Automated Fiber Placement for Dome-Type Structures Using Dual Robots

Yasaman Hedayatnasab

A Thesis

in

The Deportment

of

Mechanical, Industrial & Aerospace Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Mechanical Engineering) at

Concordia University

Montréal, Québec, Canada

October 2024

CONCORDIA UNIVERSITY

School of Gradation Studies

This is to certify that the thesis prepared

By: **Yasaman Hedayatnasab**

Entitled: **Automated Fiber Placement for Dome-Type Structures Using Dual Robots**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Mechanical Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Suong Van Hoa

_____ Examiner
Dr. Suong Van Hoa

_____ Examiner
Dr. Mehdi Hojjati

_____ Supervisor
Dr. Wen-Fang Xie

_____ Co- Supervisor
Dr. Farjad Shadmehri

Approved by _____

Department of Mechanical and Industrial Engineering

_____ 2024

_____

Faculty of Engineering and Computer Science

ABSTRACT

Automated Fiber Placement for Dome-Type Structures Using Dual Robots

Yasaman Hedayatnasab

The growing complexity and precision required in modern manufacturing processes have led to an increased use of automation and robotics, particularly in advanced composite material fabrication techniques such as Automated Fiber Placement (AFP). This thesis explores the ways to improve the fiber placement process and quality through the use of dual robot systems. The principal objective of this research is to investigate the potential of dual-robot systems to improve the precision and efficiency of fiber placement on dome-type geometries with singularity avoidance, particularly within the aerospace industries.

In this study, a dual robot system consisting of two industrial robots, i.e., Fanuc M20-iA with an automated fiber placement (AFP) head and a second robot are used to simulate the fiber placement process on a curved, dome-type surface. A typical example of this type of geometry in aerospace structures is a fuselage pressure bulkhead. RoboDK, an offline robot simulation and programming tool, is employed to program and test the kinematic and dynamic constraints of the robots, thereby ensuring precision in the laying down of fibers. Moreover, simulations are conducted to overcome the geometric limitations in the fiber placement, calculate the manipulability and optimize the path planning for both robots in the dual robot system.

The simulations on the fiber placement in three kinds of dual robot systems have been conducted. The simulation examines the influences of path modifications and robotic manipulator configurations on the overall efficiency. By simulating the possible options of robot combinations, this thesis offers insights into enhancing the precision and adaptability of fiber placement using dual robot. The comparison results of three dual robot system shows that the integration of the Fanuc M20-iA robot with a 6 DOF parallel robot performs best in terms of precise positioning on the path, singularity avoidance and always keeping high functionality. Findings of this thesis are expected to contribute to robotic fiber placement, emphasizing the potential for more effective and reliable automation in composite manufacturing processes. Future work will focus on further refinement of the simulation models and expanding the approach to multi-robot coordination and experimental tests results.

Acknowledgments

I would like to begin by expressing my sincerest gratitude to my supervisors, Professor Wen-Fang Xie and Professor Farjad Shadmehri. Their exemplary guidance and support had an immense value throughout the course of my research. Their profound insight into the subject matter, empathetic support, and readiness to respond promptly were instrumental in shaping my research and refining this thesis. The completion of this work would not have been possible without the expertise and encouragement provided by my supervisors.

I would also like to express my gratitude to all my colleagues in Professor Xie's and Professor Shadmehri's research groups, including Alireza Saboukhi, Ehsan Zakeri, Marizeh Masoodi Nia, Shayan Ghiasvand and Ibrahim Babiker for their collaboration and support during this process.

I would like to express my sincerest gratitude to my parents and my sister for their unwavering support and encouragement throughout my academic career. I am indebted to them for their assistance and guidance.

Contents

List of Figures

List of Tables

List of Abbreviation

AFP	Automated Fiber Placement
ATL	Automated Tape Laying
DOF	Degrees of Freedom
RTM	Resin Transfer Molding
CAM	Computer-Aided Manufacturing
CCR	Clamp Cut and Restart
CSV	Comma-Separated Values
DH	Denavit-Hartenberg
TS	Thermoset
TP	Thermoplastic

List of Symbols

T_i	Transformation matrix of joint i
α_i	Link twist angle of joint i
a_i	Link length of joint i
d_i	Link offset of joint i
θ_i	Joint angle of joint i
J	Jacobian matrix
J_v	Linear velocity Jacobian
J_ω	Angular velocity Jacobian
ω	Manipulability measure
z_i	Unit vector along the z-axis of joint i
o_i	Position vector of joint i
P	Position vector
R	Rotation matrix
$R_x(\theta_x)$	Rotation matrix around the x-axis (roll)
$R_y(\theta_y)$	Rotation matrix around the y-axis (pitch)
$R_z(\theta_z)$	Rotation matrix around the z-axis (yaw)
P'	Transformed position vector
T	Homogeneous transformation matrix

Chapter 1

1 Introduction

1.1 Overview

Composite materials have been used in widespread applications across various industries, including aerospace, automotive, wind energy, civil engineering, and medical devices. These materials are favored because of their good specifications such as their lightweight nature, high resistance to chemicals and corrosion, superior impact properties, and their excellent mechanical performance. They also offer great design flexibility meanwhile enhancing strength-to-weight and stiffness-to-weight ratios compared to conventional materials like steel, aluminum, or titanium alloys. The aerospace industry is replacing traditional materials like aluminum in fuselage and wing components with composites, which can account for up to 65% of the empty weight of modern aircraft [1].

The Boeing 787 Dreamliner is designed to carry around 250 passengers and consume 15-20% less fuel than any other commercial aircraft that are currently in industry. It is the first jetliner whose primary structure that about 75% of the total structure is made from composite materials. Similarly, components in GE's aero engines, such as fan cases and compressor blades, are made from carbon fiber composites, delivering a 20% reduction in operational costs and a 15% decrease in emissions. The use of composite fan cases alone reduces aircraft weight by 180 kg compared to aluminum alternatives[2]. The cabin pressure bulkhead is an important structure in aircraft fuselage designs, and it is responsible to withstand significant internal pressure changes at high altitudes [3]. The

pressure bulkhead in the modern aircrafts, like the Airbus A350, is a crucial component which is made of carbon fiber reinforced plastics (CFRP), and its production needs high positioning accuracy to avoid defects such as wrinkles and bridging, which may lead to structural failures [4].

Despite the advantages of composites, traditional methods for manufacturing these parts, such as manual lay-up or tape-laying, are labor-intensive, time-consuming, and pose safety hazards. These processes have significant material waste and suffer from low repeatability. Operators should be expertise in handling these materials and ensure the strict safety protocols are considered, especially in industries like aerospace, where defects could lead to catastrophic failures [5]. Alternative automated techniques, such as Resin Transfer Molding (RTM), Automated Tape Laying (ATL), and Automated Fiber Placement (AFP), have been developed to make composite components more competitive with milled parts with a better repeatability. These automated methods have significantly enhanced composite manufacturing by improving material deposition speed, repeatability, compaction, waste reduction, and the seamless transition from design to production in comparison with the traditional methods. RTM also can inject a liquid-phase composite matrix into a mold filled with preformed fabric. However, one of its main drawbacks in these methods compared to manual lay-up, is the relatively low fiber volume fraction achieved[6].

ATL employs wide prepreg tape to be applied directly to the mold surface using a composite tape lay-up tool mounted on a large robotic arm. Key parameters in this method are the lay-up speed, tape temperature, and tension which are closely regulated during the process. The difference between thermoset tapes and thermoplastic tapes is that thermosets typically require a post-lay-up autoclave, while thermoplastic tapes do not. The tow-placement technique used in ATL has saved of up to 50% of the cost and scrap reduction of 75% in military applications, according to extensive testing on various large composite structures [7].

AFP operates similarly to ATL, but the difference is that instead of laying up a single wide tape, it deposits narrow bands of prepreg composite tows side by side onto the mold to create the part. This approach reduces the incidence of fiber wrinkling while maintaining a reasonable overall lay-up width [5]

Robotic systems in AFP are evaluated based on their kinematic capabilities, including workspace, singularities, and manipulability aside the maximum load that they can handle. In cooperative multi-robot systems, each robot's kinematic model should show a good capability

working synchronously without collisions or mechanical interference with each other[8]. The automated techniques like Automated Fiber Placement (AFP) and Automated Tape Laying (ATL) machines instead of the traditional methods have emerged as production methods capable of meeting industry requirements that were previously unattainable with traditional approaches. In addition, using the standardized tape sizing formats for conventional prepreg systems now has facilitated the process automation. Research and interest in automated composite manufacturing has been started in the 1960s with the introduction of automated filament winding systems and the development of AFP/ATL systems in the 1970s[8].

The introduction of parallel robots, which offer better stiffness and positioning precision while having a good degree of freedom, has enhanced the AFP system's ability to manufacture complex geometries [9].

An AFP machine is composed of two key elements which are a robotic arm and a manufacturing head. The robotic arm is responsible for positioning the fiber placement head specifically the end-effector at specific, pre-determined locations within the workspace on the mold, guiding it to move with remarkable precision on the defined path as shown in Figure 1.2. As the arm moves the head, the manufacturing end-effector simultaneously applies pressure to the composite tapes to be laid onto the part. This coordinated action ensures that the tapes are correctly positioned and securely bonded to the substrate to form the desired composite structure [12]. Figure 1.1 shows a case where the tool can rotate and the serial manipulator deposits the fiber.

Figure 1.1 Image of a robotic arm used in industry [12]

Figure 1.2 shows visual examples of AFP machines in action at Concordia University. It includes a photo of an AFP machine equipped with a thermoplastic head, demonstrating the integration of the heat source and its role in the manufacturing process. Also depicted is a smaller AFP machine with a thermoset head, highlighting the machine's adaptability to different composite materials and production needs[13].

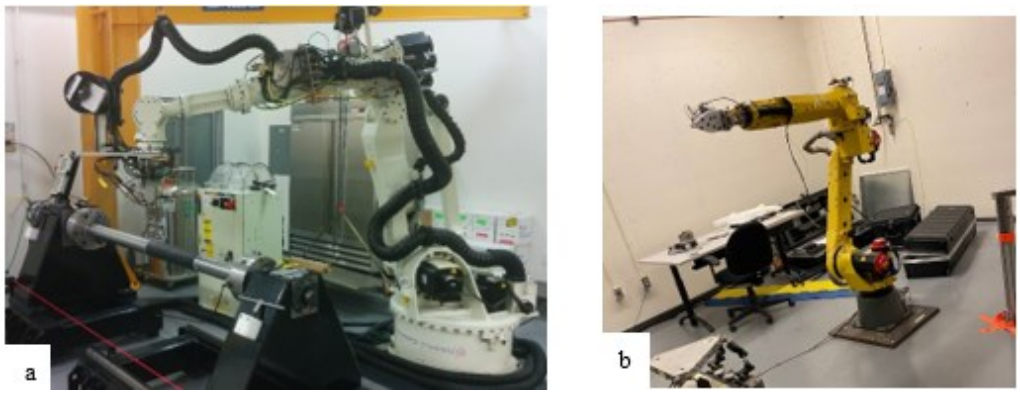

Figure 1.2 a) AFP machine at Concordia University [14] and b) the small-size AFP head [13]

The AFP at Concordia University's CONCOM (Concordia Center for Composites) laboratory consists of a ZX130L Kawasaki 6-axis articulated robot arm and it can handle a payload that is up to 125 kg. The robot arm can be equipped with two interchangeable heads produced by Trelleborg for a thermoplastic (TP) fiber placement head or an independent tow control (ITC) thermoset (TS) head [15]. Additionally, Figure 1.3.b shows a small-sized AFP head, designed specifically for laying thermoset layers on V-shaped structures and tight angles, which is connected to a Fanuc M20-iA robot located at Concordia University's Robotics Lab[13]. As the usage of the AFP has grown in the past years as one of the automated manufacturing techniques, they play an important role in producing large and complex parts (e.g. curved structures) while meeting high quality requirements [8],[16].

1.2 Problems and Solutions

Tooling plays a vital role in AFP deposition, determining the shape of the laid preforms as they are commonly large structures made of metal or cured composite's part and require large upfront investment, contributing significantly towards the overall cost of composites parts [17].

This is the case in aerospace asking for the high-quality parts with shape accuracy and good laminate quality. Tooling geometry and surface tribology significantly impacts layup quality and the likelihood of manufacturing defects in the final product. Defects related to tape placement on complex geometries (e.g., angle deviations, gaps, and overlaps) lead to reduced mechanical properties. These defects are key limitations of AFP and can reduce the mechanical performance of the product by up to 25% [18].

Now there is a need for manufacturing components that have complex shapes, contours, and curves. These geometries come with challenges for fiber placement, especially in terms of the path planning and maintaining high accuracy when the fibers are laid down across all surfaces. Most of the current AFP machines are not capable of manufacturing the shapes with complex curvatures, tubes with T shape or Y shape or tube with flanges having circular shape. To be able to expand the manufacture capabilities of AFP machines in a simple and low-cost way, it is necessary to increase the number of DOF of the robotic system to manufacture complex parts [19]. Most common problem that occurs in composites produced by AFP machines are shown in Figure 1.3 [20]. Gaps and overlaps that are not designed for the fiber placement and happen due to the incapability of the robots, will affect the parts mechanical properties[20],[21].

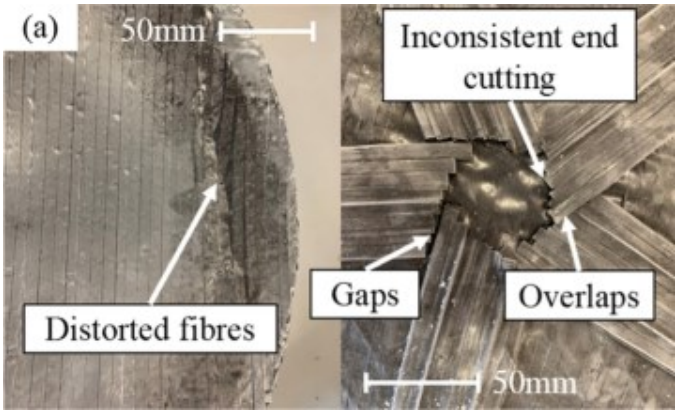

Figure 1.3 Gaps, overlaps and inconsistent fiber end cuttings in composites produced by AFP machines[20]

For the inspection of these problems there is a tolerance definition. In Figure1.5 the accepted tolerance range has been shown in part a, an acceptable composite part is shown in part b while an unacceptable composite part is shown in in part c [22]. There are two fibers that are positioned outside of the tolerance range and makes the part to be inacceptable. These problems occur based on the robot's incapability of precise positioning and mainly due to the robot's low precision at

certain points in its workspace [23]. The accuracy provided by manufactures is only the nominal value and the real accuracy of a robotic arm fluctuates in the workspace based on the robot's configuration. These fluctuations cause the problems shown in Figure 1.3 and Figure 1.4 [20], [22], [23].

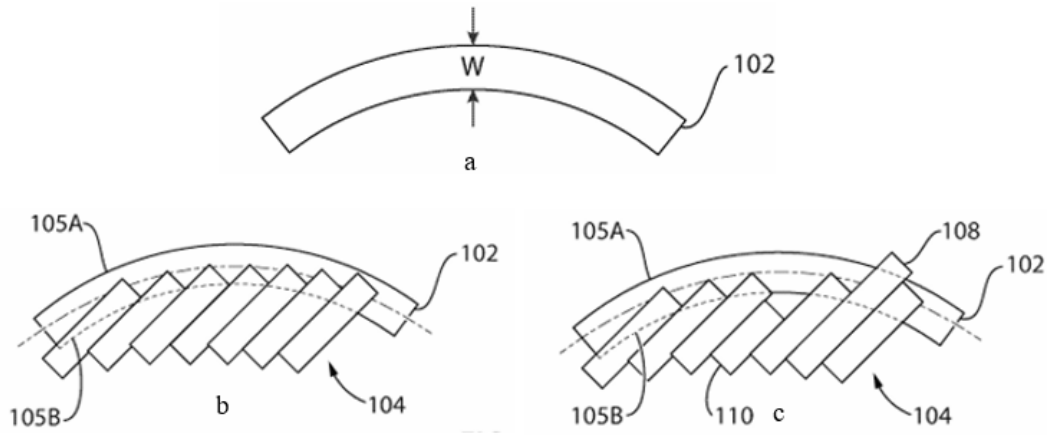

Figure 1.4 a) illustrates an exemplary tolerance window as an inspection feature for tow location, b) composite component that is compliant, c) composite component that is non-compliant [22]

Using two robots, one to hold the mold and the other one to do the fiber placement is a solution that is used nowadays but the problem is that as shown Figure 1.5, the second robot is usually a mandrel or a robot with 1 DOF [24].

Figure 1.5 A serial manipulator and a mandrel doing a fiber placement [19],[24]

Another solution is the utilization of the 6 DOF parallel robot as mandrel holder to collaborate with AFP machines due to its better stiffness and their precise positioning capability [25], as shown in Figure 1.6. However, this type of collaboration is not that common in industry and only a rotary robot is mainly used.

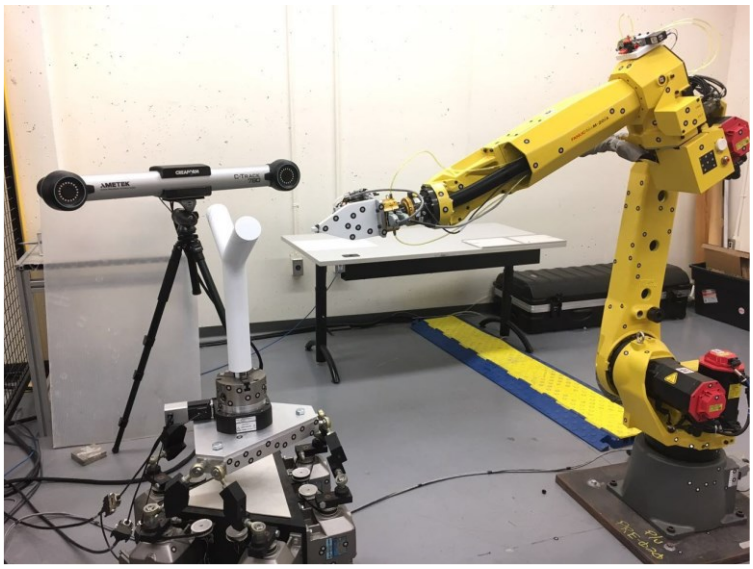

Figure 1.6 Cooperative AFP System Setup at Concordia University [19]

1.3 Scope and Objectives

The thesis aims to improve the quality and accuracy of fiber placement on complex geometric structures, such as a bulkhead, by taking advantage of the dual robot system including a Fanuc M-20iA and an additional robot to increase degrees of freedom (DOF) of the system and improve the manipulability of the robotic arm. The scope includes the proper path planning for manufacturing a composite product i.e. a pressure bulkhead and using the simulation to verify the designed path and improvement of the quality of the fiber placement process. In addition, the project will meet the challenges of trajectory planning, collision avoidance, and singularity mitigation in a dual-robot setup.

This thesis aims to demonstrate that using two robotic arms for Automated Fiber Placement (AFP) on complex geometric surfaces can significantly improve the quality of fiber placement if the second robot has multi degrees of freedom, for instance a parallel robot with 6 DOF. The key is that increasing the degrees of freedom (DOF) using dual robots enhances the precision and efficiency of the process, particularly for complex shapes where single-robot systems may face limitations and, dual robots give more control during the operation.

A critical factor in this study is the manipulability factor, which is used to compare different configurations and combinations of robots. The goal is to quantify how the addition of degrees of

freedom improves the fiber placement quality in various scenarios by comparing their manipulability factors. By addressing the challenges such as trajectory planning, collision avoidance, and singularity mitigation, the project seeks to optimize the process for more accurate and reliable fiber placement on intricate surfaces and to pave a path for adopting multiple robots in composite manufacturing industry.

1.4 Publications

A. Saboukhi, Y. Hedayatnasab, S. V. Hoa, W. -F. Xie and F. Shadmehri, "Toward a compact AFP head capable of performing V-shape structures: Design and Implementation," 2024 IEEE International Conference on Advanced Intelligent Mechatronics (AIM), Boston, MA, USA, 2024, pp. 1038-1043

1.5 Thesis Organization

This thesis consists of 5 chapters. The outline of the thesis is given as follows.

- Chapter 1 offers an introductory overview of composite materials and the AFP machine. It examines the principal shortcomings of existing systems and puts forward possible remedies. Furthermore, the chapter delineates the thesis's scope and objectives.
- Chapter 2 presents a comprehensive literature review on the current state-of-the-art in AFP machines, integrated AFP machines, serial and parallel manipulators, and path planning methods, as well as simulation software.
- Chapter 3 presents the kinematics of the robots utilized in each scenario, as well as the geometric and positioning design based on the robots.
- Chapter 4 is dedicated to the implementation of the simulation using RoboDK software. It commences with the configuration of the simulation environment, explaining the system specifications and initial conditions. Thereafter, the methodology used to implement algorithms is presented. In the end, the results of the simulation are analyzed based on theoretical knowledge to evaluate performance.
- Chapter 5 outlines the conclusions and contributions of the thesis and suggests topics for further research.

Chapter 2

2 Literature Review:

A literature survey on the relevant studies regarding the current AFP machines and path planning and usage of serial manipulators for AFP machines is conducted in this chapter.

2.1 AFP Machines

In the early stages of composite manufacturing, processes were largely reliant on manual lay-up, followed by consolidation and curing in an autoclave. While the hand lay-up method offered enhanced compatibility in comparison with other techniques for specific intricate components, it was inherently time-consuming and heavily dependent on the expertise of technicians. This led to elevated manufacturing costs and variable production quality [26], [27] . Consequently, research studies have been conducted with the objective of developing automated composite manufacturing processes, while simultaneously enhancing the associated devices and technologies to achieve enhanced flexibility, multifunctionality, and efficiency. Two principal automation technologies are currently employed in the manufacture of large composite components which are automated tape laying (ATL) and automated fiber placement (AFP). The implementation of these automated processes has resulted in a notable reduction in labor-intensive activities and a considerable enhancement in manufacturing efficiency [28].

The ability to individually control each fiber allows to produce fiber bands of varying widths, which enables the creation of more intricate surfaces with greater precision and reduce material waste when compared to ATL processes [27].

