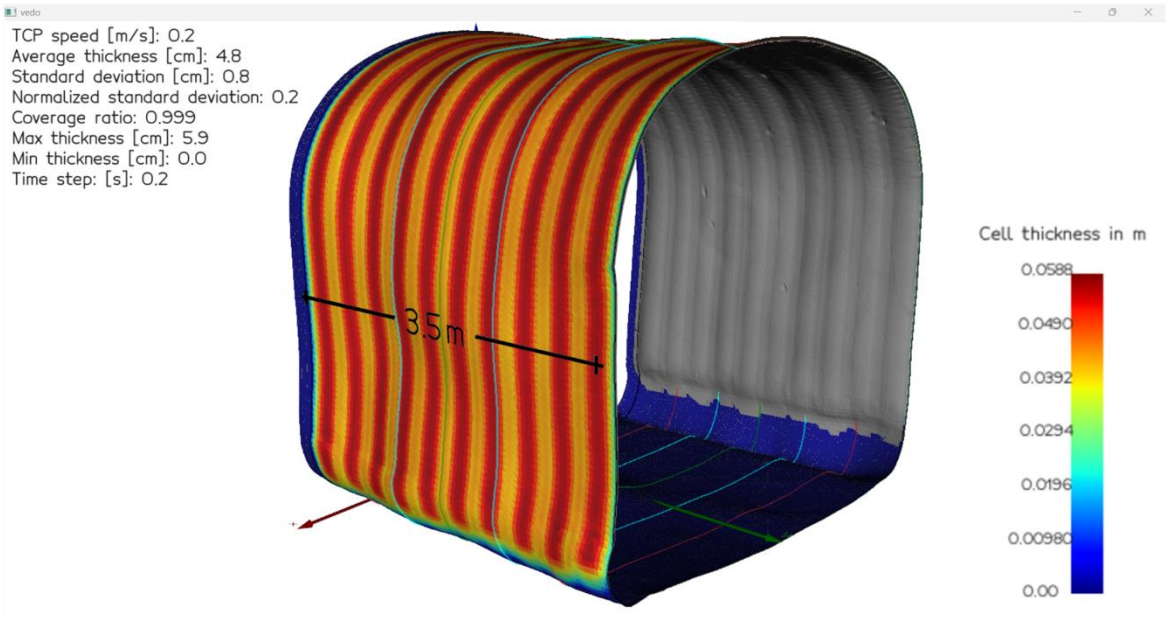


Figure 5.14: Resulting mesh (grey) and heightmap representation of projecting 10 sections with 0.35 m increments along the y direction, using constant flow of 4.0 liters/s.

Table 3 condenses some of the information previously shown, regarding the optimal tests for the different types of trajectories, including single layer and multiple layer cases.

Table 3: Summary of most relevant metrics for the tests previously detailed.

Best Test/Metric	Concrete flow duty-cycle	Number of sections	Average thickness [cm]	Normalized standard deviation	Coverage ratio
Zigzag+circles	2/3	1	5.0	0.51	0.96
Straight+circles	2/3	1	5.3	0.45	0.95
Straight+circles	2/3	2	5.8	0.35	0.97
Straight	1	1	4.5	0.32	0.98
Straight	1	10	4.8	0.20	1.0

5.3. Tests on real robots

In order to test the path generation algorithm, tests were performed on an IRB 1200 ABB robot, on the real IRB 6700, and on a virtual IRB 6700 (using RobotStudio). The communication with both the virtual and the real controllers is done using TCP/IP, and there is no difference from the point of view of the developed program, i.e., the application works the same independently of being controlling a virtual or a real robotic manipulator. A video of the virtual robot executing several squared-zigzag with circles trajectories is available

online[8], which was obtained at an intermediate development stage of the project with another tunnel mesh, to test the communication and robot control.

For the tests performed in RobotStudio and the real robot, a zone parameter of 5 mm was used, meaning the robot starts approaching the next target once its TCP is within 5 mm of the current one, resulting in a slight approximation of the generated trajectory. The shotcrete simulation developed assumes constant TCP speed. However, it was verified that the speed profile of the virtual IRB 6700 (and the real one) was not constant, due to acceleration limits, particularly for higher speeds, as shown in Figure 5.15.