Since 1997, researchers have been engaged in a systematic program of advancement in the technologies of composite manufacturing. The program has yielded a series of innovative explorations and comprehensive discussions of findings and perspectives. Grant et al. [28] conducted an extensive evaluation of processing techniques, including ATL, AFP, and winding machines, and observed an increasing preference for cost-effective automation in the realm of composite manufacturing. Dirk et al. [29] undertook a study which focused on the development of prepreg layup using AFP and ATL. Furthermore, they provided a comparison of the constraints and limitations of these machines. Despite its pivotal function in the quality of composite forming, there has been a lack of attention paid to the review of the mechanisms, mechanical structures, and research advancements pertinent to AFP systems. Consequently, Zhang et al. [27] have provided an overview of the mechanisms of automated composite manufacturing methods, advantages and disadvantages, and the critical process parameter control of AFP, including minimal fiber length, tension, and compaction force.

2.1.2 History and development of AFP

The automated fabrication of fibrous composites is commonly achieved using techniques such as AFP, ATL, filament winding (FW), and resin transfer molding. The FW machine was initially developed in the 1940s and was subsequently employed with considerable success in the fabrication of rocket motor cases. Subsequently, in the late 1960s, ATL was introduced for several military programs, including the manufacture of missile shells [28].

In the late 1970s, in response to the expansion of the aircraft manufacturing sector and the advent of advanced composite materials, a new concept emerged: the AFP process. This initiative was designed to overcome the limitations of the existing FW and ATL processing techniques. Concurrently, advancements in fiber reinforcement manufacturing enabled the advent of AFP system manufacturers, which resulted in substantial economic advantages.

Hercules Aerospace (ATK) and Cincinnati Machine were among the first American manufacturing companies to develop their AFP systems in-house during the early 1980s. Subsequently, a multitude of manufacturers and institutions from the United States, including

Automated Dynamic, MAG Cincinnati, Ingersoll Machine Tool, Electro impact, Inc., ATK, and Accudyne, have engaged extensively in the advancement of AFP systems. Their European counterparts have also played a pivotal role in this field, with notable contributions from MTorres in Spain, Mikrosam in Macedonia, and Coriolis in France [27]. Automated Dynamic Cooperation and Coriolis employ robotic technology to facilitate the provision of advanced fiber preparation (AFP) systems, which are utilized for research and design studies conducted by institutions of higher education. The systems can accommodate up to eight-bundle fibers. CNC machine tool technology has been instrumental in meeting the demands of various industrial sectors. Prominent examples include the production of aircraft components, such as wing stringers, fuselage sections, panels, and pressure bulkheads, by Cincinnati Ingersoll, Accudyne, ATK, MTorres, and Mikrosam. Electrompact, MTorres, Automated Dynamics, and Coriolis have developed a range of systems based on a modular design approach with high levels of integration. In comparison to automated tape laying (ATL) systems, AFP systems utilize a greater number of individuals slit prepreg fibers, typically with a width of 0.125–0.500 in, to manufacture more complex curved surfaces and small structures. This allows to produce components with greater intricacy and precision, which can also be customized according to customer specifications [30].

Regarding the flexibility of the system and cost efficiency, while industrial robots may offer a viable alternative to gantry units regarding the implementation of modular equipment, the initial expenditure associated with the AFP manufacturing process remains considerable. To reduce the cost of production, an alternative approach is to enhance productivity and functionality once the requisite specifications have been fulfilled. As an example, Izco et al. [31] constructed an AFP machine with the capacity to clamp, cut, and restart (CCR) isolated fiber strands at high velocity, thereby facilitating a substantial enhancement in efficiency exceeding 45 kg/h material deposition. This machine is currently available for purchase on the market. Moreover, a research team from Italy has devised a novel tape placement technique that employs two compacting rollers to facilitate alternative deposition movement. The system is designed to facilitate the continuous deposition of structures with variable thicknesses [32]. In 2017, Add Composites Company initiated the development of AFP tool heads with the objective of investigating the potential for combining AFP with 3D printing, or integrating multiple processes into a single automated tool with the aim of reducing costs [33].

A typical AFP system comprises three principal components. First an AFP mechanism, second one is computer-aided manufacturing (CAM) software and finally a moving platform. The moving platform may take the form of an industrial robot [34], or it may assume the role of a gantry, a column configuration machine, or a bespoke design [35]. The CAM software is employed for the purposes of trajectory planning and fiber path generation, whereas the AFP mechanism is tasked with the independent delivery of fiber and the regulation of process variables to ensure a precise fiber layup.

Accordingly, the primary areas of focus for the AFP manufacturing industry encompass mechanical design, process parameter control, and trajectory planning technologies [36]. These components are fundamental to coordinating the functional mechanisms that provide independent fibers with individual feed speed and tension control to facilitate the formation of composite parts. However, the functional mechanisms within AFP systems are inherently complex. This is due to their involvement of numerous degrees of freedom, the necessity for control of multiple actuators, and the presence of intricate coupling constraints. It is therefore essential to have a clear understanding of these modules and their interactions to describe an AFP process in an effective manner.

2.2 Avoiding Singularity in Manipulators

In the field of robotics, a singularity is defined as the loss of one or more degrees of freedom experienced by a manipulator, which subsequently affects its capacity to move or exert force in specific directions. In most cases, singularities manifest within two distinct operational contexts: at the boundaries of a robot's workspace and within its interior. A practical example of a singularity can be observed in the case of a two-link arm that is fully extended or folded back on itself. In such instances, the capacity for movement is confined to a single direction, resulting in the Jacobian losing rank. It is imperative that control systems take these configurations into account to prevent excessive joint motion and potential damage to the system.[37]

A workspace boundary singularity which is shown in Figure 2.1 occurs when the robot's end-effector reaches the limits of its operational space. For example, if the manipulator is fully

extended or folded back, the end-effector is constrained to a single direction of movement, effectively reducing the robot's degrees of freedom and limiting its range of motion.

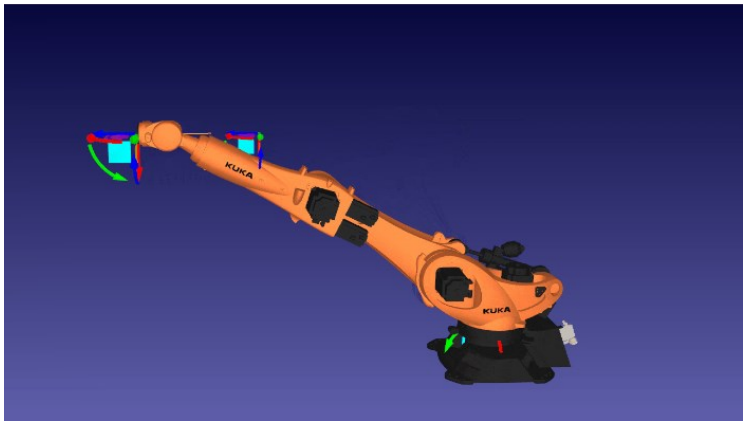

Figure 2.1 Robot in a boundary singularity pose [38]

Workspace-interior singularities are defined as occurring inside the robot's workspace when two or more joint axes align. This phenomenon has the effect of rendering certain movements in Cartesian space impossible. Despite the manipulator's continued ability to move, specific directional movements are restricted or result in exceedingly high joint speeds as the robot approaches the singularity. Most of the industrial six-degree-of-freedom (6 DoF) robots possess three joints in their wrist (joints 4-6). In many robots, the axes of these three joints converge at a singular point. In such cases, the wrist singularity occurs when joints 4 and 6 become coincident. For instance, Figure 2.2 shows a robot when the three wrist joint axes do not converge at a single point, therefore they cannot become coincident. In this instance, singularity occurs when the axes of joints 4 and 6 become parallel.[38]

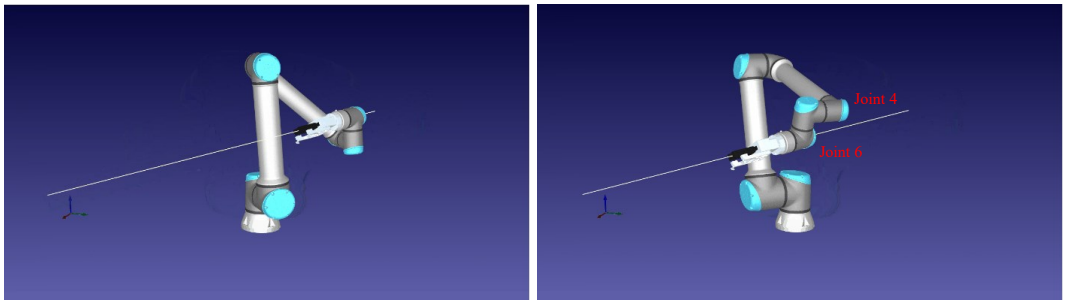

Figure 2.2 A robot when the axes of Joints 4 and 6 become parallel [38]

A significant mathematical instrument for identifying singularities is the Jacobian matrix, which translates joint velocities into end-effector velocities. When the determinant of the Jacobian

is equal to zero, the manipulator is at a singularity, signifying that its capacity to regulate motion in all directions is constrained. Singularities result in the inverse of the Jacobian becoming undefined, which in turn gives rise to instability in control, such as joint velocities becoming infinite.

Several studies focused on developing methods to analyze and avoid singularities in robotic manipulators. A common approach is to partition the Jacobian matrix into smaller submatrices to isolate and manage singularities more effectively. This enables the planning of trajectories that steer clear of singular configurations, thereby improving system performance [39] .

Trajectory Optimization for Singularity Avoidance is a method for optimizing trajectories in manipulator motion planning that can also effectively address singularities. By calculating the kinematics of singular configurations, potential functions can be employed to direct the manipulator along paths that are free of singular poses. This method has been demonstrated to reduce computational times and enhance trajectory efficiency [40].

A further method to avoid singularities is to utilize redundant degrees of freedom. An increase in the degrees of freedom (DOF) in manipulators can assist in the avoidance of singularities without the necessity of additional motors. This approach enhances the manipulators' agility, rendering them more adaptable in constrained environments and improving their ability to avoid singular configurations [41].

2.3 Manipulability in Robotic Systems

Manipulability is a critical factor in robotic systems used for fiber placement. Manipulability is a measure that indicates the robot’s capability of moving in different directions when the robot is in a position. It shows how well a manipulator can generate velocities aside showing the end-effector in response to the joint movements. Manipulability is derived from the Jacobian matrix of the manipulator. This is a critical measure to avoid singularities because a decrease in the manipulability comes with an approach to a singular configuration of the manipulator [42].

Maximizing the Manipulability as a control objective is another effective method to avoid singularities. By framing manipulability as an objective function, real-time adjustments can be made to optimize the manipulator's performance without falling into singular poses. Techniques such as using dynamic neural networks have been proposed to achieve such real-time optimization [43].

In the context of Automated Fiber Placement (AFP), the achievement of a high-quality composite layup is significantly influenced by the manipulability of the robot, particularly in maintaining the perpendicular orientation of the fiber placement tool. High manipulability allows the AFP head to accurately follow the contours of complex surfaces and maintain the perpendicularity of the roller to the tool surface, which is critical for consistent material deposition and it is shown that when the robotic system maintains high manipulability, it reduces defects such as fiber wrinkles, misalignment, and defects at critical points such as bends, thereby improving the structural integrity and mechanical performance of the composite [44].

This capability is important in aerospace applications, where structural components require minimal variation in fiber alignment to meet the standards. Misalignments and gaps can act as stress concentrators; by reducing these problems, the composite layup achieves better load distribution across the fibers, which lead to a better stiffness and strength [21].

2.4 Dual Robots or Cobots

Dual Robots (Cobots) is a collaboration between two or more robots to do a task such as fiber placement. This collaborative robotics system has significantly grown in industrial applications because of their flexibility and efficiency. These systems have been effectively employed in prototype manufacturing environments, where the need for flexibility in tool handling and collision avoidance is crucial [45]. Dual-arm robots have been integrated into smart factories to enable collaborative and disassembly tasks. The incorporation of dual tooling and advanced vision systems enhances dexterity and control, making these collaborative robots a good choice for highly complex tasks who requires simultaneous manipulation of components[46]. The dual-arm robotic systems with motion and force control capabilities, allowing for coordinated two-handed grasping

and manipulation have been developed with research, which further expands their application in handling varied shapes and objects [47]. The most significant advantage of dual robot systems is their ability to encounter singularities in comparison to a single robot arm. When a single robot arm approaches a singular configuration, its abilities will be lower, however dual-arm robots with their additional degrees of freedom, can exchange their tasks and coordinate motion to avoid these singular regions. Techniques such as artificial ellipses in the Jacobian matrix have been used and shown to detect and avoid singularities, thus maintaining stability and control during operations [48].

2.5 Pressure Bulkheads and Quality Issues

Pressure bulkheads are critical structural components in aerospace applications. Fiber-reinforced composites are increasingly being used for pressure bulkheads due to their strength-to-weight ratio. Quality issues in pressure bulkheads often arise from poor fiber placement or improper interlaminar bonding. Inadequate compaction pressure during fiber placement can result in bridging, which compromises the bulkhead's structural integrity [49]. Robotic systems can help reduce defects in pressure bulkheads by ensuring consistent material application. Fiber orientation is a critical factor in their structural performance. The defects caused during fiber-placement, such as voids or gaps between layers, can significantly reduce the mechanical strength of pressure bulkheads. Automated systems using precise control algorithms reduce the probability of defects during the fiber placement process. The development of compacting roller ensures that the layers are laid with appropriate adhesion and bonding.

2.6 Path planning methods for Fiber Placement

The objective of path planning in the robotic fiber placement process is to determine the trajectory to be followed by the robot, the number of fiber tows to be performed, and the requisite gap and overlap between them. This is based on the geometric characteristics of the component, the desired precision, and the layer direction. This process entails the completion of several specific tasks, including ensuring that the tangent vector of each point within the laying path aligns with the desired layer direction, thus ensuring that the requisite mechanical engineering properties are

met; maintaining the distance between the tows within the permitted maximum tow width, while simultaneously maintaining component forming accuracy; and acquiring the normal vector of the structure surface, which is utilized to control the pose of the placement head and ensure its continued perpendicular alignment with the surface. Path planning methods are typically classified into two categories of fixed or variable angle path planning [50]. These algorithms are classified according to whether the fiber laying angle is fixed or varies. In this context, the term "fiber laying angle" is defined as the angle between the direction of the fiber tow at a specific point and a reference line. Should the angle of deviation between the intended trajectory and the reference line remain constant, this algorithm is designated the "fixed angle path planning method". In the event of a change in angle, the algorithm is classified as belonging to the variable angle category. It is important to note that the classification is contingent upon the assumption that layers are being deposited upon simple shapes. It is therefore necessary to introduce an additional classification for the case of more complex structures, such as the internal surface of a curved structures, for which the sole feasible methodology is to rotate the robotic apparatus and transform the perpendicular normal vector into an angular normal vector [51], [52], [53]. In Figure 2.3 the fixed angle path planning trajectory is shown.

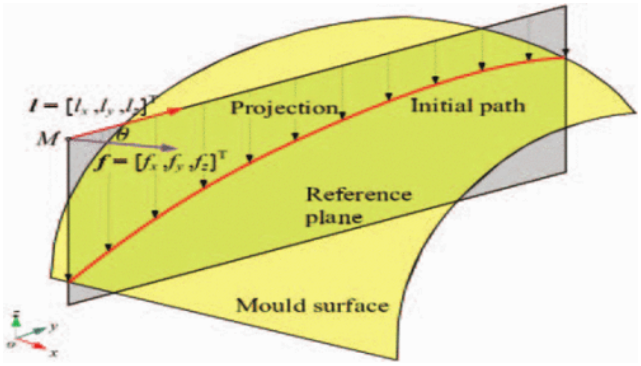

Figure 2.3 the fixed angle path planning method [51]

In Figure 2.3, M is a specific point on the initial path, which is shown with red, before it is projected onto the surface and represents the location in the reference frame. The vector I represents the initial path tangent or direction in the reference plane. It determines how the initial path aligns before being projected and vector F shows the orientation or forces associated with the path on the surface. The θ is the angle between the I and F vector.

2.7 Simulation

It is evident that simulations in simulation environments are essential prior to input the path to the robot. This is to guarantee the safety of the operator and equipment.

A variety of specialized simulation software is available for different robotics companies, including RoboGuide for Fanuc and Robo Studio for ABB. RoboDK [54] is a software package for industrial robots that permits users to construct three-dimensional models of their manufacturing environments and to simulate the movements and activities of robots. The software is compatible with a multitude of robot types, including Articulated, SCARA, Delta, and others, from prominent manufacturers such as ABB, Fanuc, Kuka, and Universal Robots. It also incorporates a repository of robot models and tools for the generation of robot programs in an array of programming languages. RoboDK is a prevalent choice in sectors such as automotive manufacturing, aerospace, and electronics assembly, where it is leveraged to enhance efficiency and precision in robot programming and deployment. For instance, Pieskä et al. [55] employed the RoboDK to conduct research on the simulation of collaborative robots for small-scale manufacturing. Furthermore, RoboDK provides a collision control feature, which has been employed by Sivasankaran et al.[56] to develop a collision mapping planner for simulating robot kinematic motions. Consequently, RoboDK has been utilized in the proposed thesis to simulate the collaboration between two robots and utilize the manipulability factor to ensure the robot's configuration for the fiber placement process on a dome-type geometry.

2.8 Summary

This chapter presents a comprehensive review of the literature on automated fiber placement (AFP), which is widely used in industry. It emphasizes the critical role of precision and the consequences of accuracy of robots in manufacturing outcomes. The review underscores the value of simulation tools, such as RoboDK, in visualizing the AFP process, addressing challenges like singularities, and exploring manipulability as a potential solution. It further highlights the need for continued advancements to enhance the precision and adaptability of AFP systems.

The following chapter will focus on the kinematic analysis of the robotic systems employed for fiber placement on a bulkhead.

Chapter 3

3 Kinematics Analysis and Systems Design of The Cooperative AFP Systems:

3.1 Introduction to Robots

To accomplish the fiber placement on the bulkhead, it is necessary to create a simulation environment based on the limitations and specifications of the robots that are used. In order to do the fiber placement on the bulkhead with multiple robots, the serial manipulator with the proper head for the fiber placement and their constraints should be considered. Based on the second robot's specifications, a proper design of a mold is necessary in order to be mounted on top of the second robot.

The thesis aims to show the improvements of fiber placement by using multiple robots and the impact of the robots and their degrees of freedom on their reachability and precision based on manipulability factor. The AFP head is installed on the end effector of a Fanuc-M20-iA robot in Concordia university. The Fanuc robot will be the main robot to do the fiber placement, and it will remain unchanged. The bulkhead, which is the part to be produced, will be mounted in three different setups: on two different rotary robots, each with one degree of freedom but differing in mounting orientation on KUKA KP1-V 500 and ABB IRBP L600 L1250, and on a parallel robot (PI H-840-D2A) equipped with a rotary stage, allowing six degrees of freedom for the part. The combination of The Fanuc M20-iA and the KUKA KP1-V 500, ABB IRBP L600 L1250 and PI

H-840-D2A will be the three different scenarios that will be compared in this thesis in terms of the manipulability of the Fanuc M20-iA during the fiber placement on the bulkhead.

3.2 Fanuc M20-iA

The FANUC M-20iA is an advanced industrial robot designed for a wide range of manufacturing applications, including material handling, machine tending, and automation tasks. The FANUC M-20iA is an industrial robot known for its versatility, precision, and reliability in various manufacturing applications, including robotic fiber placement[57]. The design allows movements in confined spaces. The robot's advanced kinematics and control systems make it an ideal candidate for complex tasks requiring high degrees of freedom and precision[57] .

Figure 3.1 FANUC M-20iA[58]

The FANUC M-20iA's lightweight construction can handle a maximum payload of 20 kilograms and a reach of 1,811 millimeters while having flexibility and strength. Fanuc M20-iA also has a high repeatability of ±0.08 millimeters that ensures consistent precision, which is critical in processes like robotic fiber placement where exacting standards are essential [57].

In the field of robotic fiber placement, the FANUC M-20iA has demonstrated exceptional capabilities because the process involves the precise laying of fiber materials onto molds or forms to create composite structures, which is critical in industries such as aerospace, automotive, and renewable energy [57].

The robot's payload capacity accommodates end-of-arm tooling necessary for fiber placement, such as creels, cutters, and compacting rollers, without compromising performance. Additionally, the FANUC M-20iA can be seamlessly integrated with positioners like the KUKA KP1-V 500 or ABB IRBP L600 L1250 as used in this project. This integration enhances its ability to access different areas of the workpiece by manipulating its position and orientation, providing robots with enhanced access to complex geometries and hard-to-reach areas.

Implementing the FANUC M-20iA in fiber placement applications yields several benefits that directly impact productivity and product quality. High-speed movements and efficient path planning reduce the time required to complete fiber layups and increase the throughput. Consistent and precise fiber placement enhances the mechanical properties of composite parts, resulting in higher quality products with better performance characteristics. The robot's adaptability allows for quick reconfiguration of production setups, accommodating different part designs without extensive downtime. Automating the fiber placement process reduces the need for manual labor in potentially hazardous environments, improving workplace safety and ergonomics.

Integrating the FANUC M-20iA into a robotic fiber placement system involves careful consideration of several factors to optimize performance. Designing custom end-of-arm tooling that meets the specific requirements of fiber placement is essential for successful operation. Utilizing advanced programming techniques and simulation software allows for planning efficient and collision-free paths, which is crucial in complex fiber layup processes. Coordinating the movements of the robot with positioners enhances accessibility and allows for continuous fiber placement on complex geometries. Implementing robust control systems that manage the interactions between the robot, tooling, and positioners ensures smooth operation and minimizes the risk of errors or downtime.

Several industries have successfully employed the FANUC M-20iA in fiber placement applications, demonstrating its effectiveness. In the aerospace industry, manufacturers use the M-20iA to produce composite aircraft components where precision and repeatability are critical for meeting stringent safety and performance standards. In the automotive sector, robots contribute to the production of lightweight composite parts that improve fuel efficiency and reduce emissions. In wind energy, the M-20iA assists in laying fibers accurately to ensure the structural integrity and longevity of wind turbine blades [53].

In conclusion, the FANUC M-20iA robot embodies a blend of advanced technology and practical design, making it a valuable asset in modern manufacturing environments. Its capabilities in precision, speed, and flexibility enable it to meet the demanding requirements of robotic fiber placement and other complex tasks. By integrating the M-20iA into fiber placement systems, manufacturers can achieve higher levels of efficiency, product quality, and operational flexibility. The robot's adaptability to various applications and seamless integration with other automation components, such as positioners and advanced control systems, positions it as a key contributor to the advancement of automated composite manufacturing technologies.

As industries continue to evolve and seek innovative solutions to enhance productivity and competitiveness, the FANUC M-20iA stands as a testament to the potential of robotic automation in meeting these challenges. Its role in facilitating precise and efficient fiber placement underscores the transformative impact that advanced robotics can have on manufacturing processes, driving progress and innovation across multiple sectors.

3.2.1 The AFP Head on the Fanuc M20-iA

A small-size AFP head is used for this project which was aimed at manufacturing complex composite components and having the ability to deposit layers on a flat surface. The AFP head is attached as a tool to the end effector of a FANUC M-20iA with 6-degree-of-freedom (DOF) serial robot. It utilizes a thermoset tow of ¼ inch width and is designed to be small in size and as optimum as possible to ensure optimal maneuverability to place the Fiber on the inner surface of V-shaped structures [13].

Some research work has been dedicated to the design and build of an AFP head to be smaller than the commercially available AFP machines which are designed to manufacture simple structures like shallow shells or tubes. The current commercial AFP head are large and bulky; therefore, they are incapable of handling some applications with more complex shapes and in small workspaces[59]. Fig 3.2 shows the AFP head that has been installed and designed in Concordia laboratory. This robot was inspired by the thermoset AFP head designed by Trelleborg which is available at Concordia AFP lab. This head has been optimized to be able to work on tight corners.

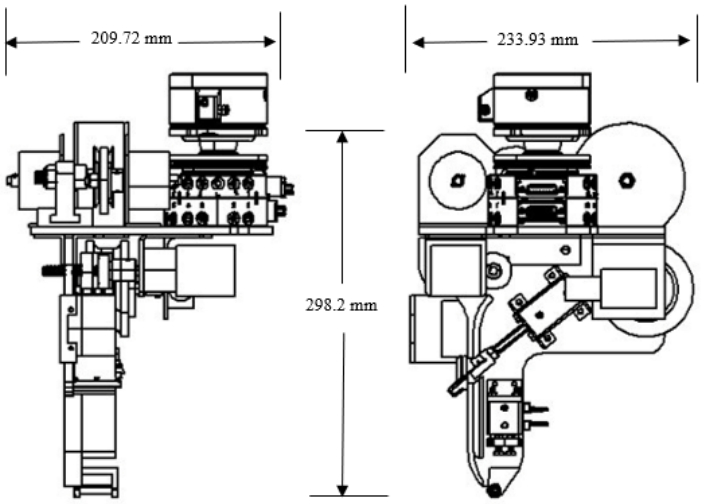

Figure 3.2 Dimensions of the AFP head [13]

Figure 3.3 shows the 3D design of the AFP head that has been designed in the CAD software packages and has been used in the simulation software (RoboDK) to check for collisions. Also, the different joints of the robot are shown in Figure 3.3.

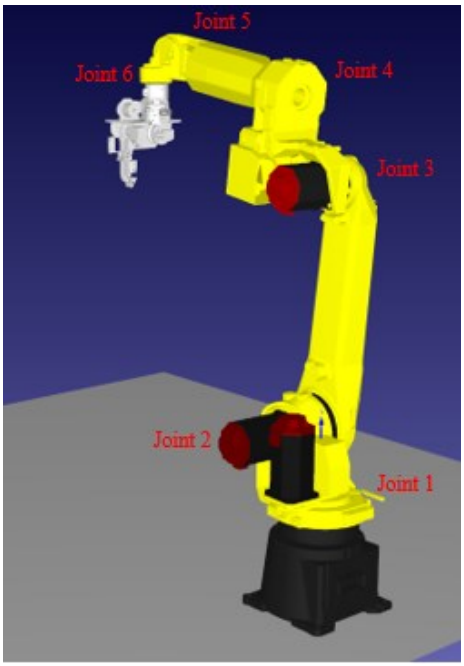

Figure 3.3 AFP head mounted on the Fanuc robot in simulation environment

3.2.2 Kinematic Modeling and Denavit-Hartenberg Parameters

To effectively utilize the FANUC M-20iA in simulation and control applications, it is crucial to develop a mathematical model representing its kinematic structure. The Denavit-Hartenberg (DH) convention is a systematic method for describing the geometry of serial-link manipulators, allowing for the derivation of transformation matrices that relate joint parameters to end-effector positions and orientations. Denavit-Hartenberg Parameters, known as DH parameters, consist of four key elements for each joint:

Link Length (a_i): The distance between the axes along the common normal.

Link Twist (α_i): The angle between the axes about the common normal.

Link Offset (d_i): The distance along the previous z-axis to the common normal.

Joint Angle (θ_i): The angle about the previous z-axis to the x-axis

The angle between the joint axes z_{i-1} and z_i is α_i, the length of link i is a_i, the distance between the o_{i-1} and o_i along z_i is $d_i$and the angle between x_{i-1} and x_i is θ_i which are shown in Figure 3.4.

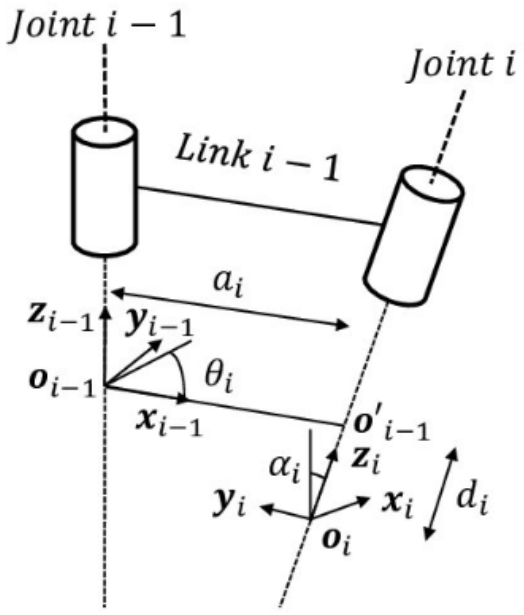

Figure 3.4 Calculation of the DH parameters for two joints [60]

The kinematic model of Fanuc M20-iA is obtained by using the D-H parameters. Its D-H parameters are shown in Table 3.1.[38]

Table 3.1 D-H parameters of Fanuc M20-iA

Joint i	α_i (degrees)	a_i(mm)	d_i(mm)	θ_i(degrees)
1	0	0	525	θ_1
2	-90	150	0	θ_2
3	0	790	0	θ_3
4	-90	250	835	θ_4
5	90	0	0	θ_5
6	-90	0	100	θ_6

The angles are given in degrees and will be converted to radians for calculations.

3.2.3 Jacobian Matrix Calculation

Each link's Transformation matrix T_i can be computed by using the DH parameters [61]

$$T_{i-1}^{i} = \begin{bmatrix} \cos\theta_i & -sin\theta_i \cos\theta_i & \sin\theta_i sin\alpha_i & \alpha_i \cos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_i & -\cos\theta_i\, sin\alpha_i & \alpha_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

This matrix gives the pose of each link relative to the previous link. The Jacobian matrix J relates joint velocities to end-effector velocities:

$$\dot{x} = J\,\dot{\theta} \tag{3.2}$$

where $\dot{x}$ is the end-effector velocity vector and $\dot{\theta}$ is the joint velocity vector. The Jacobian matrix is composed of:

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix} \tag{3.3}$$

where J_v (3x6) is Linear velocity Jacobian and J_ω (3x6) is the angular velocity Jacobian. For revolute joints, the i^{th} column of the Jacobian is calculated as below.

$$J_v^i = \frac{\partial P}{\partial \theta_i} \tag{3.4}$$

where P is the position of the end effector and the rotational part of Jacobian is derived as below:

$$J_\omega^i = Z_{i-1} \tag{3.5}$$

where Z_i is the z-axis of the coordinate frame of joint i. The linear velocity component can be calculated as below.

$$J_{vi} = z_{i-1} * (o_n - o_{i-1}) \tag{3.6}$$

Angular velocity component also is calculated.

$$J_{\omega i} = z_{i-1} \tag{3.7}$$

z_{i-1} stands for unit vector along the axis of joint i in the base frame, o_{i-1} for position vector of joint i in the base frame and o_n for position vector of the end-effector.

3.2.4 Singularity and Manipulability

As explained in chapter 2, singularities occur when the Jacobian matrix loses rank, leading to a loss of control in certain directions. When $\det(J) = 0$, the robot is at a singularity. To avoid

singularities which affect the quality of the robot's movement, it is essential to plan trajectories that steer them clear of singular configurations.

The manipulability measure ω provides an index of the robot's dexterity at a given configuration. Manipulability is a measure of how effectively a robot's end-effector can move or apply forces from a given position. It can be any number equal or higher than zero and high manipulability is ideal for flexibility and precision, while low manipulability indicates restricted or less stable movement, which is often near a singularity [61].

$$\omega = \sqrt{\det(J.J^T)} \tag{3.8}$$

3.3 KUKA KP1-V 500

The KUKA KP1-V 500 is a single-axis vertical rotary positioner renowned for its robust design and exceptional performance which has been shown in Figure 3.5. Designed to handle workpieces weighing up to 500 kilograms, this positioner offers a versatile solution for industries requiring precise manipulation of components. Its integration into robotic systems allows for the seamless rotation of workpieces around a vertical axis, providing robots with enhanced access to different sides of a part without necessitating repositioning or manual adjustments.

Figure 3.5 Isometric view of a KP1-V 500 [62]

The KP1-V 500's compact footprint makes it an ideal choice for facilities with limited space. Its floor-mounted design ensures stability during operations, while its high positioning accuracy guarantees repeatability and precision—essential factors in processes like fiber placement, where exacting standards are the norm [63].

One of the key advantages of the KP1-V 500 is its compatibility with KUKA robots and controllers. This compatibility streamlines the integration process, enabling synchronized movements between the robot and the positioner. As a result, programming becomes more straightforward, and operational efficiency is significantly enhanced.

In practical applications, the KP1-V 500 has demonstrated its value in improving the reachability of robots. By rotating the workpiece, the positioner allows the robot to access complex geometry and hard-to-reach areas, which is particularly beneficial in fiber placement tasks. This capability not only reduces cycle times but also enhances the quality and consistency of the fiber deposition process.

3.4 ABB IRBP L600 L1250

Complementing the capabilities of the KUKA KP1-V 500, the ABB IRBP L600 L1250 positioners offer even greater flexibility and load-handling capacity. The IRBP L series includes positioners capable of handling workpieces weighing up to 600 kilograms (L600) and 1,250 kilograms (L1250), making them suitable for heavy-duty applications.

Figure 3.6 Isometric view of 3.2.3 an ABB IRBP L600 L1250[64]

The IRBP L600 L1250 positioners are equipped with two rotational axes—one around a vertical axis and another around a horizontal axis. This dual-axis design provides an extensive range of motion, allowing for intricate positioning and orientation of large and complex workpieces. Such flexibility is essential in robotic fiber placement, where the ability to adjust the workpiece orientation can significantly impact the quality of the fiber layup.[64]

Integration with ABB robots and controllers is seamless, thanks to the positioners' design, which ensures compatibility and synchronization. The robust construction of the IRBP L600 L1250 ensures stability during operations involving heavy loads, while their high positioning accuracy and repeatability meet the stringent requirements of precision-dependent processes.

In the context of fiber placement, the dual-axis rotation of the IRBP L600 L1250 enables robots to maintain optimal tool orientation throughout the process. This capability is critical when dealing with complex component geometries, as it allows for consistent fiber deposition even on curved or angled surfaces. By minimizing the need for multiple setups or manual repositioning, these positioners contribute to increased productivity and reduced operational costs.

3.5 The Parallel Robot

The PI H-840-D2A is a high-precision parallel kinematic positioning system designed for applications that require exact positioning, fast response, and multi-axis control. The H-840-D2A belongs to a class of Stewart Platforms, which utilize six actuators working in parallel to control the motion in all six degrees of freedom: translation along the X, Y, and Z axes, as well as rotation around these axes (pitch, yaw, and roll).

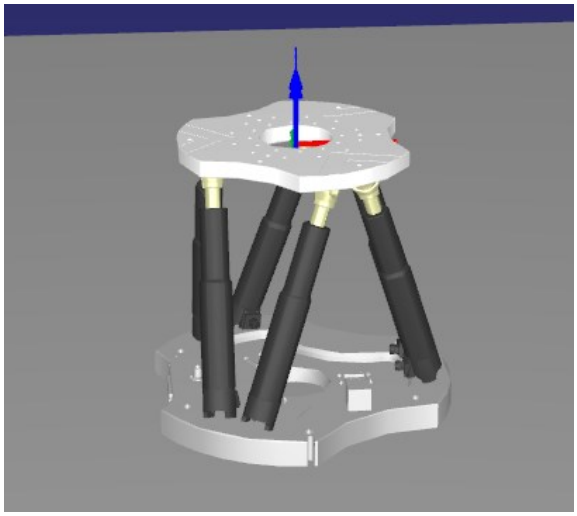

Figure 3.7 Isometric view of the PI H-840-D2A Parallel robot

It is widely used in fields such as aerospace, automation, and robotics for precision positioning and orientation adjustments. Key Features of the PI H-840-D2A are mentioned below.[65]

Unlike serial robots where actuators are arranged in a chain, the Stewart platform has a parallel structure where all actuators work together to control the platform's position and orientation. This allows for increased rigidity and precision.

The PI H-840-D2A can manipulate objects in all 6 degrees of freedom—three translational (X, Y, Z) and three rotational (pitch, roll, yaw). This makes it extremely versatile for tasks that require precise spatial manipulation. With its advanced actuators, the PI H-840-D2A offers high precision for positioning, which makes it suitable for tasks that require extremely fine adjustments, such as fiber optic alignment or micro-manipulation. Despite its small footprint, the platform has a high stiffness-to-weight ratio, providing accurate and stable movements even under varying loads. The system is designed for high dynamic performance, meaning it can execute rapid, precise adjustments without overshooting, making it well-suited for applications that require quick changes in positioning.

The PI H-840-D2A consists of a top and a bottom platform connected by six linear actuators. The actuators are controlled by precise motors, and their extension and contraction define the position and orientation of the top platform. The bottom platform is fixed, while the top platform moves in response to commands from a control system. The actuators work in parallel, ensuring the platform maintains stability while providing fine control over the platform's movements.

In the fiber placement simulation described in this thesis, the PI H-840-D2A acts as the base for the bulkhead. The bulkhead is positioned on the top platform, which moves and rotates under the control of the PI H-840-D2A to align the bulkhead optimally for fiber placement by the FANUC M-20iA robot. The platform's ability to manipulate the bulkhead in all 6 DOF allows for complex fiber placement patterns that would be difficult to achieve with a single robot alone. Additionally, by providing smooth, precise rotations, the PI H-840-D2A improves the overall accuracy of the fiber placement process.

It is also important to mention that in this thesis it has been suggested to place a rotary stage on top of the parallel robot in order to be able to rotate the part at the same time as the ability to

rotate the other roll, pitch and yaw rotations. This gives the capability of more movements for the bulkhead that is placed on the parallel robot.

Figure 3.8 shows the bulkhead that has been placed on the three robots in the simulation for the three scenarios that will be compared to find the best dual collaboration with Fanuc M20-iA for the fiber placement.

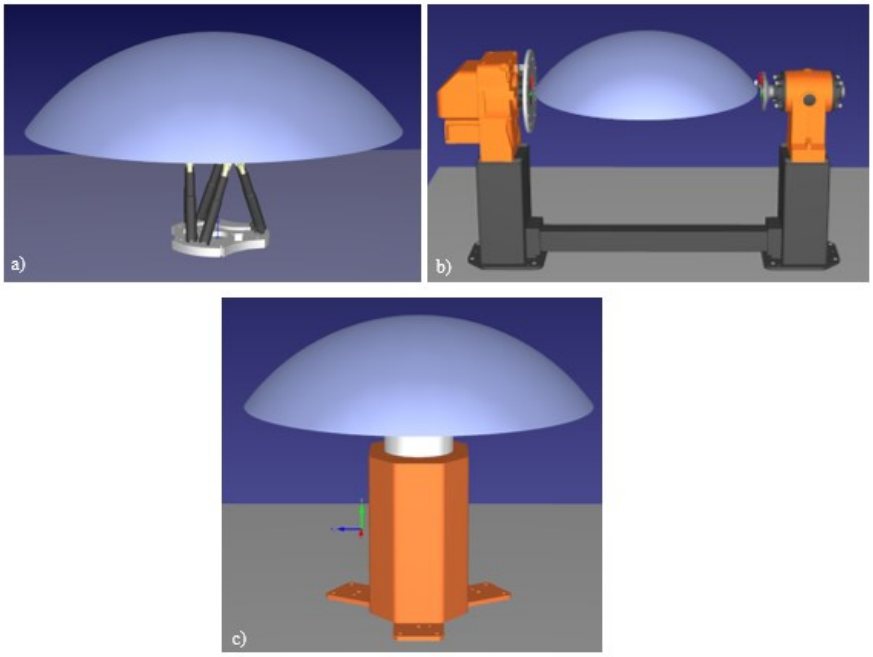

Figure 3.8 The bulkhead mounted on the a) KP1-V 500, b) ABB IRBP L600 L1250, c) PI H-840-D2A robot in simulation environment

3.6 The Pressure Bulkhead

In the realm of industrial manufacturing and engineering, a bulkhead refers to a structural partition within a larger vessel or framework that serves to provide strength, rigidity, and compartmentalization. Bulkheads are integral components in various industries, including maritime, aerospace, automotive, and construction, where they perform critical functions essential to the safety and integrity of structures. A pressure bulkhead is a critical structural component found in modern aircraft, particularly at the rear of the fuselage. It plays a pivotal role in maintaining cabin pressurization and ensuring the overall structural integrity of the aircraft during

flight. The design and construction of pressure bulkheads must accommodate the significant stress and pressure differentials that occur at high altitudes, ensuring the safety and comfort of passengers[66].

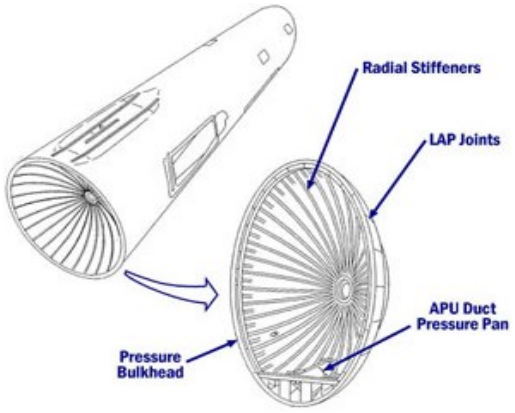

Figure 3.9 Pressure bulkhead[67]

This bulkhead is designed to maintain cabin pressure and ensure the structural integrity of the fuselage [3].

Radial stiffeners are essential structural elements that reinforce the bulkhead by distributing stresses evenly across its surface. Given the pressure differentials between the inside of the cabin and the exterior of the aircraft at altitude, these stiffeners ensure that the bulkhead maintains its shape and integrity. Radial stiffeners also prevent localized stresses from becoming concentrated, which could otherwise lead to material fatigue or failure.

The aim of the project is to find the best combination of dual robots to produce big and complex components such as a bulkhead with higher fiber placement accuracy. In Figure 3.10 the dimensions of the designed bulkhead to be placed on the robots for the fiber placement has been shown. It is also ensured that the part has the proper design to be placed on the KP1-V 500, ABB IRBP L600 L1250 and the PI H-840-D2A robots. To have a good comparison of the dual robots to be used, the dimensions of the structure will be the same in all the simulations.

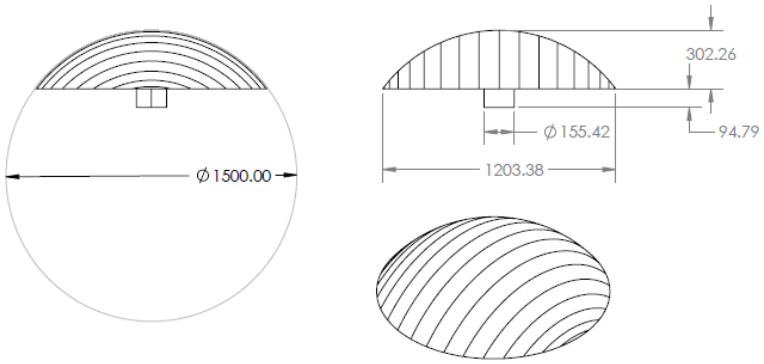

Figure 3.10 Dimensions of the Dome-Type structure (mm)

In the fiber-placement process, it's crucial to define and follow specific paths. Figure 3.11 illustrates the defined paths for fiber placement. For a fiber placement on the selected Dome-Type structure with fibers having a 1/4" width, a total of 188 paths were required based on the structure's dimensions. However, to optimize the simulation time, only 16 paths were selected and shown in Figure 3.11. Additionally, zones with a 3-inch width were defined, and one path was generated for each zone. It's ensured that the shortest and longest paths are included in the simulation, as they are the most critical paths for collision avoidance during fiber placement.

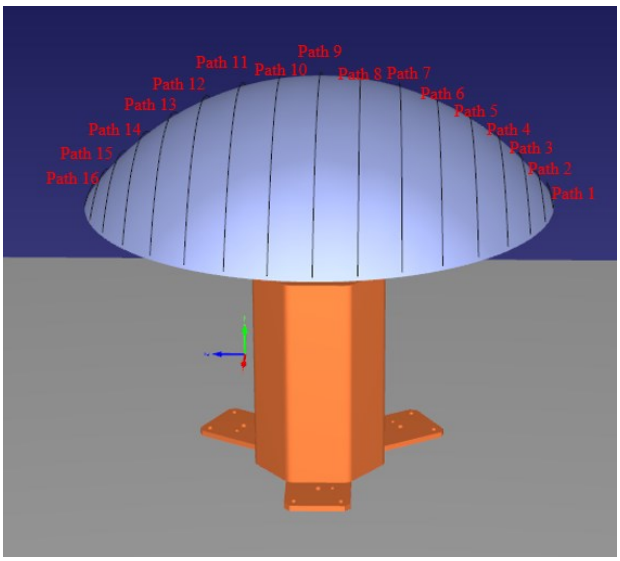

Figure 3.11 Paths on the bulkhead

The paths remain the same in all simulations and if the manipulability can be increased for these paths, the paths that are not included can achieve higher manipulability too. It is also important to mention that these paths are the middle line for a fiber with 1/4" width. The AFP

head that has been used in the simulation has the proper roller for this fiber and the middle of the roller moves on the paths.

Since the bulkhead's paths have been predefined to ensure the capability of the robots to do the fiber placement, they should be added to the simulation environment and then be moved to their correct position on top of the bulkhead on the second robot because when the paths are added manually, they will be shown as if their reference point is [0,0,0] in the simulation environment. Therefore, there is a need for a rotation to be done in the simulation. Based on the positioning of the robots, the transformation of the paths to their position for fiber-placement will be different. This happens since the bulkhead should be mounted on the second robot and based on the robot's specifications, the position of the bulkhead's center will be different in the workspace, and the paths are defined in relation to the bulkhead's center.