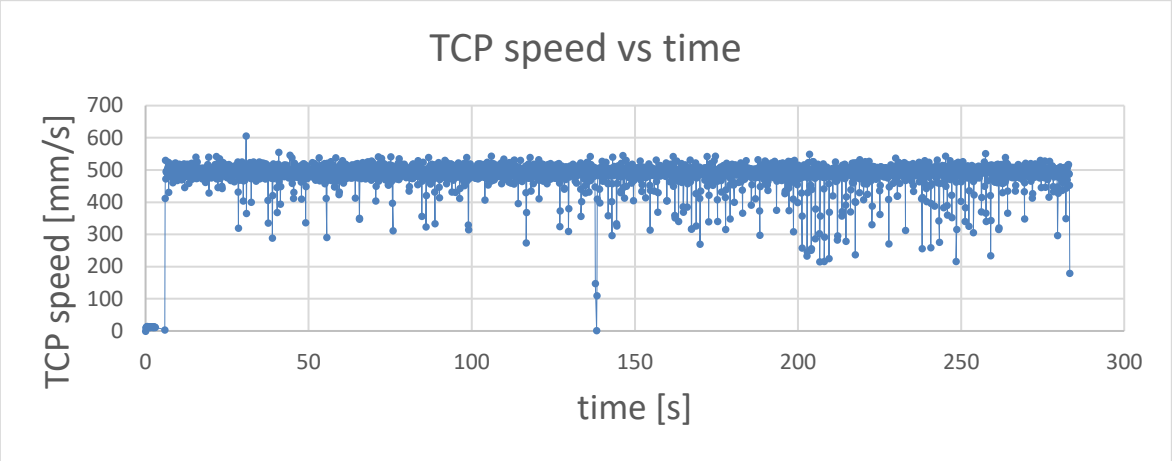


Figure 5.15: Non constant TCP speed represents a discrepancy between reality and the shotcrete simulations.

While the work described included generation of circular paths, the robotic motion type between consecutive targets was always linear. One proposed solution to mitigate this issue is to use circular motions instead of linear ones. As an example, generating a path including correction of unreachable poses, collision avoidance and graph search in joint space can take 10 s, for a path without circles, and 440 s for a squared-zigzag path with circles, on a 4 core, 1.8 GHz computer, with 8 Gb of RAM. For the same types of paths and using the same computer, the shotcrete simulation program might take 23 s and 340 s, respectively, using a time step of 0.2 s.

[8] https://www.youtube.com/watch?v=faR4FFeyqAM

6. Conclusions and Future Work

The path generation algorithm proved to be quite robust. Several hundred different paths were generated without running into unexpected problems. By providing a mesh and some parameters, a path is created that avoids collisions. Unreachable poses are corrected and a graph search in joint space is done to obtain a joint space path feasible by linear motions. Depending on the user preferences, the underlying path may be straight or of squared-zigzag type, and circled motion can also be superimposed. Several section paths can also be created automatically with a specified offset between them. Finally, communication via TCP/IP is implemented which allows transmitting the paths to a virtual or real robot controller.

The developed shotcrete simulation program, based on [27], performed more than 2000 simulations without encountering unexpected issues. It uses cartesian paths created by the path generation program and begins by processing them. Then iteratively simulates concrete depositions on the input mesh, storing the results in a heightmap, which is a list of floats that assigns a thickness value to every mesh face. It is prepared to use a square concrete flow function where the period, duty-cycle and high value are parameters. Simulating constant concrete flow simply requires changing the duty-cycle to one. The amount of concrete sprayed at every iteration is determined by integrating the flow function for an interval equal to the time step, and the spatial distribution of concrete is computed using the algorithm detailed in [27]. To determine the thickness that adheres to each face, the rebound is also taken into account.

One of the discrepancies between the shotcrete simulations performed and reality is the speed profile, assumed constant for the simulations. In the future, developing paths using circular motion type can improve the fidelity of the shotcrete simulation program.

Besides the non-constant TCP speed, two other discrepancies can be identified between the developed shotcrete model and reality. Firstly, the rebound may be different between the cases when the concrete is sprayed against a rock surface and when it is sprayed against a surface already covered in concrete, which means it might be necessary to develop a more complex rebound model. Secondly, the concrete flow was modelled according to the concrete pressure curves. Measuring the actual concrete flow would allow a more accurate flow model, improving the real-time estimation of the shotcrete result.

Of the more than 2000 shotcrete simulations performed using non constant concrete flow, the minimum normalized standard deviation was 0.45. For the test performed with constant flow using a straight path with no circles, the same metric had a value of 0.32 and, when 10 sections were deposited side by side, the metric became 0.2. This confirms that being able to output constant concrete flow improves layer homogeneity significantly. Either using a concrete spraying system that is able to output it at a constant rate, or being able to measure the flow in real-time, and control the TCP speed, accordingly, might be an option to mitigate this issue.

While several papers concerning shotcrete automation and more specifically trajectory generation for shotcrete automation have been mentioned in Chapter 2, the work described in the current report stands out by allowing different path types to be generated, while combining these with a shotcrete simulation program to optimize the generated trajectory parameters.

Bibliography

[1] S. Nabulsi, A. Rodriguez, and O. Rio, 'Robotic Machine for High-Quality Shotcreting Process', in *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, Jun. 2010, pp. 1–8.