3.6.1 Transformation matrix

As mentioned before, there is a need to do a transformation on the paths to place them in their correct position on the bulkhead which is mounted on the second robot. A transformation matrix in the context of 6D (three translations and three rotations) is a mathematical tool used to move or transform a point or a rigid body in 3D space from one position and orientation to another. It combines both translational and rotational transformations into a single matrix, often referred to as a homogeneous transformation matrix. In 6D space, the transformation includes three translations (along the x, y, and z axes) and three rotations (about the x, y, and z axes). A 6D transformation matrix is a 4x4 matrix that represents both rotation and translation in a single operation. The general form of this matrix is:

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{3.9}$$

where T is a 4x4 transformation matrix and R is a 3x3 rotation matrix which handles the rotation of the point or object, t is a 3x1 **translation vector**, which represents the translation along

the x, y, and z axes, 0 is a row vector [0,0,0] and 1 is a scaler. Translation: The translation vector t consists of the displacements along the x, y, and z axes. In 3D space, this can be represented as:

$$t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{3.10}$$

where the components are the translations along the x, y, and z axes, respectively.

Rotation: The rotation matrix R is a 3x3 matrix that describes the orientation of an object. The full rotation in 3D can be broken down into three successive rotations about the coordinate axes, Rotation about the x-axis (roll) as θ_x, Rotation about the y-axis (pitch) as θ_y, and Rotation about the z-axis (yaw) as θ_z. The rotation matrix is the product of three individual rotation matrices, which correspond to the rotations around each axis:

$$R = R_z(\theta_z)R_y(\theta_y)\,R_x(\theta_x) \tag{3.11}$$

where the rotation about the x-axis (roll) is as below.

$$R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ 0 & \sin(\theta_x) & \cos(\theta_x) \end{bmatrix} \tag{3.12}$$

The rotation about the y-axis (pitch) is:

$$R_y(\theta_y) = \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) \\ 0 & 1 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) \end{bmatrix} \tag{3.13}$$

The rotation about the z-axis (yaw) is:

$$R_z(\theta_z) = \begin{bmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

To move or transform a point P from one location to another, the transformation matrix is applied. A point in 3D space is represented as a 4x1 column vector in homogeneous coordinates:

$$P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.15}$$

The purpose of the "1" in the vector is used in homogeneous coordination to allow the translation operations when applying the T matrx. To apply the transformation to this point, one may multiply the point by the transformation matrix T:

$$P' = T.P = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.16}$$

The resulting point P'will be the new transformed position after applying both the rotation and translation [37].

3.6.2 Fiber Path Transformation to the Correct Positions on the Bulkhead

In this section, we detail the mathematical framework and computational process applied to transform the path points into the correct position in the workspace of the robotic arm. When the paths are imported to the simulation environment, their points will be placed accordingly on point [0,0,0] and there is a need to place them on the bulkhead by applying a transformation on the points. The transformation involves both translation and rotation matrices to accurately position the fiber paths in the robot's workspace and rotate the robot's end-effector for optimal manipulability during the fiber placement process too. The core of this process lies in the transformation of path points stored in CSV files, which represent the fiber placement paths. The transformation matrix ensures that the points on the paths are moved and rotated into the correct

position and orientation in 3D space on the bulkhead that is mounted on the second robot. In the case that the bulkhead is mounted on the Kuka, KP1-V 500 robot, the translation and rotation parameters are:

$$t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} 1050 \\ 0 \\ -50 \end{bmatrix} \tag{3.17}$$

And $\theta_x = 90°, \theta_y = 0°, \theta_z = 90°$, the transformation moves each point to a new location and orientation within the robot's workspace. This ensures the fiber placement paths are correctly aligned relative to the robot's tool and workpiece. The complete transformation matrix T, which combines both rotation and translation, is defined as:

$$T = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

This matrix is applied to each point P in the path to yield the new transformed point P'.

$$P' = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11}x + r_{12}y + r_{13}z + t_x \\ r_{21}x + r_{22}y + r_{23}z + t_y \\ r_{31}x + r_{32}y + r_{33}z + t_z \\ 1 \end{bmatrix} \tag{3.19}$$

3.6.3 Paths Rotation Around the Z-Axis or Y-Axis

Based on the robot used for holding the bulkhead, there is a need to rotate the paths with the robot's rotation. Depending on the axis of rotation, R_z , R_x or R will be used. When applied to a position vector P, the formula becomes:

$$P' = R_z \times (P - C) + C \tag{3.20}$$

where C is the center of rotation and P' is the new position of the point after transformation.

3.7 Summary

In this chapter, the kinematic analysis and modeling of dual robot system including a 6-DOF Fanuc M20-iA holding the fiber placement head, and additional second robot holding the pressure bulkhead is presented. Three kinds of second robot have been explored to increase the manipulability including KUKA KP1-V 500, ABB IRBP L600 L1250 and one parallel robots. The CAD model of the bulkhead to be manufactured is given. The paths of the fiber placement for dual robot system are shown and the transformation of the path to the position on top of the bulkhead on the second robot is given.

Next chapter focuses on simulations and the algorithms for fiber placement that have been done along with the results.

Chapter 4

4 Setup, Simulation and Results of the Dual Robot Collaboration:

4.1 Introduction

In this chapter the focus is to find the best dual robot collaboration to do a high-quality fiber placement by comparing manipulability factor in each scenario which has been used to avoid singularities. To this end, simulation environments and different approaches have been used in order to validate the possibility of fiber-placement and find the best combination for the robots to do fiber placement on a complex shape. After finding the positioning values and the CAD file for each scenario, the simulation has been implemented. Figure 4.1 shows the AFP head design in the SolidWorks software package and represents the current AFP head design with the exact dimensions for all the components, and it has been used in the simulation to keep the AFP head perpendicular to the part and avoid collisions for the best results.

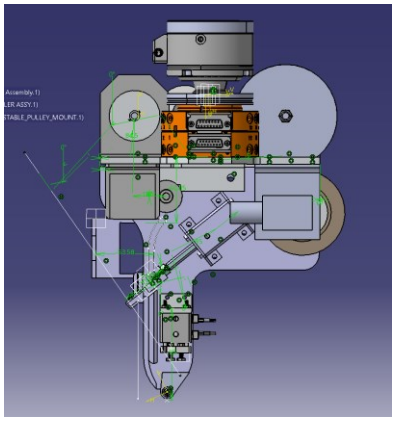

Figure 4.1 AFP head in SolidWorks environment[13]

There are some limitations of the robot which have to be addressed and calculated. The first step is to simulate the process in a simulation environment in order to check the procedure of the path planning.

4.2 Path Planning Algorithm for Fanuc M20-iA and the KUKA KP1-V 500

In this simulation setup, the bulkhead is placed on the KUKA KP1-V 500 robot and the fiber placement will be done by the Fanuc M20-ia robot. The produced paths for the bulkhead in Chapter 3 which were shown in Figure 3.11, will be added to the RoboDK workspace based on the position of the bulkhead being mounted on the rotary robot. In this scenario, it is only possible to rotate the part and the paths only around Z axis shown in Figure 4.2. Now that the positioning has been determined, the Python code generated for these robots combination for the fiber placement will be used to show the simulation.

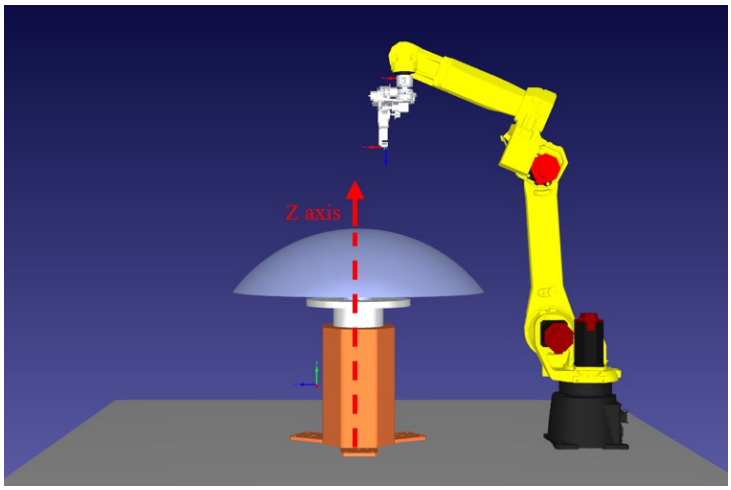

Figure 4.2 Fanuc M20-ia and the KUKA KP1-V 500 in RoboDK simulation

In this section, the simulation and the approach to do fiber-placement on the bulkhead for the combination of Fanuc M20-iA and the KUKA KP1-V 500 will be explained.

4.2.1 RoboDK Connection and Robot Validation

The script in Appendix A, begins by connecting to the **RoboDK** API, where the robots are selected from the RoboDK station. Each robot's validity is checked to ensure proper initialization. Any previously generated paths are deleted to clean up the environment for a new simulation.

4.2.2 Reading and Transforming and Visualizing Path Points

The fiber placement paths that were defined in chapter 3 are stored in CSV files which contain point coordinates and orientations in space. These paths represent paths that the robot will follow. The script reads these files and applies a series of transformations (translation and rotation) to each point. The transformation matrix is defined in Chapter 3. For each path the transformation matrix is applied to each point on the path to adjust the coordinates based on the rotary robot's base position and orientation and then saved as transformed points. This transformation is necessary to ensure that the paths are correctly placed on the bulkhead which is mounted on the Kuka robot. After transformation, 16 paths are visualized in RoboDK. A Z-offset correction (+530 units) is applied to ensure that the paths are to be reached with the end effector in the workspace. This is the length of the AFP head that is mounted on the Fanuc M20-iA. The paths are added to the RoboDK scene for the user to view the paths before the robot starts moving along them which is shown in Figure 4.3. It is important to note that the rotations about the z-axis mentioned are not related to the fiber orientations themselves. For instance, if the goal is to manufacture a composite layer with the first layer at 0°, the fibers should be placed on the bulkhead exactly as shown in Figure 4.3.

However, due to issues with the reachability of the robot, it is impossible to reach path 16. To resolve this, the bulkhead needs to be rotated about the z-axis and do the fiber placement. If the laminate includes other fiber orientations (e.g., a symmetric quasi-isotropic laminate with the [0/+45/90/-45]s layup sequence), initially the bulkhead needs to be rotated about z-axis with the corresponding angle for each orientation and then the algorithm can proceed as if no change has been made.

In all simulations, fiber placement is performed for the 0° layer, with rotations about the z-axis applied to achieve this orientation. By the end of the simulation, and after all rotations, the fibers placed on the bulkhead will match exactly as shown in Figure 4.3.

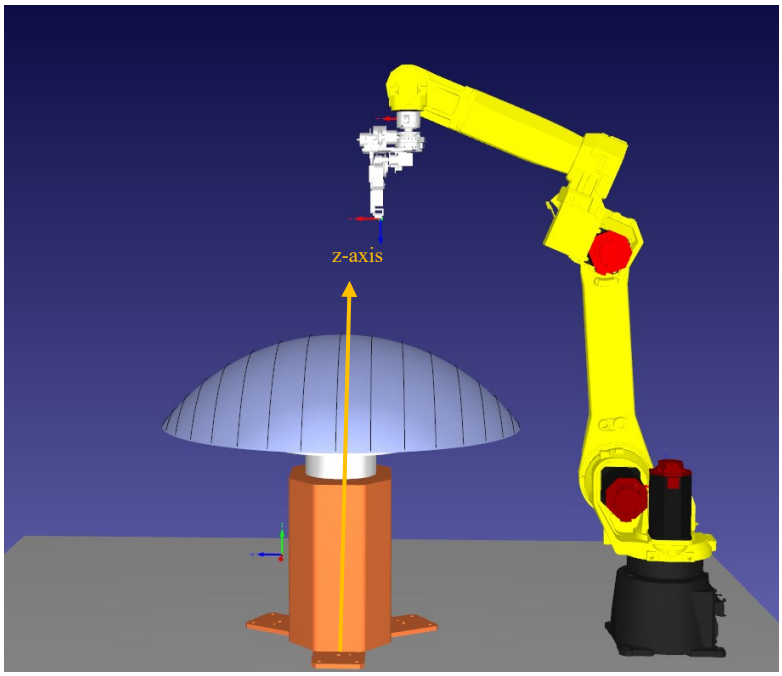

Figure 4.3 Initial position of the paths in the simulation

4.2.3 Calculating Sphere Center and Angles

To ensure that the robot is perpendicular to the paths, a sphere center is calculated using three randomly selected points from different paths. Figure 4.4 shows a point on a sphere and the center of it.

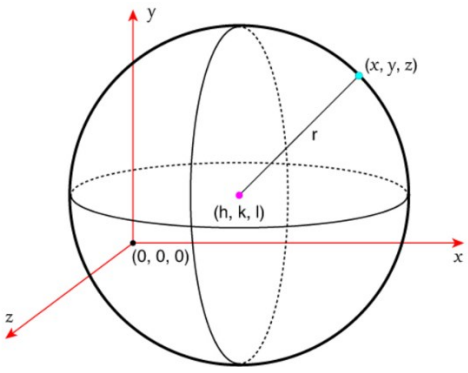

Figure 4.4 A pointon a sphere and the center of the sphere

Since all the points on the sphere should have the same distance from the sphere based on the following equation the center can be calculated.

$$(x-h)^2+(y-k)^2+(z-I)^2=r^2 \tag{4.1}$$

Where (x, y, z) are the coordinators of each point and r is the radius of the sphere that is available from the bulkhead's design. This center is used to compute the angles between the vectors formed by the points on the path and a reference vector in order to find the perpendicular vector on the surface. These vectors will guide the Fanuc M20-iA's movement by controlling the orientation of the end-effector along the fiber path. Figure 4.4 shows the normal vectors on the points on a path, which is the orientation of the end effector of the AFP head.

4.2.4 Robot Movement Along Paths

The robot moves along the path points using the calculated vectors. The motion is executed in joint space using the proper commands, where the position and orientation of the robot's end-effector are set based on the normal vectors of the points on the paths. For paths, specific collision avoidance measures are taken by adjusting the orientation of the robot to prevent contact with obstacles while maintaining a perpendicular position to the surface. The script checks if the robot successfully reaches each point on the path. If any point is unreachable, the process for that path is aborted, and an alert will come up in the code. In this step, the robot will move on the paths while being perpendicular to the surface and the manipulability of the robot will be calculated at each point, then each path will be rotated around the Z axis and this process will continue, at the end for each path the best position based on the highest manipulability will be found and then each path will come with a Z angle which shows the amount of rotation needed around the Z axis for the best fiber placement and the data will be stored to be used later. In the final code this data will be just imported and shows the final fiber placement version. Figure 4.5 illustrates the paths after the rotation.

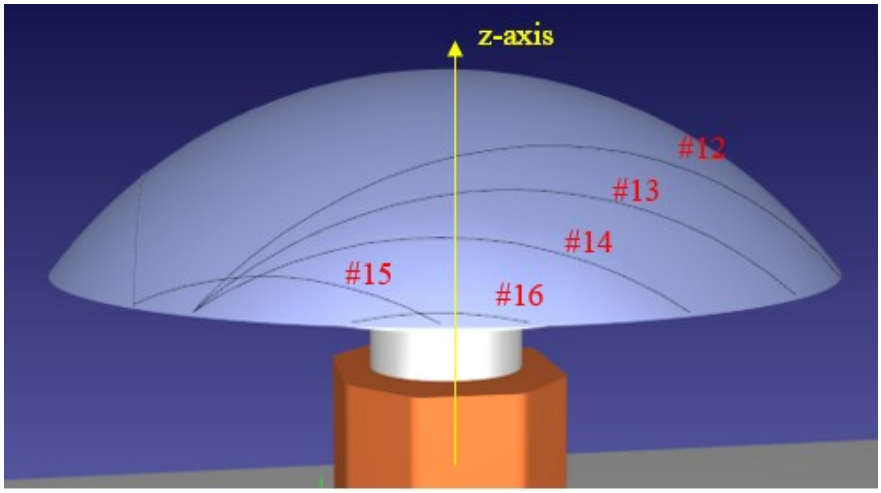

Figure 4.5 The rotated paths with highest manipulability

As it was mentioned before, there are some paths that are not reachable and there is a need to rotate the bulkhead to bring them into a position in the Fanuc M20-iA's workspace in order to do the fiber placement. The paths shown in Figure 4.5, are the ones that had singularity, reachability or low manipulability factors and they needed a rotation for a better positioning so that the Fanuc M20-iA could have moved on them for the fiber placement. Each time the bulkhead will be rotated with the necessary amount for each path and then the Fanuc M20-iA will do the fiber placement. To clarify the results of the fiber placement on rotated paths shown in Figure 4.5 will be the paths that were shown in Figure 4.3 when the fibers oriented at 0°. Paths that are not shown in Figure 4.5 are on the other side of the the structure and are not visible from this point of view.

Table 4.1 shows the results for the rotation needed for each path to obtain a high manipulability.

Table 4.1 The rotation for each path to achieve high manipulability

Path number	1	2	3	12	13	14	15	16
Z angle rotation	90°	75°	60°	100°	90°	80°	60°	80°

4.2.5 Final Visualization

The rotated paths are displayed in RoboDK, showing the final fiber placement paths after all transformations. This allows users to visualize the robot's motion and the effect of the applied rotations. Figure 4.6 shows the final paths, some of which should stay the same because of reachability problems that could not have been solved by a simple rotation.

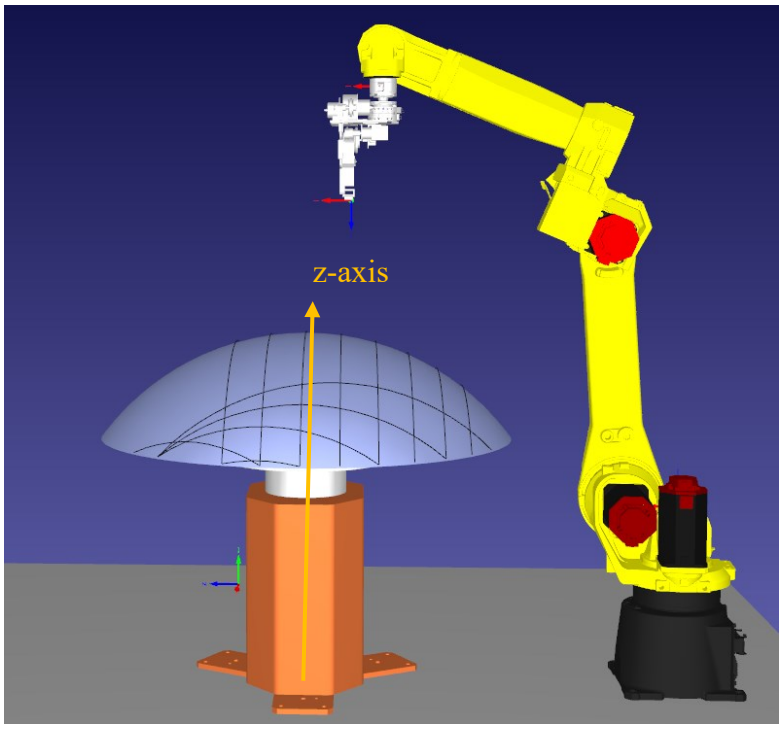

Figure 4.6 Final paths positions for the combination of Fanuc M20-ia and the KUKA KP1-V 500

4.3 Fanuc M20-ia and ABB IRBP L600 L1250

In this simulation, the bulkhead is placed on the ABB IRBP L600 L1250 robot and the fiber placement will be done by the Fanuc M20-iA robot. The produced paths for the bulkhead in Chapter 3 will be added to the simulation area based on the position of the bulkhead being on the rotary robot. Now that the positioning has been determined, the python code generated for this

particular robot combination for the fiber placement will be used to show the simulation. Figure 4.7 shows the setup for this combination.

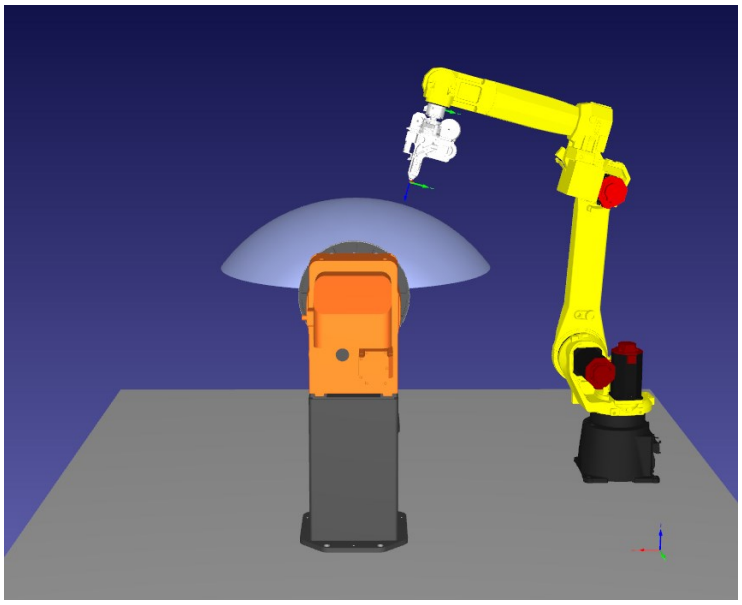

Figure 4.7 Shows the initial setup for Fanuc M20-ia and ABB IRBP L600 L1250

4.3.1 Algorithm and code explanation

With the connection established, the next critical step is defining robotic models to be used in the simulation. The Fanuc M-20iA and ABB IRBP L600 L1250 robots were retrieved from the RoboDK library, and their validity within the simulation environment was confirmed to ensure accurate modeling.

This scenario has two codes that are in Appendix B and Appendix C. In this scenario the first code finds the best rotation and position for each path and the second code implements the best positions of the paths and will be the actual fiber placement on the bulkhead in this scenario.

To maintain a clean simulation environment, any previously generated paths were systematically deleted. This cleanup process prevented residual data from previous simulation runs to avoid any problems in the current analysis and this ensures that each simulation begins with a fresh workspace.

4.3.2 Configuring the Rotary Mechanism

Central to the simulation environment is the rotary mechanism which rotates along the Y-axis. Two points were defined in the center of the ABB IRBP L600 L1250's joints to establish the rotation axis. Later this rotation axis will be visualized in the simulation environment. Defining these points was crucial, as the rotation axis served as the pivot for all rotational transformations applied to the paths on the bulkhead and the bulkhead itself. Precise alignment of the rotation axis ensured that the robots' movements adhered to the intended fiber placement trajectories.

4.3.3 Developing the Rotation Matrix

To facilitate rotations around the defined axis, a rotation matrix was crafted using Rodrigues' rotation formula[37], [68]. This has enabled the simulation to apply precise angular transformations to the robots, ensuring that fiber placement followed the desired trajectories.

For this scenario there is a new transformation matrix for the paths because the bulkhead is placed on the ABB IRBP L600 L1250 robot and the dimensions are different from the first scenario. After defining the new transformation matrix, the paths are shown in Figure 4.8.

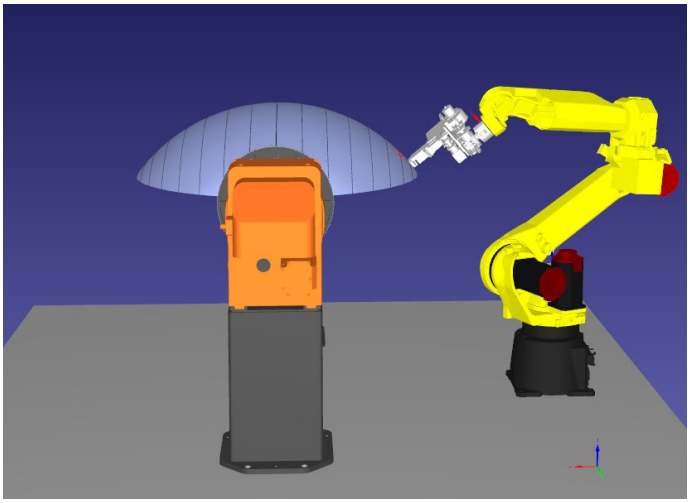

Figure 4.8 The initial position of the paths in this setup

4.3.4 Establishing the Initial Position of the Robots

It was essential to define the Fanuc M20-iA's initial poses within the simulation environment. This involved setting their positions and orientations far away from a singular pose. The actual AFP head is mounted on the Fanuc M20-iA to ensure collision avoidance.

4.3.5 Implementing Transformation and Rotation Functions

To start simulation process, transformation matrix was developed to the transformations on the paths to bring them to their actual position on the bulkhead. This step is entitled "Initial Transformation" in Appendix B. The new transformation functions were implemented to apply the rotation around the y-axis when the ABB IRBP L600 L1250 rotates.

Additionally, a visualization function was incorporated to render the rotation axis within RoboDK, providing a clear reference point for understanding the rotational dynamics of the robots during the simulation. In this scenario, it is only possible to rotate the part and the paths only around Y axis as shown in Figure 4.9. This step provided a clear reference point for understanding the rotational dynamics of the robots during the simulation.

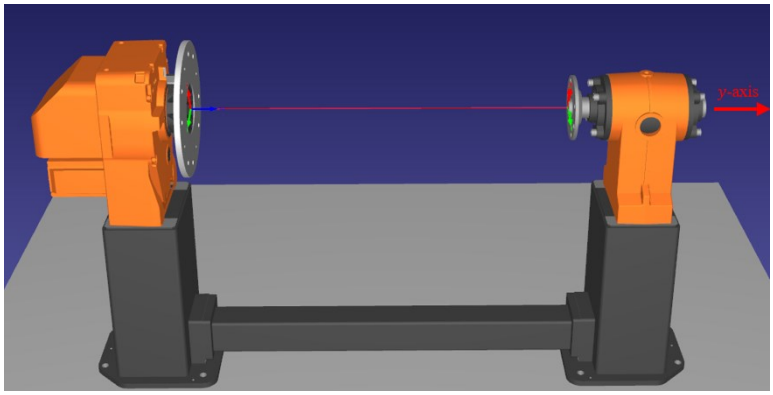

Figure 4.9 The rotation axis for the ABB IRBP L600 L1250 robot

4.3.6 Rotating and Storing Fiber Placement Paths

The paths shown in Figure 4.8 are in the initial position of the ABB IRBP L600 L1250 robot and this is when the ABB IRBP L600 L1250 robot has not rotated and its in 0°. Then the ABB IRBP L600 L1250 robot will rotate 5° until the robot rotates 90°. Each time a simulation of the fiber placement by Fanuc M20-iA will be done and the manipulability factor will be calculated and stored as a CSV file. The best amount of rotation about y-axis for each path based on manipulability factor was found by using the code in Appendix B. These results were stored to be used in Appendix C for the final simulation. Figure 4.10 shows the position of the bulkhead and the paths as if the ABB IRBP L600 L1250 robot was rotated for 10°.

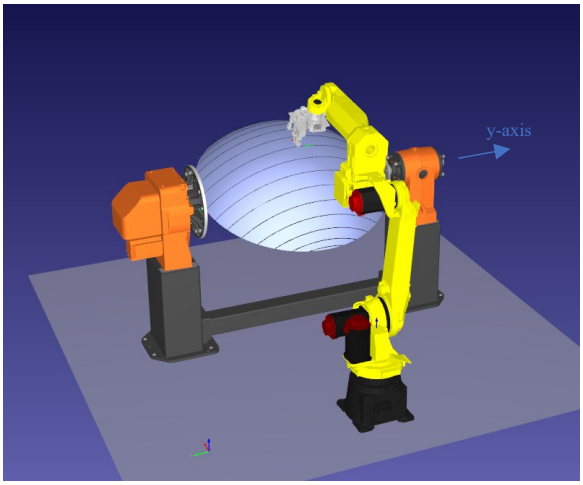

Figure 4.10 ABB IRBP L600 L1250 at 10° rotation for fiber placement

It is important to note that the best results for each path were determined based on high manipulability factor and with the least number of huge drops in the manipulability factor during the fiber placement on each path.

4.3.7 Assigning Rotation Angles to Paths

Each fiber placement path was assigned a specific rotation angle about y-axis based on the initial simulations.

By defining a mapping of the new paths respective to their optimal rotation angles, the simulation evaluated the robots' performances during the fiber placement. This has been implemented using the code in Appendix C.

4.3.8 Processing Each Fiber Placement Path

The simulation iterated through each defined path, applying the designated rotational transformations and guiding the Fanuc robot along the transformed path while coordinating with the ABB robot. Throughout this process, manipulability factor were calculated to verify the robots' performance.

For each path, the Fanuc robot's pose was reset to its original state, a rotation matrix was created and applied based on the optimal angle identified earlier, and the path points were transformed accordingly. The transformed path was then visualized within RoboDK to verify alignment and accuracy. The Fanuc robot was subsequently moved along the transformed path, with manipulability indices recorded at each path point which is shown in Figure 4.11. Simultaneously, the ABB robot adjusted the rotary mechanism to maintain optimal fiber placement angles, ensuring coordinated and efficient operations.

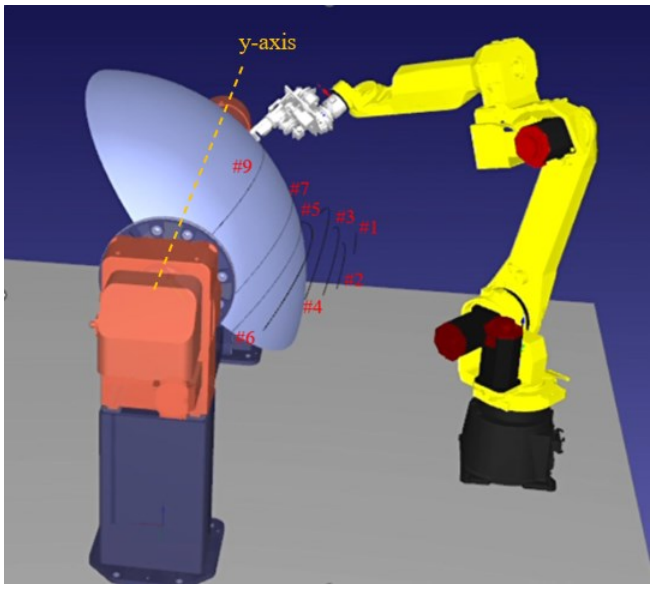

Figure 4.11 Fiber placement on the 9th path with Fanuc M20-iA

Figure 4.11 shows the simulation when the bulkhead has rotated, and the Fanuc is doing the fiber placement on the 9th path. After finishing the 9th path, the path will remain in the position where the fiber placement was done. For a better understanding figure 4.12 illustrates the simulation for the 13th path and the previous paths are in the positions where the fiber placement was done. Some paths are not visible in figure 4.12 because the bulkhead is covering them.

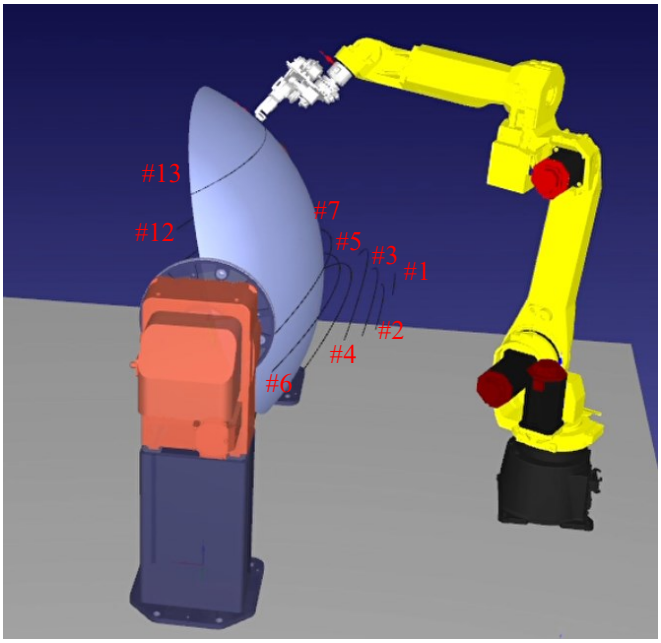

Figure 4.12 Fiber placement on the 14th path with Fanuc M20-iA

4.3.9 Visualizing Manipulability Indices

Utilizing Python's plotting capabilities, a graph was generated to illustrate how manipulability changes along each path by making rotations to the second robot. The visualization included distinct color-coding for each path based on their assigned rotation angles, to clarify the results.

Figure 4.13 to Figure 4.22 show the results of manipulability of Fanuc M20-iA by rotating the ABB IRBP L600 L1250 robot on the points of each path after possible rotations. Smaller rotations have also been done and the final results have been shown in Table 4.2.

The best rotation for each path has been chosen based on the form of the plot, the plots that had a huge drop or low manipulability factors have been eliminated.

Second Scenario Fanuc M20-iA and ABB IRBP L600 L1250 Robots

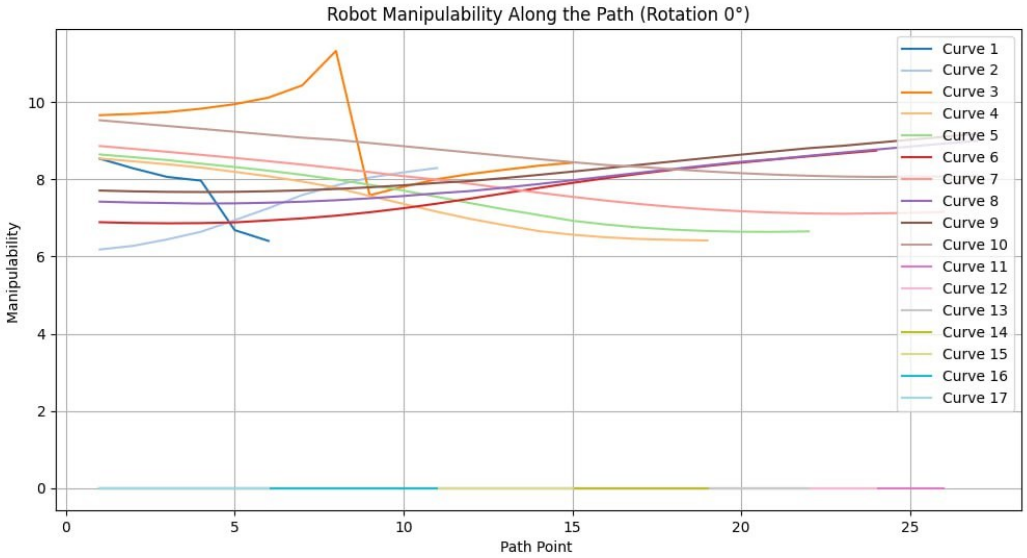

Figure 4.13 Manipulability of the Fanuc M20-iA on each path when the ABB IRBP L600 L1250 has rotated 0°

Second Scenario Fanuc M20-iA and the ABB IRBP L600 L1250 Robots

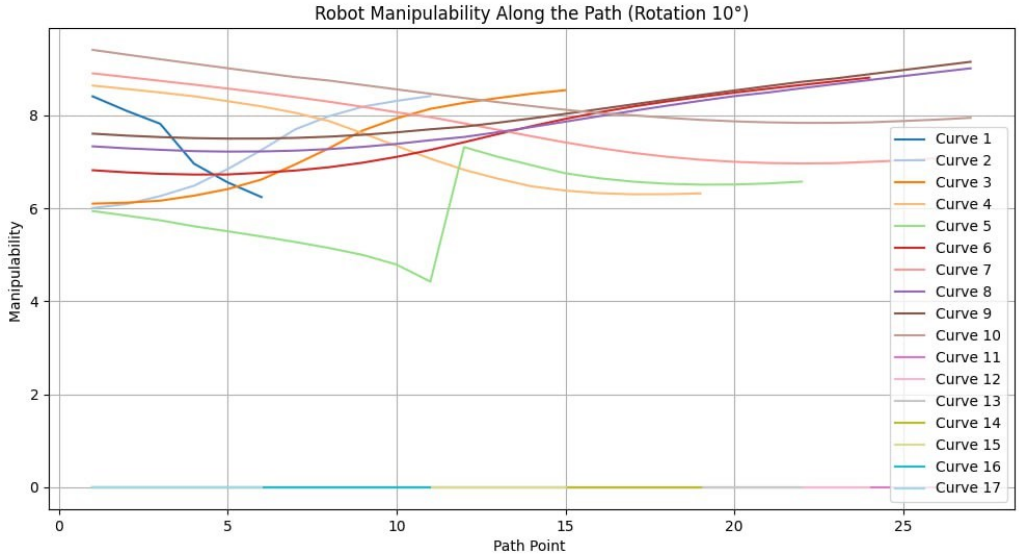

Figure 4.14 Manipulability of the Fanuc M20-iA on each path when the ABB IRBP L600 L1250 has rotated 10°

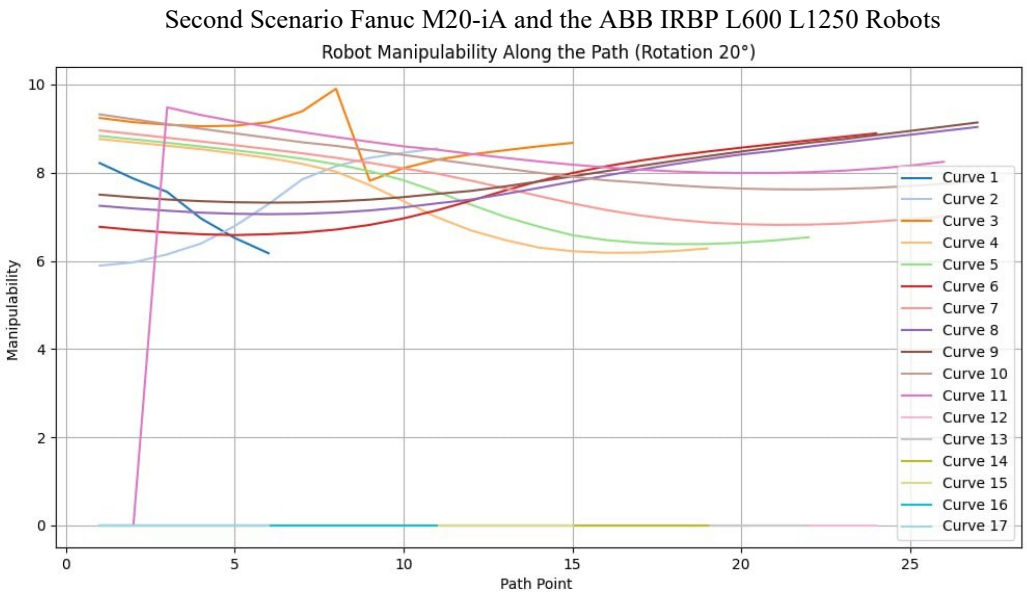

Figure 4.15 Manipulability of the Fanuc M20-iA on each path when the ABB IRBP L600 L1250 has rotated 20°

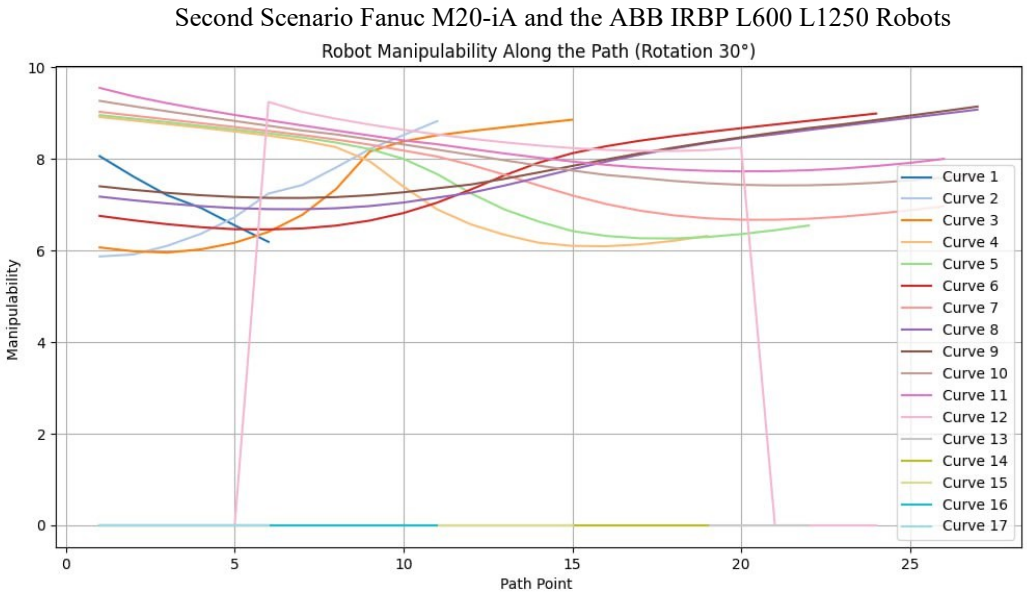

Figure 4.16 Manipulability of the Fanuc M20-iA on each path when the ABB IRBP L600 L1250 has rotated 30°

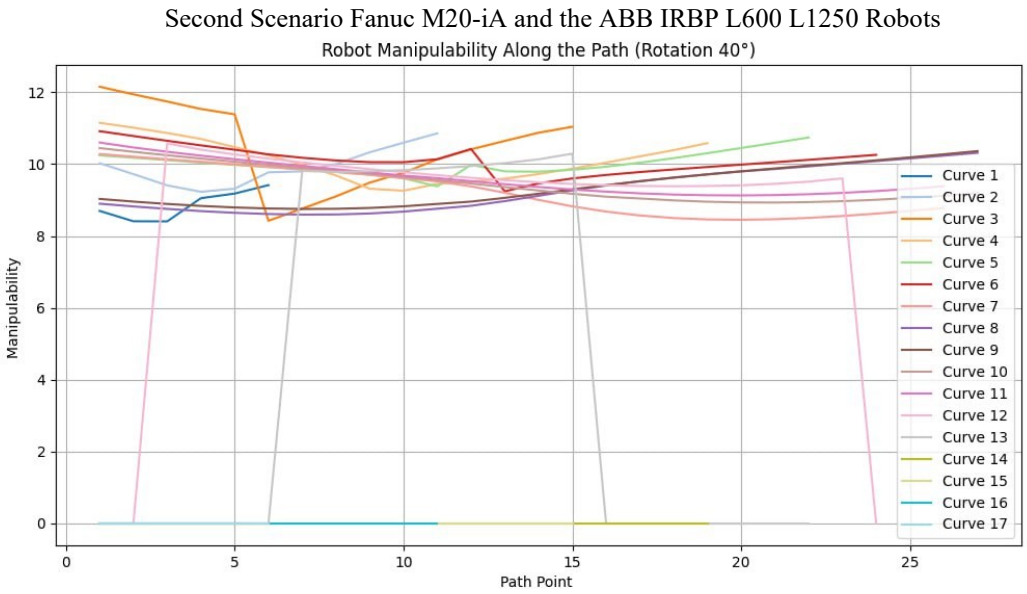

Figure 4.17 Manipulability of the Fanuc M20-iA on each path when the ABB IRBP L600 L1250 has rotated 40°

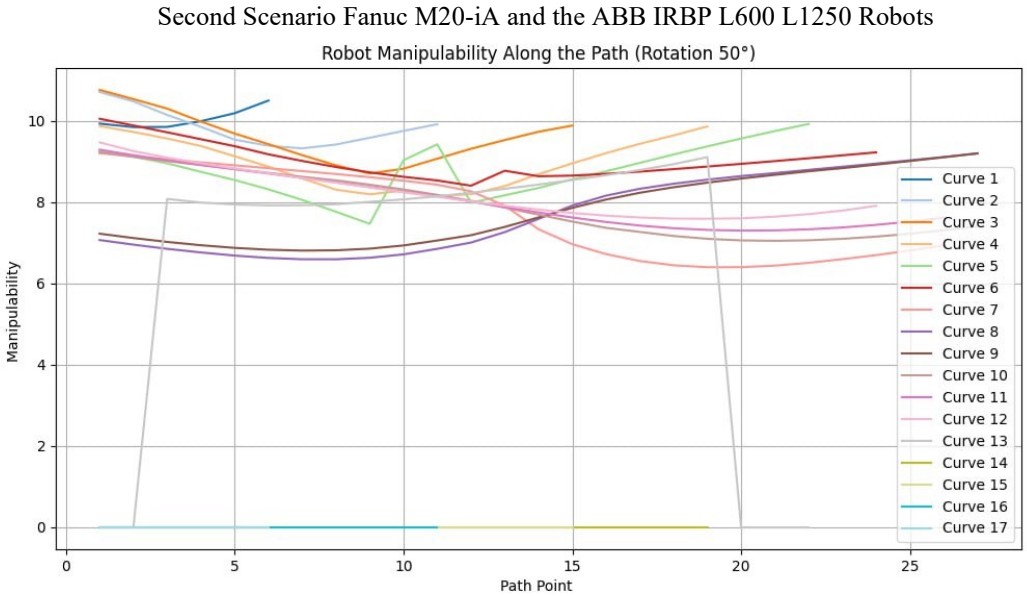

Figure 4.18 Manipulability of the Fanuc M20-iA on each path when the ABB IRBP L600 L1250 has rotated 50°