[2] G. Liu, X. Sun, Y. Liu, T. Liu, C. Li, and X. Zhang, 'Automatic spraying motion planning of a shotcrete manipulator', *Intel Serv Robotics*, vol. 15, no. 1, pp. 115–128, Mar. 2022, doi: 10.1007/s11370-021-00405-3.

[3] I. Galan, A. Baldermann, W. Kusterle, M. Dietzel, and F. Mittermayr, 'Durability of shotcrete for underground support– Review and update', *Construction and Building Materials*, vol. 202, pp. 465–493, Mar. 2019, doi: 10.1016/j.conbuildmat.2018.12.151.

[4] G. Bernardo, A. Guida, and I. Mecca, 'Advancements in shotcrete technology', presented at the STREMAH 2015, A Coruña, Spain, Jul. 2015, pp. 591–602. doi: 10.2495/STR150491.

[5] D. Beaupré, 'Rheology of high performance shotcrete', University of British Columbia, 1994. doi: 10.14288/1.0050424.

[6] G. Xue, R. Li, S. Liu, and J. Wei, 'Research on Underground Coal Mine Map Construction Method Based on LeGO-LOAM Improved Algorithm', *Energies*, vol. 15, no. 17, Art. no. 17, Jan. 2022, doi: 10.3390/en15176256.

[7] M.-Y. Cheng, Y. Liang, C.-M. Wey, and J.-C. Chen, 'Technological enhancement and creation of a computer-aided construction system for the shotcreting robot', *Automation in Construction*, vol. 10, no. 4, pp. 517–526, May 2001, doi: 10.1016/S0926-5805(00)00104-7.

[8] L. Malmgren, E. Nordlund, and S. Rolund, 'Adhesion strength and shrinkage of shotcrete', *Tunnelling and Underground Space Technology*, vol. 20, no. 1, pp. 33–48, Jan. 2005, doi: 10.1016/j.tust.2004.05.002.

[9] X. Lin, D. Song, M. Qin, W. Zhang, X. He, and B. Xie, 'An Automatic Tunnel Shotcrete Robot', in *2019 Chinese Automation Congress (CAC)*, Nov. 2019, pp. 3858–3863. doi: 10.1109/CAC48633.2019.8996350.

[10] L. Chun-Lei, S. Hao, L. Chun-Lai, and L. Jin-Yang, 'Intelligent Detection for Tunnel Shotcrete Spray Using Deep Learning and LiDAR', *IEEE Access*, vol. 8, pp. 1755–1766, 2020, doi: 10.1109/ACCESS.2019.2962496.

[11] G. Girmscheid and S. Moser, 'Development of a high performance fully automated application system for shotcrete', in *Proceedings of ISEC-02*, Swets & Zeitlinger, 2003, pp. 1523–1529. doi: 10.3929/ethz-a-005935889.

[12] G. Girmscheid and S. Moser, 'Fully Automated Shotcrete Robot for Rock Support', *Computer-Aided Civil and Infrastructure Engineering*, vol. 16, no. 3, pp. 200–215, May 2001, doi: 10.1111/0885-9507.00226.

[13] Á. Rodríguez and O. Río, 'Analysis of real time technical data obtained while shotcreting: An approach towards automation', presented at the ECCOMAS Thematic Conference on Computational Methods in Tunnelling (EURO:TUN 2007), Vienna, Austria, 2007, pp. 156–165.

[14] 'SmartSpray automated concrete spraying', Normet. Accessed: May 09, 2022. [Online]. Available: https://www.normet.com/smartspray/

[15] G. Moniz and H. Costelha, 'Path Generation and Execution for Automatic Shotcrete in Railway Tunnels', in *2023 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Apr. 2023, pp. 214–219. doi: 10.1109/ICARSC58346.2023.10129548.