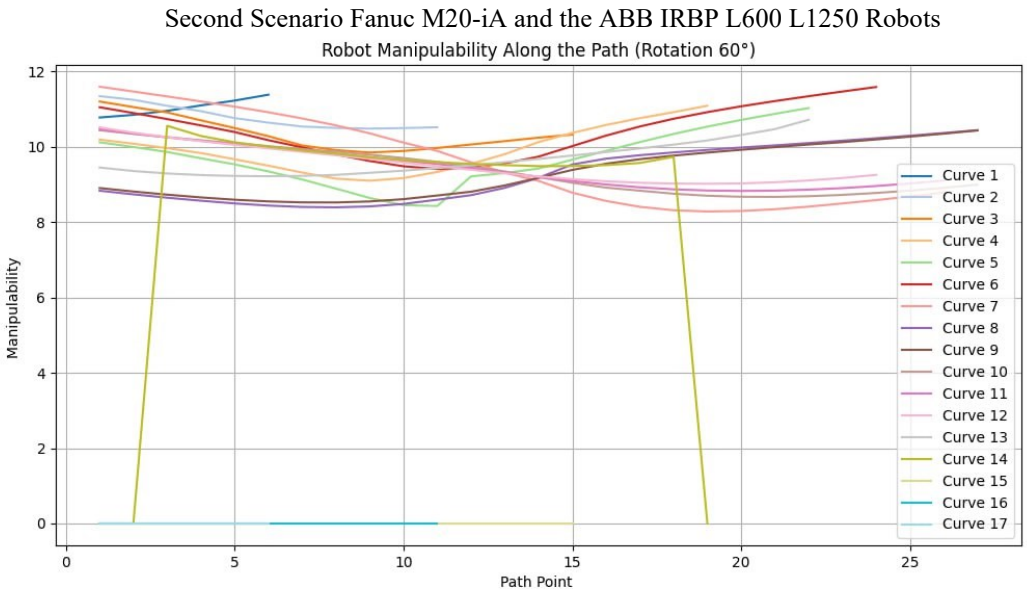

Figure 4.19 Manipulability of the Fanuc M20-iA on each path when the ABB IRBP L600 L1250 has rotated 60°

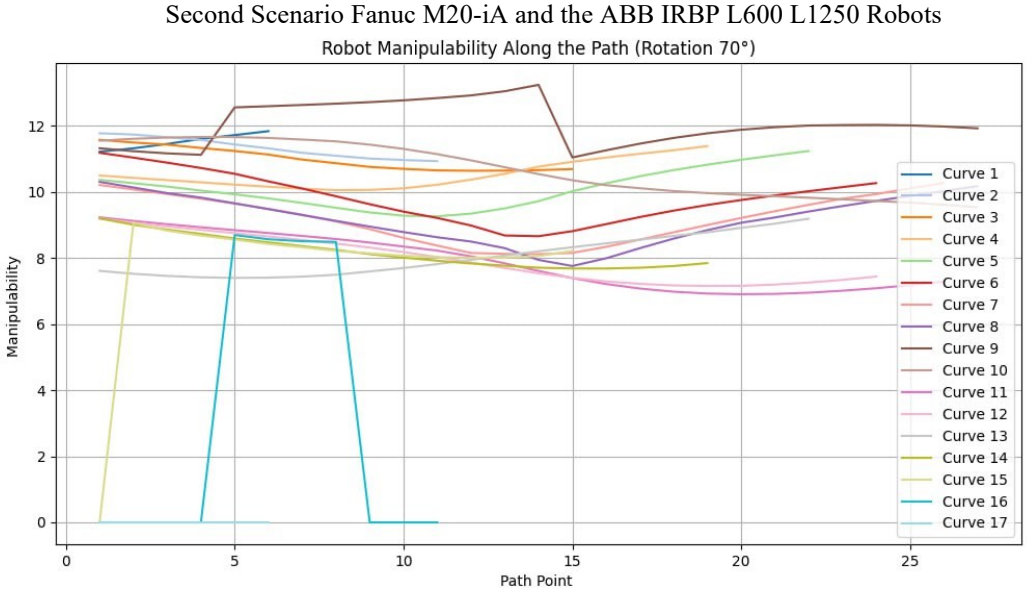

Figure 4.20 Manipulability of the Fanuc M20-iA on each path when the ABB IRBP L600 L1250 has rotated 70°

Second Scenario Fanuc M20-iA and the ABB IRBP L600 L1250 Robots

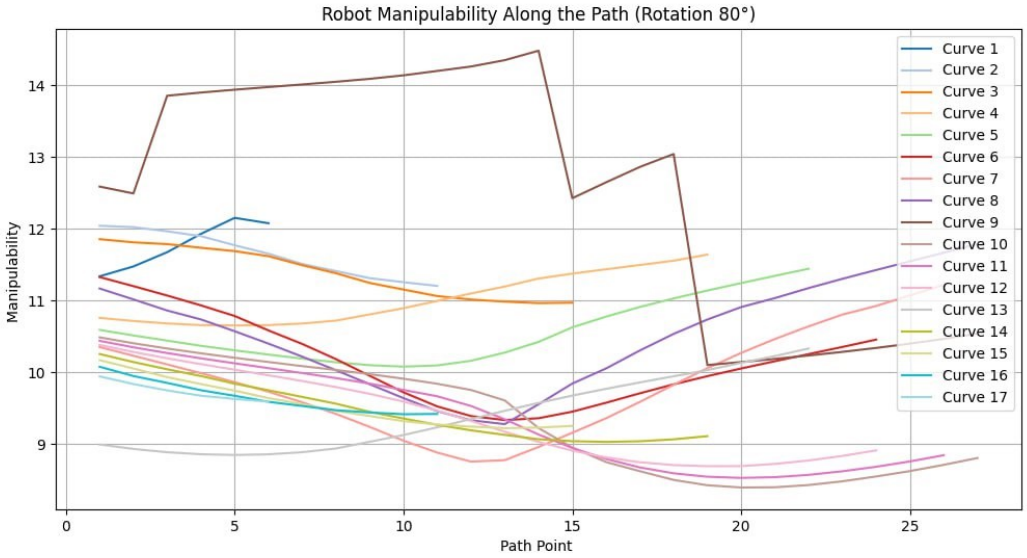

Figure 4.21 Manipulability of the Fanuc M20-iA on each path when the ABB IRBP L600 L1250 has rotated 80°

Second Scenario Fanuc M20-iA and the ABB IRBP L600 L1250 Robots

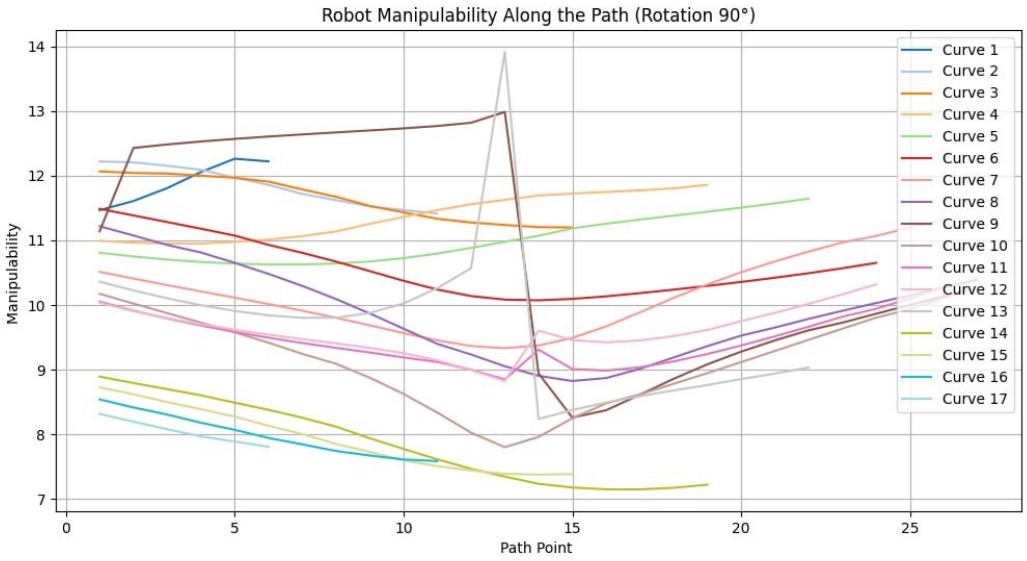

Figure 4.22 Manipulability of the Fanuc M20-iA on each path when the ABB IRBP L600 L1250 has rotated 90°

It is important to mention that the length of the paths shown in the Figures are different because on the bulkhead the lengths of the paths are different too and the points were selected evenly based on distance. For instance, the first path has 8 points distributed each 5cm and the 10th path has 28 points distributed each 5cm. The best rotation for each path has been selected from the

plots with summing the manipulability of the points on that path and checking for huge drops in the plot.

Table 4.2 shows the rotations that the ABB IRBP L600 L1250 needed to reach the highest manipulability possible of the Fanuc M20-iA for each path to obtain a high-quality fiber placement.

Table 4.2 The rotations for each path to achieve high manipulability

Path number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
y-axis Rotation (degree)	10	20	20	25	40	50	55	5	50	60	60	63	75	80	80	85

4.4 Fanuc M20-ia and the Parallel Robot with a Rotary Stage

4.4.1 Introduction

This is the third scenario in which the fiber placement is done by using the FANUC M-20iA robot combined with the parallel robot. FANUC M-20iA and PI H-840-D2A as the parallel robot has been used. In the Concordia university lab, there is a parallel robot, but since it was not available in RoboDK, the PI H-840-D2A has been used instead in the simulation environment. In this part there are two codes mentioned in Appendix D and Appendix E, the first code is to obtain the rotations for the paths and the second code is to implement the best results on the paths and illustrate the fiber placement.

4.4.2 Establishing the Simulation Environment

The simulation begins by setting up the necessary environment within RoboDK. Both robots are retrieved and validated to ensure they are ready for simulation. The robot's initial position, pose and joint configurations are defined.

4.4.3 Loading and Transforming Fiber Placement Paths

The fiber placement paths are defined by paths and each path represents a specific path along which fibers are to be placed on the component. For each point in the path, the combined rotation and translation has been applied to obtain the transformed points and be placed on the bulkhead. Translation Values (1045, 0, -45) were chosen to position the paths within the robot's reachable workspace, considering the physical layout of the robot and the component and the Rotation Angles (90°, 0°, 90°) align the paths with the robot's coordinate system, ensuring that the paths are correctly placed on the bulkhead. After transforming the paths, they are visualized within RoboDK as shown in Figure 4.23, to verify their positions and orientations.

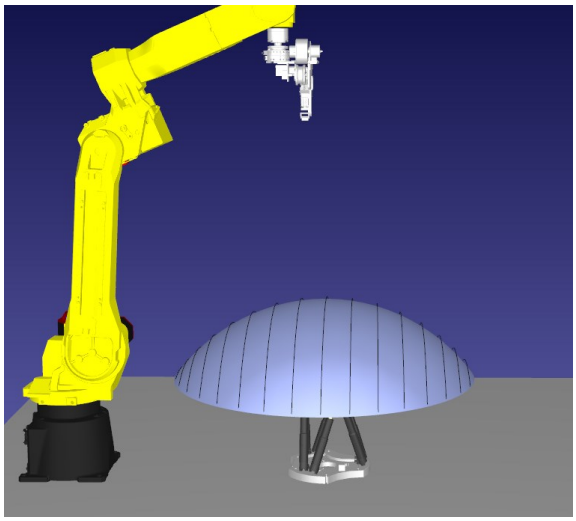

Figure 4.23 Initial setup and path positions

4.4.4 Calculating the Bulkhead Center

To optimize the robot's movements, it's essential to understand the central point around which rotations will be applied and to find the perpendicular vectors to find the Fanuc's orientation to reach the points on the paths. Computing the mean position of all transformed points to determine the central point of the bulkhead. The bulkhead center serves as the pivot point for rotations, ensuring the ability to calculate the normal vectors. Figure 4.24 illustrates the normal vectors on the path.

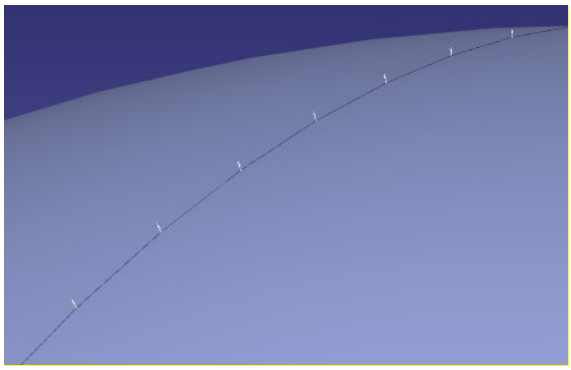

Figure 4.24 Normal Vectors on the Bulkhead in RoboDK

4.4.5 Evaluating Manipulability and Determining Optimal Rotations

To enhance the robot's performance, the simulation evaluates how different rotations of the paths affect the robot's manipulability. There are different rotation types. Z-Axis Rotation which rotates the paths around the Z-axis at various angles. Parallel Platform Rotation which simulates roll, pitch and yaw rotations. Z-Axis Rotation Angles have been tested from 0° to 360° in 10° increments to cover most of the possible orientations around the z-axis. Stewart Platform Angles have been also tested, roll, pitch and yaw angles in their limits each in 10° increments to simulate possible rotations by the parallel robot. Optimal Angle Selection is based on the highest manipulability factor for each path. These are the results of the code in Appendix D implementing and plotting different rotations to choose the best rotation for each path and they have been shown in Figure 4.25 to Figure 4.40.

Third Scenario Fanuc M20-iA and the PI H-840-D2A Robots

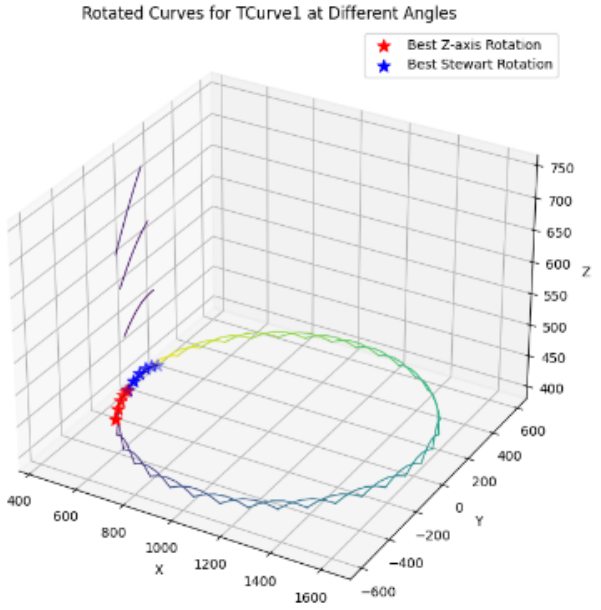

Figure 4.25 Different positions for Path 1 by movements of the PI H-840-D2A

Third Scenario Fanuc M20-iA and the PI H-840-D2A Robots

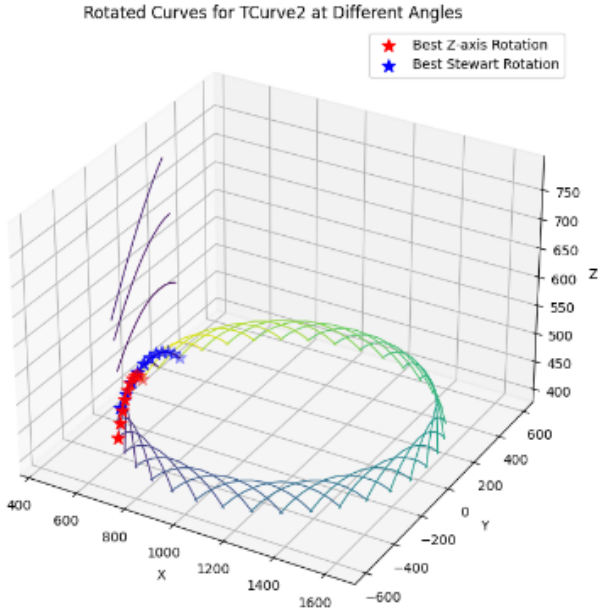

Figure 4.26 Different positions for Path 2 by movements of the PI H-840-D2A

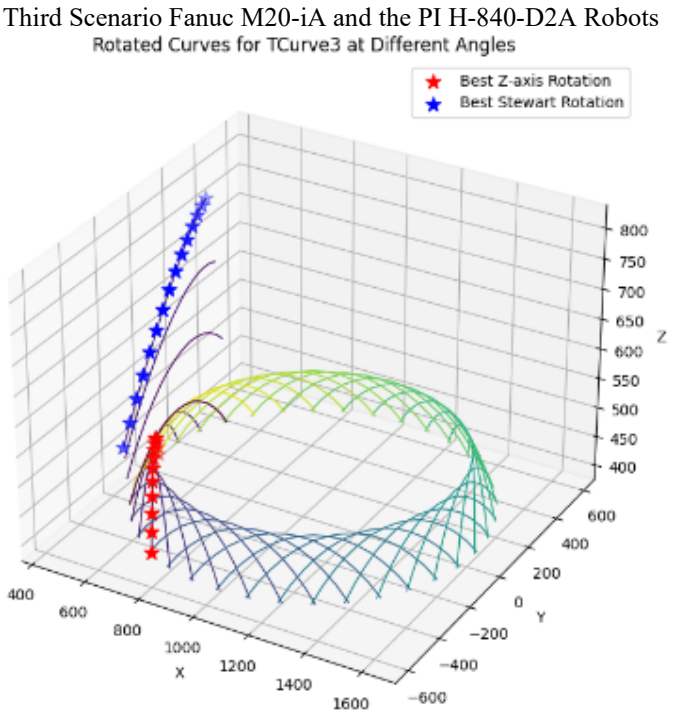

Figure 4.27 Different positions for Path 3 by movements of the PI H-840-D2A

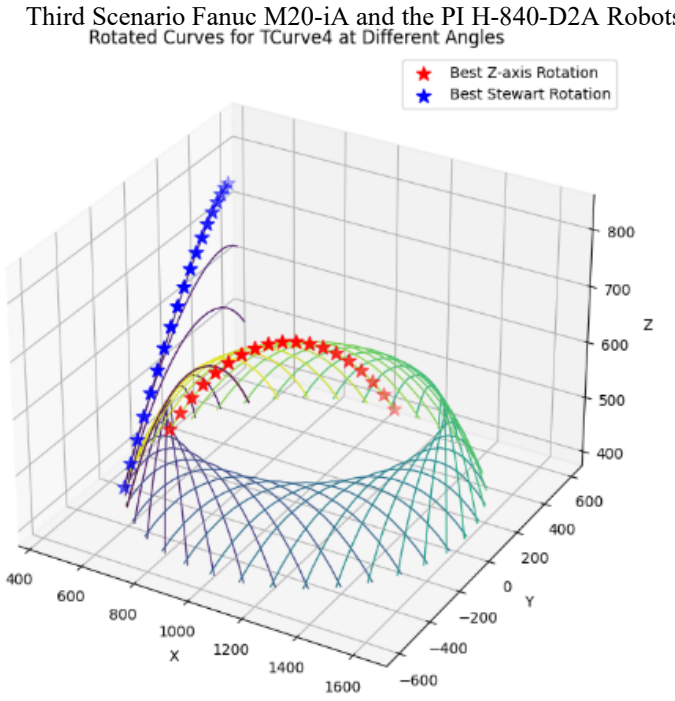

Figure 4.28 Different positions for Path 4 by movements of the PI H-840-D2A

Third Scenario Fanuc M20-iA and the PI H-840-D2A Robots

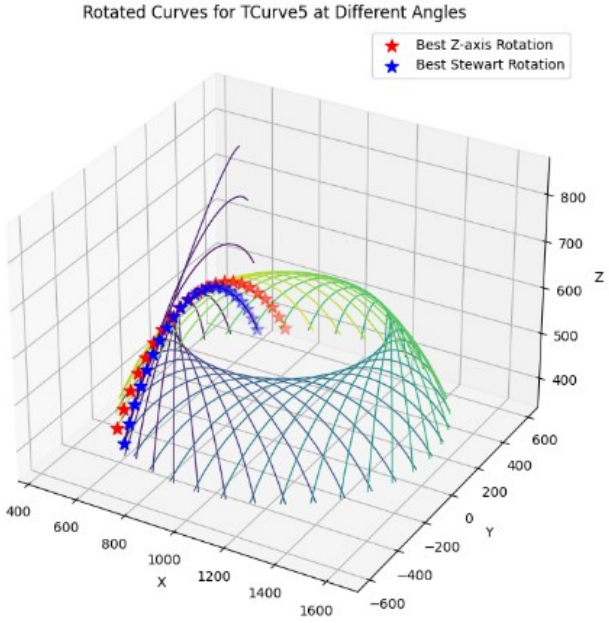

Figure 4.29 Different positions for Path 5 by movements of the PI H-840-D2A

Third Scenario Fanuc M20-iA and the PI H-840-D2A Robots

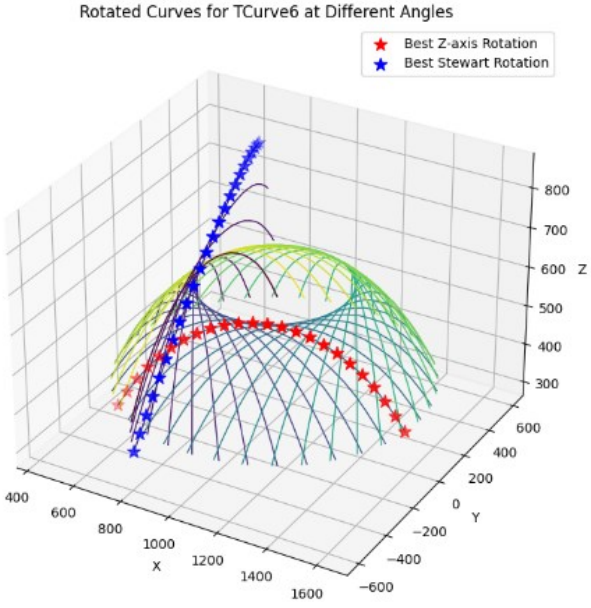

Figure 4.30 Different positions for Path 6 by movements of the PI H-840-D2A

Third Scenario Fanuc M20-iA and the PI H-840-D2A Robots

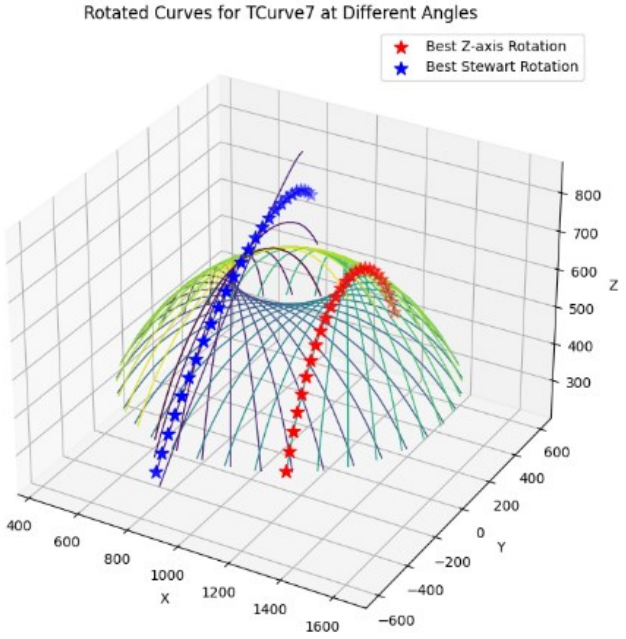

Figure 4.31 Different positions for Path 7 by movements of the PI H-840-D2A

Third Scenario Fanuc M20-iA and the PI H-840-D2A Robots

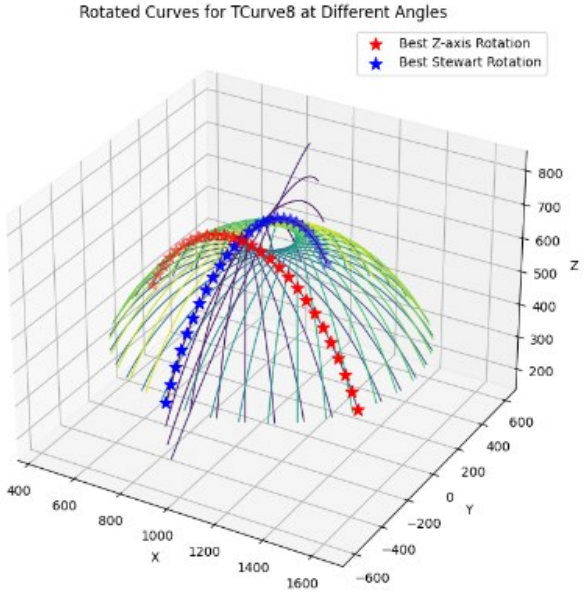

Figure 4.32 Different positions for Path 8 by movements of the PI H-840-D2A

Third Scenario Fanuc M20-iA and the PI H-840-D2A Robots

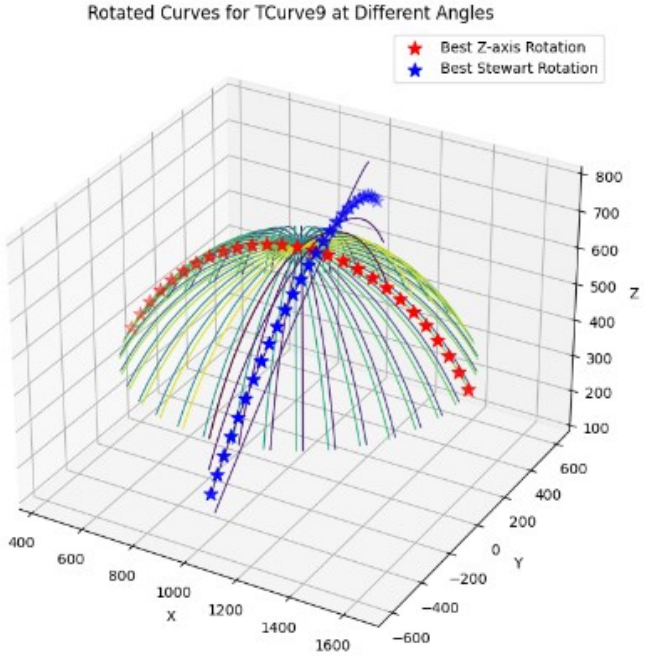

Figure 4.33 Different positions for Path 9 by movements of the PI H-840-D2A

Third Scenario Fanuc M20-iA and the PI H-840-D2A Robots

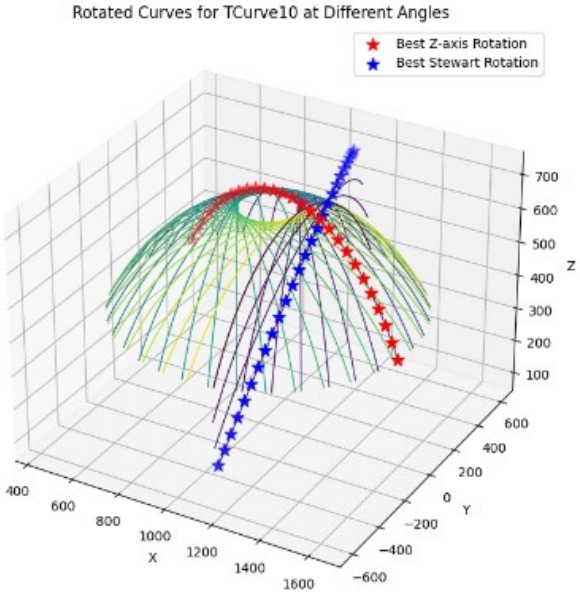

Figure 4.34 Different positions for Path 10 by movements of the PI H-840-D2A

Third Scenario Fanuc M20-iA and the PI H-840-D2A Robots

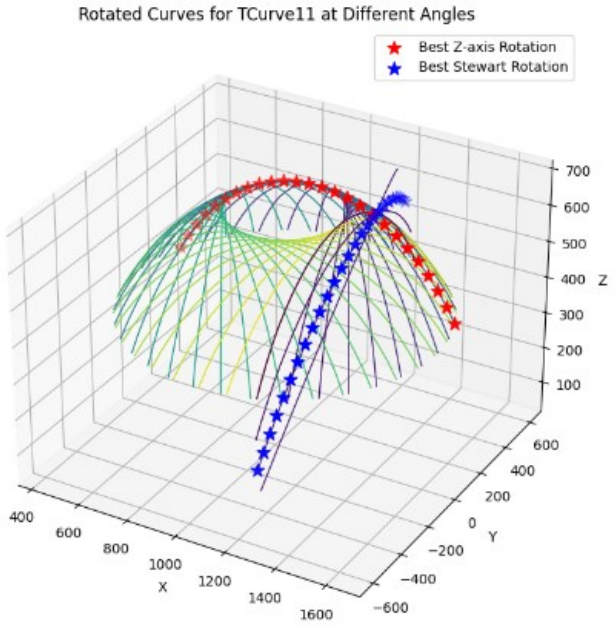

Figure 4.35 Different positions for Path 11 by movements of the PI H-840-D2A

Third Scenario Fanuc M20-iA and the PI H-840-D2A Robots

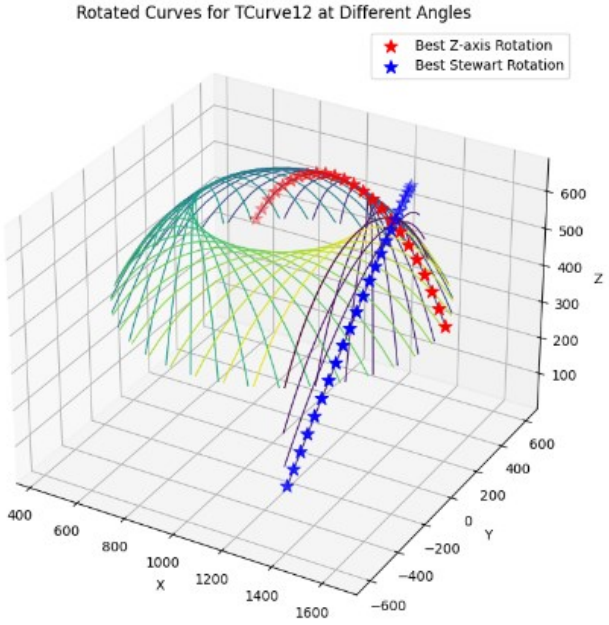

Figure 4.36 Different positions for Path 12 by movements of the PI H-840-D2A

Third Scenario Fanuc M20-iA and the PI H-840-D2A Robots

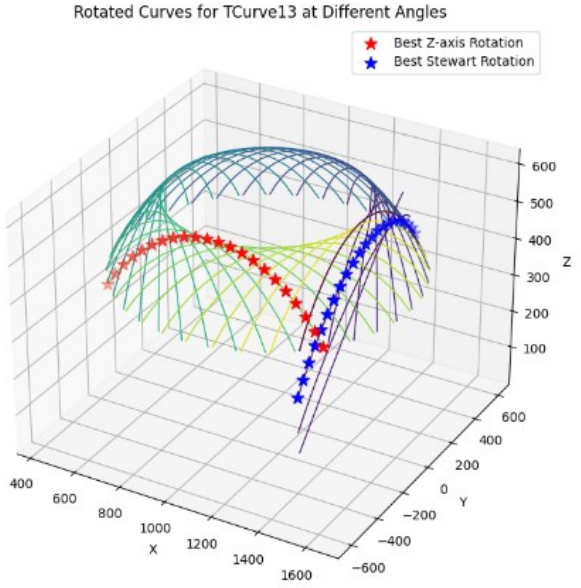

Figure 4.37 Different positions for Path 13 by movements of the PI H-840-D2A

Third Scenario Fanuc M20-iA and the PI H-840-D2A Robots

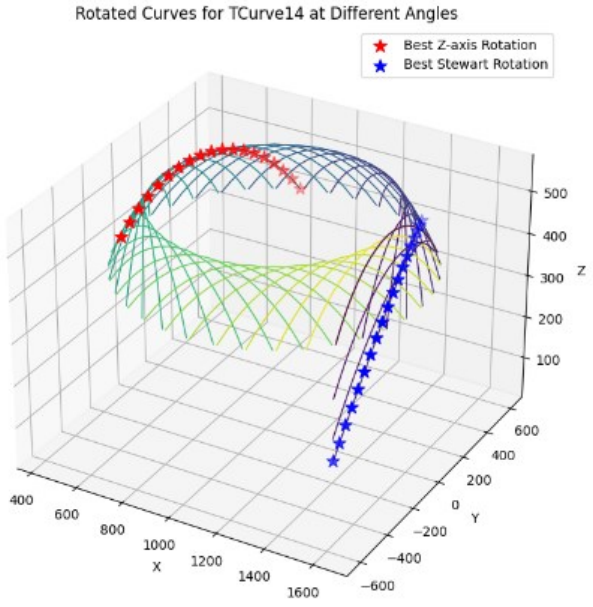

Figure 4.38 Different positions for Path 14 by movements of the PI H-840-D2A

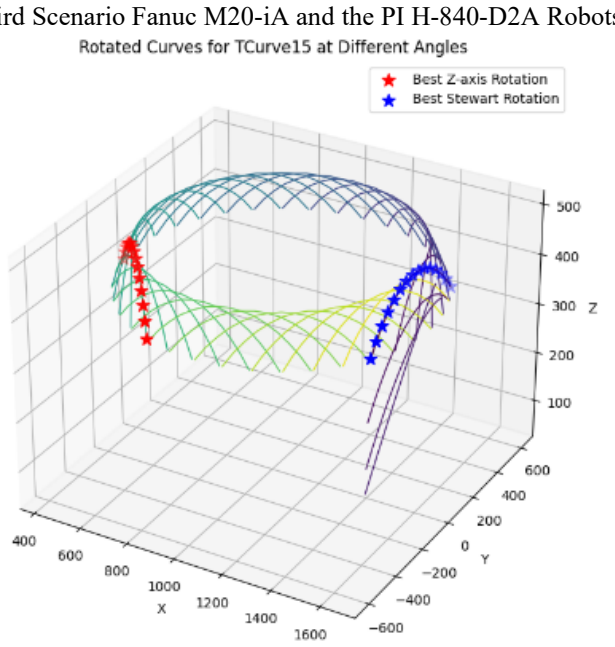

Figure 4.39 Different positions for Path 15 by movements of the PI H-840-D2A

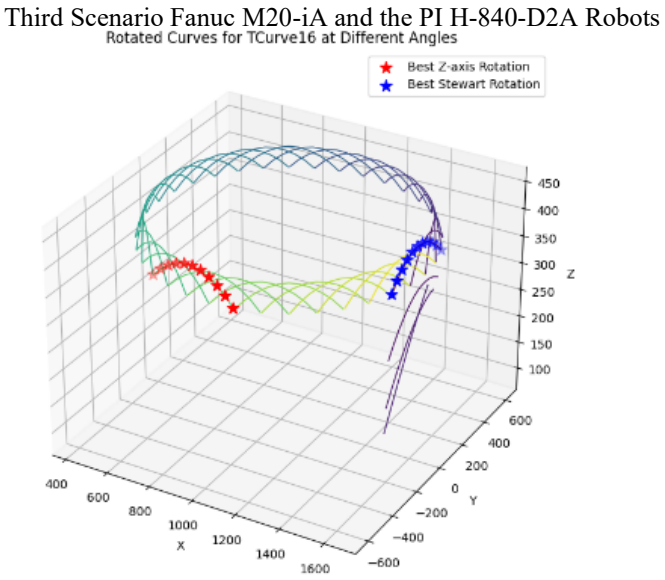

Figure 4.40 Different positions for Path 16 by movements of the PI H-840-D2A

Figure 4.41 shows the best results for the combination of the two rotations.

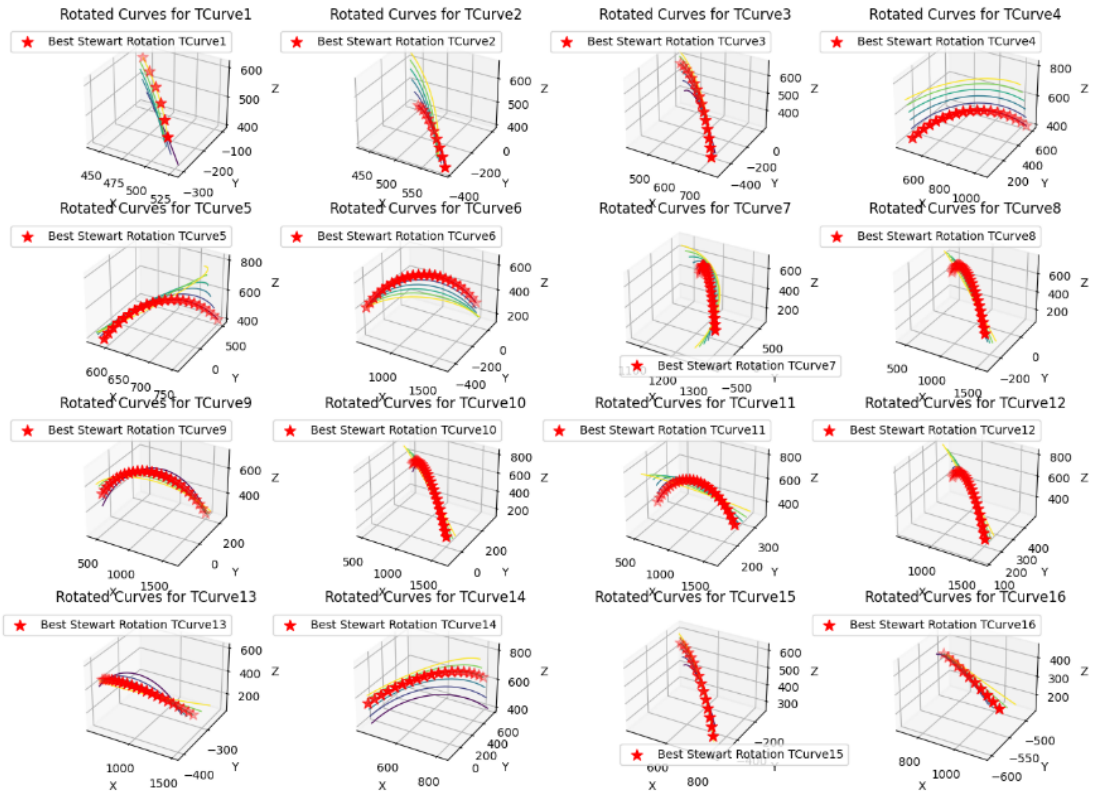

Figure 4.41 Best result for each path by movements of the PI H-840-D2A

4.4.6 Applying Optimal Rotations to Final Paths

Using the optimal rotation angles determined in Appendix D, the paths are adjusted for the fiber placement simulation. In Appendix E the final simulation and manipulability calculation has been done. Figure 4.42 shows the positions of the paths at their highest manipulability. As explained before, these paths will be under the fiber placement one by one after rotating the Parallel robot and the bulkhead. At the end of the simulation the orientation of the fibers will be 0°.

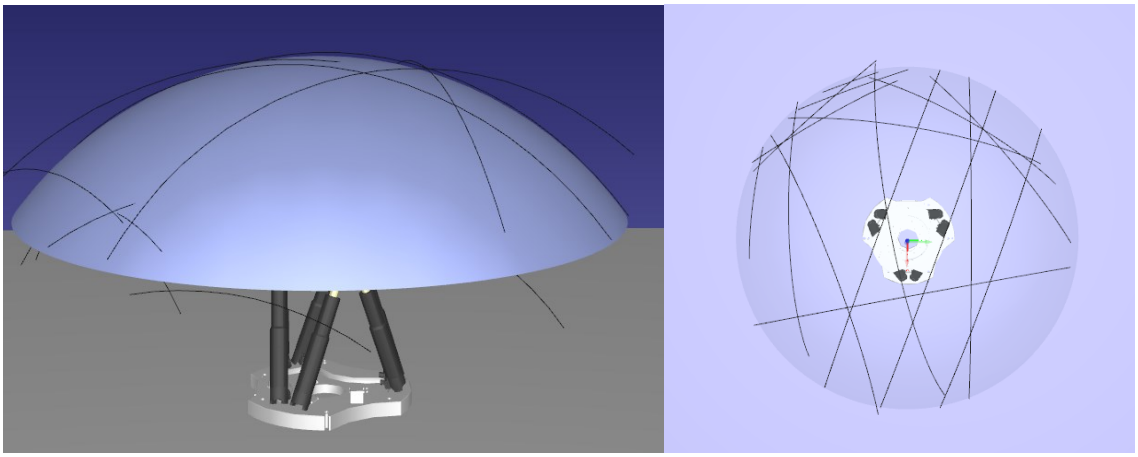

Figure 4.42 Front and Top view of Final Paths positions, in the fiber placement with the Fanuc M20-iA and the PI H-840-D2A

4.4.7 Moving the Robot Along Adjusted Paths

With the final paths prepared, the robot is commanded to move along each path, simulating the fiber placement process.

Movement Function is defined to move the robot along a given path, point by point. As before the center of the bulkhead is calculated and normal vectors on each path is determined. By constructing the transformation matrices for both robot's, the fiber placement simulation has been done. Error handling to manage any unreachable points or movement exceptions is included. The factors mentioned below have been ensured:

- Path Accuracy: Ensuring the robot follows the paths precisely to simulate accurate fiber placement.
- Collision Avoidance: Monitoring potential collisions with the component or other objects in the simulation environment.
- Performance: Efficiently moving the robot through all points without unnecessary delays or movements.

4.4.8 Integration of Both Codes and Justification of Rotation Choices

The main simulation code relies on the results obtained from the manipulations and evaluations conducted in the first code. The rotation angles calculated in the first code are applied in the main simulation to adjust the paths and robot orientations. The final paths used in the robot movement are derived from the adjusted paths after applying the optimal rotations. The rotations were chosen based on their ability to maximize the robot's manipulability along each path. By selecting angles that enhance manipulability, the robot can perform movements more efficiently, with reduced risk of encountering singularities or joint limits.

4.5 Results

In this section, the results of the three scenarios will be shown and compared. Figure 4.43 shows the results of the simulation with Fanuc M20-iA and placing the bulkhead on the KUKA KP1-V 500. As is shown in Figure 4.43, even the best positions for paths have a huge drop at some points in terms of manipulability and this also negatively affects the quality of the composite and the control on the robot and affects the speed of the robot too. It is also important to mention the difference between the results of the first codes and second codes in each scenario. The first code calculated the manipulability for the points that were defined on the paths, but the final simulation calculates the manipulability for 60 points on each path and sums the results.

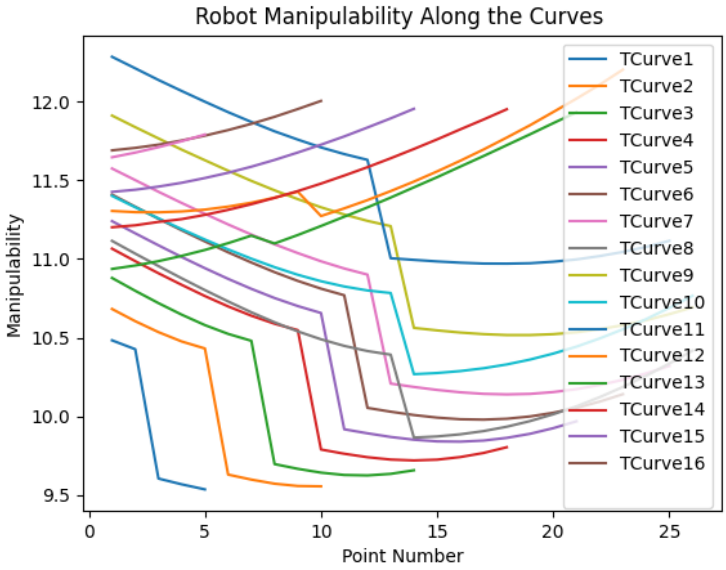

Figure 4.43 Manipulability of the points on the paths during the Fiber placement by Fanuc M20-ia on the bulkhead on the KUKA KP1-V 500

Figure 4.44 shows the results of the simulation with Fanuc M20-ia and placing the bulkhead on the ABB IRBP L600 L1250. On the contrary side of the previous simulation in some paths when there is a drop in the manipulability, there is a smooth change, which is an improvement but still there are some paths, for example path number 16 and 10, there is a sudden huge drop of manipulability which is also not good based on what mentioned for the results of Figure4.43.