[16] J. A. O. Barros *et al.*, 'A Multidisciplinary Engineering-Based Approach for Tunnelling Strengthening with a New Fibre Reinforced Shotcrete Technology'. Rochester, NY, Jul. 07, 2023. doi: 10.2139/ssrn.4503644.
[17] N. Ginouse and M. Jolin, 'Investigation of spray pattern in shotcrete applications', *Construction and Building Materials*, vol. 93, pp. 966–972, Sep. 2015, doi: 10.1016/j.conbuildmat.2015.05.061.
[18] J. F. Lamond and J. H. Pielert, *Significance of Tests and Properties of Concrete and Concrete-making Materials*. ASTM International, 2006.
[19] C. Camp, 'CIVL 1101 - Properties of Concrete', CIVL 1101 - Properties of Concrete. Accessed: Apr. 09, 2022. [Online]. Available: http://www.ce.memphis.edu/1101/notes/concrete/section_3_properties.html
[20] S. C. Gnanaraj, R. B. Chokkalingam, and G. LiziaThankam, 'Effects of Admixtures on the Self Compacting Concrete State of the Art Report', *IOP Conf. Ser.: Mater. Sci. Eng.*, vol. 1006, no. 1, p. 012038, Dec. 2020, doi: 10.1088/1757-899X/1006/1/012038.
[21] P. Sulman, 'Shotcrete Equipment: Wet or Dry?', *Shotcrete magazine*, vol. 11, no. Summer 2009, pp. 4–5, 2009.
[22] D. Morgan, N. McAskill, B. Richardson, and R. Zellers, 'A Comparative Evaluation of Plain, Polypropylene Fiber, Steel Fiber, and Wire Mesh Reinforced Shotcretes', presented at the International symposium on recent developments in concrete fiber composites, Washington, DC, USA, 1989, pp. 78–87. Accessed: Apr. 08, 2022. [Online]. Available: https://onlinepubs.trb.org/Onlinepubs/trr/1989/1226/1226-011.pdf
[23] M. Croutch, 'Pedestrian Tunnel at Billy Bishop Toronto City Airport', *Shotcrete magazine*, vol. 16, no. Fall 2014, pp. 38–40, 2014.
[24] C. Zynda, 'ACI Nozzleman Certification and Underground Robotics', *Shotcrete magazine*, vol. 18, no. fall 2016, pp. 40–42, 2016.
[25] A. Nitschke, F. Townsend, and R. Schallom, 'Encapsulation of Reinforcement in Tunnel Shotcrete Final Linings', *Shotcrete magazine*, vol. 22, no. Fall 2020, 2020. Accessed: Apr. 09, 2022. [Online]. Available: https://shotcrete.org/wp-content/uploads/2021/05/Fall2020_SCM_interactive.pdf
[26] R. Friedberg, 'Rodrigues, Olinde: "Des lois géométriques qui régissent les déplacements d'un système solide..."', translation and commentary'. arXiv, Nov. 14, 2022. doi: 10.48550/arXiv.2211.07787.
[27] G. Velez, L. Matey, A. Amundarain, F. Ordás, and J. A. Marín, 'Real-Time Modelling and Rendering of Sprayed Concrete', Jun. 2011.
[28] Putzmeister Ibérica S.A., 'Sika-PM500 by Putzmeister'. Jun. 28, 2006.
[29] M. Sakoparnig *et al.*, 'On the significance of accelerator enriched layers in wet-mix shotcrete', *Tunnelling and Underground Space Technology*, vol. 131, p. 104764, Jan. 2023, doi: 10.1016/j.tust.2022.104764.
[30] T. Melbye, 'Sprayed Concrete for Rock Support'. Accessed: Aug. 20, 2023. [Online]. Available: http://www.episkevesold.civil.upatras.gr/selida%20ektoks%20skyrod/arthra%20pdf/SprayedConcrete.pdf
[31] 'Wolfram|Alpha: Making the world's knowledge computable'. Accessed: Nov. 17, 2023. [Online]. Available: https://www.wolframalpha.com

Appendices

Appendix A – Graph search pseudocode

```
# Returns either a path in joint space feasible exclusively by
# linear motions or information that that is not possible
Function Graph_search_in_joint_space(nodes):

	curr_level = 1 # Current level of the graph search
	# Solution is initialized with the node at index 0 of the first level
	solution = []
	solution.append(nodes[curr_level][0])
	nodes_explored = [] # List of explored nodes
	nodes_explored.append(nodes[curr_level][0])

	# While the final level (pose) has not been reached
	while curr_level < max_level:
		# If the root node is reached make the solution at level 1 the next
		# node on level 1
		if curr_level == 0:
			# Move immediately to level 1 given root node is ficticious
			curr_level = 1
			solution[1] = next node in nodes[1] # Update solution
			# Add the node to the nodes_explored list
			nodes_explored.append(solution[1])

		# Set robot joint according to the current father node
		set_robot_joints_to(curr_solution[curr_level])

		# For every node in the level immediately below
		for node in nodes[curr_level+1]:
			# The If statement is satisfied if node has not yet been explored
			# and a linear motion is possible between itself and the father node
			if (node not in nodes_explored) &
				(linear motion possible between
				curr_solution[curr_level] and node):
				# Current solution includes this node at level equal to
				# curr_level
				solution[curr_level] = node
				# Move to next graph level
				curr_level = curr_level + 1
				# Add node to the list of explored nodes
				nodes_explored.append(node)
				break

		nodes_explored.append(node) # Add node to the list of explored nodes

		# If this point is reached, backtracking occurs
		# Keep backtracking until finding nodes at curr_level+1 that are yet
		# to be explored
		while all(node in nodes_explored for node in nodes[curr_level + 1]):
			curr_level = curr_level - 1 # Backtrack
			# If no more levels to backtrack to
			# then there is no solution
			if curr_level < 0:
				return "No solution"

	return solution # Ordered list of joint vectors
```