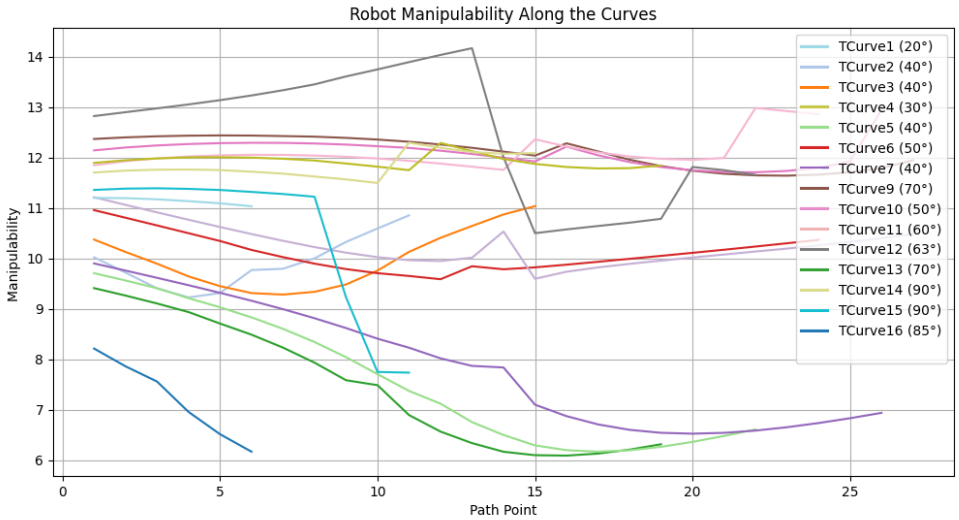

Figure 4.44 Manipulability of the points on the paths during the Fiber placement by Fanuc M20-ia on the bulkhead on the ABB IRBP L600 L1250

As mentioned, the higher manipulability gives us better control on the robot's movement which impacts the quality of the composite. Figure 4.45 shows the manipulability results of the fiber placement by the Fanuc M20-iA and the PI H-840-D2A robot. In the comparison between Figure 4.43 to Figure 4.45, The highest amount of manipulability has been reached with using the parallel robot as the second robot and after that the results from the situation that the ABB IRBP L600 L1250 has been used is better. Figure 4.45 shows that by using a parallel robot not only higher manipulability has been achieved but also huge drops have been avoided too. It has been shown that the speed of the manipulator has a direct effect on the material yield. If the speed of the end effector increases 12m/min, there might be a 20% increase in the material yield [69]. As mentioned, higher manipulability shows the capability of the robot of moving in different direction and shows being far from a singular pose which restricts the end effector's speed. Also, the change in the speed or fluctuation of the speed will cause vibration amplitude which affects the material's specifications[70].

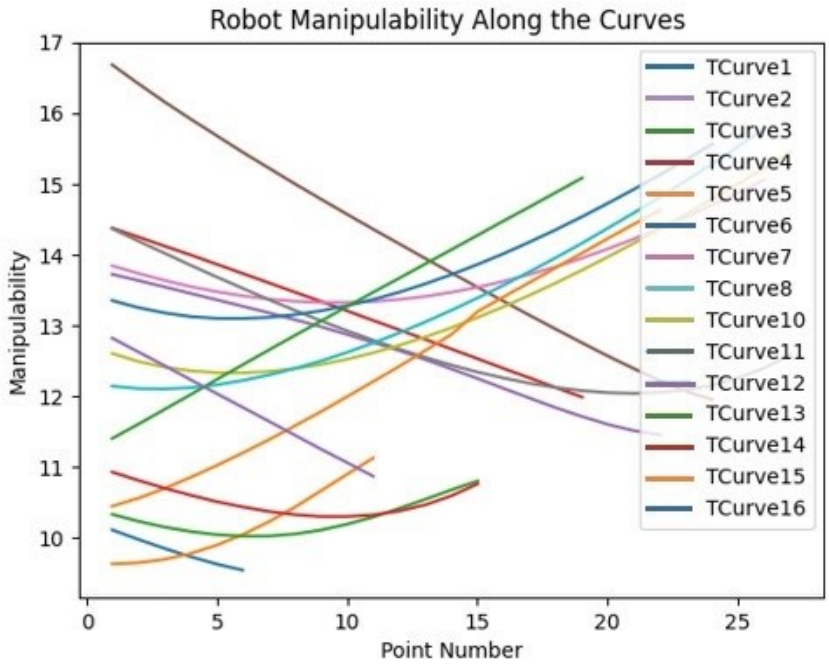

Figure 4.45 Manipulability of the points on the paths during the Fiber placement by Fanuc M20-ia on the bulkhead on the PI H-840-D2A

In the final simulations, 60 points on each path were created and the manipulability of those points was calculated. Table 4.3 shows the summation of the manipulability's of the points on each path during the fiber placement. It is shown that in most cases the parallel robot has given better positions to do a fiber placement while avoiding the drops in the results.

Table 4.3 Summation of the Manipulability of the points on each path for the three fiber placement scenarios

Path number	ABB IRBP L600 L1250 Summation of Manipulability (Y-axis rotation)	KUKA KP1-V 500 Summation of Manipulability (Z-axis rotation)	PI H-840-D2A Summation of Manipulability (Roll, Pitch, Yaw rotation)
Path1	1003.231	926.7873	1011.7423
Path2	849.7089	823.6066	857.4366
Path3	1059.602	848.5811	1081.929
Path4	1140.804	692.0615	1142.614
Path5	971.1791	777.0224	1110.492
Path6	1021.556	828.7502	1026.383
Path7	1146.399	1126.562	1147.948
Path8	1067.154	843.334	1165.131
Path9	1113.873	817.192	1116.775
Path10	1054.389	906.5593	1059.341
Path11	1140.599	979.2016	1140.599
Path12	968.3554	936.2477	968.3554
Path13	1092.495	798.5395	1099.572
Path14	1110.595	856.7592	1110.595
Path15	1100.34	981.1771	1100.719
Path16	926.7873	593.8729	1011.471

There are some cases where the other combinations seemed to have better functionality but considering the huge gaps that has happened and shown in Figure 4.43 and Figure 4.42, still the results for the combination of Fanuc and the parallel robot is the best option. Based on the results of the simulation the highest manipulability achieved in these configurations for some points was around 20, but it caused drops of manipulability or singularity positions for other points and it was not beneficial.

This systematic approach not only enhanced the reliability and efficiency of fiber placement but also contributed significantly to the advancement of automated composite manufacturing technologies.

4.6 Summary

This chapter explained the simulation environment and the algorithm with the results of the collaboration between the two robots in three different dual robot systems.

The simulation results in this chapter clearly demonstrated the potential of utilizing dual-robot systems for AFP on complex geometries. Specifically, by incorporating a parallel robot with 6 DOF, it is possible to significantly enhance the manipulability of the robotic system, reduce the risk of singularities and improve the overall efficiency of fiber placement. The findings suggested that a parallel robot is the optimal choice for maintaining high levels of control and precision in AFP applications in comparison with the robots with 1 DOF. This chapter serves as a critical step toward understanding how dual-robot configurations can enhance fiber placement in aerospace and other industries, ultimately paving the way for more reliable and automated production processes.

Next chapter will illustrate the conclusion and explain the contribution of this project and the possible future works after this project.

Chapter 5

5 Conclusion, Contributions, and Recommendations

5.1 Conclusion

This research demonstrates the potential of using a dual-robot system including a 6 DOF serial robot and a second robot with different DOFs in Automated Fiber Placement (AFP) to improve the precision, efficiency, and quality of fiber placement on complex and big geometric structures, such as pressure bulkheads. Three kinds of second robot have been explored in the simulation environment. The simulation results show that the integration of a serial manipulator (Fanuc M-20iA) and a parallel (PI H-840-D2A) robot outperforms the other two combinations in terms of flexibility and control increase, which is crucial when dealing with complex shapes in industries like aerospace and automotive.

Based on the simulations conducted in RoboDK, this study has shown that increasing the degrees of freedom (DOF) by incorporating multiple robots not only improves singularity avoidance but also gives the possibility of increasing the manipulability of the system. This improvement directly impacts the quality of the fiber placement, the speed, the size of the product that the robots can produce and gives more consistent and accurate results.

Workspace constraints also is important in the decision of choosing the robots for fiber placement. The compact design of the KP1-V 500 makes it suitable for environments where space is small and the larger size of the IRBP L600 L1250, while being capable of bearing heavier loads

and offering greater flexibility, requires more substantial floor space and may necessitate adjustments to the existing layout and finally the PI H-840-D2A gives the best flexibility and results and requires a smaller space.

The degrees of freedom offered by each positioner also influence their suitability for certain tasks. The single-axis rotation of the KP1-V 500 provides sufficient flexibility for applications where rotation around the vertical axis meets the process requirements. However, for more complex fiber placement tasks involving intricate geometry and multi-directional fiber paths, the dual-axis rotation of the IRBP L600 L1250 offers a significant advantage.

The manipulability factor was used to compare different robots' collaborations, and the results suggest that configurations with higher degrees of freedom led to higher manipulability which results in a material with fewer defects. This was particularly significant in avoiding singularity configuration, which often reduces the robot's performance during the fiber placement. This shows that Manipulability can be a good factor to consider during a fiber placement.

5.2 Contributions

A key contribution in this project is the usage of a serial robot and a parallel robot to perform a fiber placement on a pressure bulkhead. This approach significantly increased the ability to handle more complex geometry.

Utilization of the RoboDK software to simulate the combinations of the robots provided a cost-effective platform to optimize the trajectory without needing physical trials. This has reduced the development costs and time in AFP systems design.

This research used manipulability factor as a quantitative measure to evaluate robot's configuration and avoid singular configurations. This research contributes to broader utilization of the manipulability factor in industrial applications instead of the traditional singularity avoidance methods.

This study shows the singularity issues in dual robot systems which is a critical matter that can degrade the precision and control of the robots as well as the material's behavior. The research

offers practical strategies by showing the changes in singularity avoidance in case of changing and using different robots.

By using different robotic configurations, valuable insights have been shown in order to avoid common problems like misalignment, overlaps and gaps that happen during a fiber placement.

The work paves the way for future possible research in the Collaborative Robots field, particularly in case there is a need for high precision and efficiency.

5.3 Future works

For future research in this domain and for better Automated Fiber Placement (AFP) systems utilizing dual-robot configurations, several recommendations for exploration and advancement are possible to refine the current methodologies.

1- Refinement of Simulation Models

A refinement of the simulation models used in this research is needed to achieve greater accuracy in simulating real-world applications. Future studies should focus on incorporating additional geometric and physical parameters like a bicycle frame or a Y shape to achieve a better collaboration. Improvements in the kinematic and dynamic models could provide a more precise representation of the interaction between robotic arms and complex surfaces, ensuring higher fidelity in simulation results.

2- Expansion to Multi-Robot Coordination

The research demonstrates the feasibility of dual-robot systems to be used, however, introducing a third component to observe and track the robots are essential in experimental tests. This expansion would likely improve fiber placement on shapes, which positioning is important such as the aerospace structures.

3- Experimental Validation of Simulation Results

While this study mainly relies on simulation and theoretical based results, experimental validation is a crucial step to be taken. Future work should focus on the practical implementation of the dual-robot AFP system and compare their performances. Conducting real-world tests will

provide data to evaluate the effectiveness of this proposed system and it will identify any discrepancies and guide further refinements in both hardware and control algorithms.

4- Incorporation of Advanced Control Strategies

Techniques such as adaptive control, which allows real-time adjustments based on sensor feedback, could be investigated and used in smaller industries. This could optimize the fiber placement path smoothly and dynamically, avoiding issues such as singularities and improving the overall quality of fiber deposition and also reducing the time needed to simulate the fiber placement to do the path planning ahead of the actual fiber placement.

5- Enhancement of Path Planning Algorithms

Refining the path planning methods is always a solution which should be investigated. Current algorithms, particularly fixed and variable angle path planning techniques, could be changed to fulfil more complex geometric tasks.

In conclusion, these future works that have been proposed are critical to advance the state of dual-robot AFP systems. These developments will not only improve the system's efficiency and flexibility but also will start a new era in automated composite manufacturing.

References

[1] D. Groppe, “Robots improve the quality and cost-effectiveness of composite structures.” [Online]. Available: www.compositemfg.com

[2] “AERO - Boeing 787 from the Ground Up.” Accessed: Oct. 19, 2024. [Online]. Available: http://www.lb.boeing.com/commercial/aeromagazine/articles/qtr_4_06/article_04_3.html

[3] L. Brandt and M. Eckardt, “AEROSPACE Europe CEAS 2017 Conference Automated handling and positioning of large dry carbon fibre cut-pieces with cooperating robots in rear pressure bulkhead production,” 2017.

[4] I. Ahmed, B. Reddy, and A. Prof, “Stress Analysis of the Fuselage Cabin Pressure Bulkhead and Evaluation of Different Geometrical Configurations.” [Online]. Available: www.ijert.org

[5] W. Woigk, S. R. Hallett, M. I. Jones, M. Kuhtz, A. Hornig, and M. Gude, “Experimental investigation of the effect of defects in Automated Fibre Placement produced composite laminates,” *Compos Struct*, vol. 201, pp. 1004–1017, Oct. 2018, doi: 10.1016/j.compstruct.2018.06.078.

[6] Q. Chu, Y. Li, J. Xiao, D. Huan, and X. Chen, “Modeling and experimental investigation of out-of-plane deformation on mechanical performance of composites manufactured by ATL,” *Journal of Reinforced Plastics and Composites*, vol. 36, no. 5, pp. 347–359, Mar. 2017, doi: 10.1177/0731684416675747.

[7] J. Frketic, T. Dickens, and S. Ramakrishnan, “Automated manufacturing and processing of fiber-reinforced polymer (FRP) composites: An additive review of contemporary and modern techniques for advanced materials manufacturing,” *Addit Manuf*, vol. 14, pp. 69–86, Mar. 2017, doi: 10.1016/j.addma.2017.01.003.

[8] D. H. J. A. Lukaszewicz, C. Ward, and K. D. Potter, “The engineering aspects of automated prepreg layup: History, present and future,” *Compos B Eng*, vol. 43, no. 3, pp. 997–1009, Apr. 2012, doi: 10.1016/j.compositesb.2011.12.003.

[9] P. Li, X. Zhang, W. Xie, and S. Van Hoa, “Operation of the Collaborative Composite Manufacturing (CCM) System,” *Journal of Visualized Experiments*, no. 152, Oct. 2019, doi: 10.3791/59969.

[10] S. S. Krishnan, G. K. E, and C. Author, “STRESS ANALYSIS OF A REAR PRESSURE BULKHEAD OF THE FUSELAGE STRUCTURE AND FATIGUE LIFE ESTIMATION,” 2013. [Online]. Available: www.ijmerr.com

[11] R. Di Sante, “Fibre optic sensors for structural health monitoring of aircraft composite structures: Recent advances and applications,” Jul. 30, 2015, *MDPI AG*. doi: 10.3390/s150818666.

[12] A. Brasington, C. Sacco, J. Halbritter, R. Wehbe, and R. Harik, “Automated fiber placement: A review of history, current technologies, and future paths forward,” *Composites Part C: Open Access*, vol. 6, p. 100182, Oct. 2021, doi: 10.1016/j.jcomc.2021.100182.

[13] A. Saboukhi, “Designing and implementing a small-size Automated Fiber Placement (AFP) head capable of depositing thermoset layers on V-shape structures,” 2023.

[14] H. G. Karimiani, “Analysis of Residual Stresses in Thermoplastic Composites Manufactured by Automated Fiber Placement,” 2015.

[15] S. Zamotine, “FINAL CAPSTONE REPORT Team 10-FP Robot End Effector.”

[16] A. Brasington, B. Francis, M. Godbold, and R. Harik, “A review and framework for modeling methodologies to advance automated fiber placement,” Mar. 01, 2023, *Elsevier B.V.* doi: 10.1016/j.jcomc.2023.100347.

[17] Y. Li, Y. Xiao, L. Yu, K. Ji, and D. Li, “A review on the tooling technologies for composites manufacturing of aerospace structures: materials, structures and processes,” *Compos Part A Appl Sci Manuf*, vol. 154, p. 106762, Mar. 2022, doi: 10.1016/j.compositesa.2021.106762.

[18] A. W. Blom, C. S. Lopes, P. J. Kromwijk, Z. Gurdal, and P. P. Camanho, “A Theoretical Model to Study the Influence of Tow-drop Areas on the Stiffness and Strength of Variable-

stiffness Laminates," *J Compos Mater*, vol. 43, no. 5, pp. 403–425, Mar. 2009, doi: 10.1177/0021998308097675.

[19] X. Zhang, "Modeling and Control of the Cooperative Automated Fiber Placement System," 2017.

[20] A. Air, M. Shamsuddoha, E. Oromiehie, and B. G. Prusty, "Development of an automated fibre placement-based hybrid composite wheel for a solar-powered car," *International Journal of Advanced Manufacturing Technology*, vol. 125, no. 9–10, pp. 4083–4097, Apr. 2023, doi: 10.1007/s00170-023-10946-9.

[21] A. Pagani and A. R. Sánchez-Majano, "Analysis of the manufacturing signature on AFP-manufactures variable stiffness composite panels," in *Materials Research Proceedings*, Association of American Publishers, 2023, pp. 317–320. doi: 10.21741/9781644902813-69.

[22] "US10408603".

[23] X. Wen *et al.*, "Uncertainty Estimation of Robot Geometric Parameters and End-Effecter Position Based on New Generation GPS," *Math Probl Eng*, vol. 2019, 2019, doi: 10.1155/2019/7830489.

[24] B. Dasgupta and T. S. Mruthyunjaya, "The Stewart platform manipulator: a review," *Mech Mach Theory*, vol. 35, no. 1, pp. 15–40, Jan. 2000, doi: 10.1016/S0094-114X(99)00006-3.

[25] K. E. Zanganeh, R. Sinatra, and J. Angeles, "Kinematics and dynamics of a six-degree-of-freedom parallel manipulator with revolute legs," *Robotica*, vol. 15, no. 4, pp. 385–394, Jul. 1997, doi: 10.1017/S0263574797000477.

[26] M. N. Grimshaw, C. Machine, C. G. Grant, J. Manuel, and L. Diaz, "ADVANCED TECHNOLOGY TAPE LAYING FOR AFFORDABLE MANUFACTURING OF LARGE COMPOSITE STRUCTURES."

[27] W. ZHANG, F. LIU, T. JIANG, M. YI, W. CHEN, and X. DING, "Overview of current design and analysis of potential theories for automated fibre placement mechanisms," Apr. 01, 2022, *Elsevier B.V.* doi: 10.1016/j.cja.2021.04.018.

[28] C. Grant, “Automated processes for composite aircraft structure,” 2006. doi: 10.1108/01439910610651428.

[29] D. H. J. A. Lukaszewicz, C. Ward, and K. D. Potter, “The engineering aspects of automated prepreg layup: History, present and future,” *Compos B Eng*, vol. 43, no. 3, pp. 997–1009, Apr. 2012, doi: 10.1016/j.compositesb.2011.12.003.

[30] K. Singh Madhok, “COMPARATIVE CHARACTERIZATION OF OUT-OF-AUTOCLAVE MATERIALS MADE BY AUTOMATED FIBER PLACEMENT AND HAND-LAY-UP PROCESSES.”

[31] L. Izco, J. Isturiz, and M. Motilva, “High speed tow placement system for complex surfaces with cut/clamp/& restart capabilities at 85 m/min (3350 IPM),” *Additive Manufacturing of Aerospace Composite Structures: Fabrication and Reliability*, vol. 181, p. 1, 2017.

[32] L. Sorrentino, L. Carrino, L. Tersigni, and A. Leone, “Innovative tape placement robotic cell: High flexibility system to manufacture composite structural parts with variable thickness,” *J Manuf Sci Eng*, vol. 131, no. 4, pp. 0410021–0410028, 2009, doi: 10.1115/1.3160594.

[33] “Addcomposites | Plug and Place AFP.” Accessed: Oct. 19, 2024. [Online]. Available: https://pravinluthada.wixsite.com/addcomposites

[34] G. Marsh, “Automating aerospace composites production with fibre placement,” *Reinforced Plastics*, vol. 55, no. 3, pp. 32–37, May 2011, doi: 10.1016/S0034-3617(11)70075-3.

[35] W. Xie, “Design and Analysis of Collaborative Automated Fiber Placement Machine,” *International Journal of Advanced Robotics and Automation*, vol. 1, no. 1, pp. 01–14, Mar. 2016, doi: 10.15226/2473-3032/1/1/00105.

[36] G. Rousseau, R. Wehbe, J. Halbritter, and R. Harik, “Automated fiber placement path planning: A state-of-the-art review,” 2019, *CAD Solutions, LLC*. doi: 10.14733/cadaps.2019.172-203.

[37] J. J. Craig, P. Prentice, and P. P. Hall, “Introduction to Robotics Mechanics and Control Third Edition,” 2005.

[38] “RoboDK.” Accessed: Oct. 19, 2024. [Online]. Available: https://robodk.com/blog/robot-singularities/

[39] W. Xu, J. Zhang, B. Liang, and B. Li, “Singularity Analysis and Avoidance for Robot Manipulators With Nonspherical Wrists,” *IEEE Transactions on Industrial Electronics*, vol. 63, no. 1, pp. 277–290, Jan. 2016, doi: 10.1109/TIE.2015.2464176.

[40] F. Beck, M. N. Vu, C. Hartl-Nesic, and A. Kugi, “Singularity Avoidance with Application to Online Trajectory Optimization for Serial Manipulators,” *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 284–291, 2023, doi: 10.1016/j.ifacol.2023.10.1582.

[41] S. Yahya, M. Moghavvemi, and H. A. F. Mohamed, “Singularity avoidance of a six degree of freedom three dimensional redundant planar manipulator,” *Computers & Mathematics with Applications*, vol. 64, no. 5, pp. 856–868, Sep. 2012, doi: 10.1016/j.camwa.2011.12.073.

[42] M. J. Tsai and Y. H. Chiou, “Manipulability of manipulators,” *Mech Mach Theory*, vol. 25, no. 5, pp. 575–585, Jan. 1990, doi: 10.1016/0094-114X(90)90071-Q.

[43] L. Jin, S. Li, H. M. La, and X. Luo, “Manipulability Optimization of Redundant Manipulators Using Dynamic Neural Networks,” *IEEE Transactions on Industrial Electronics*, vol. 64, no. 6, pp. 4710–4720, Jun. 2017, doi: 10.1109/TIE.2017.2674624.

[44] G. Janez, K. Timi, G. Karl, and B. Miran, “Accuracy improvement of robotic machining based on robot’s structural properties,” *International Journal of Advanced Manufacturing Technology*, vol. 108, no. 5–6, pp. 1309–1329, May 2020, doi: 10.1007/s00170-020-05438-z.

[45] H.-K. Huang and G. C. I. Lin, “Development of a dual-robot system for prototype production,” *Int J Prod Res*, vol. 40, no. 15, pp. 3751–3764, Jan. 2002, doi: 10.1080/00207540210146206.

[46] J. F. Buhl *et al.*, “A Dual-arm Collaborative Robot System for the Smart Factories of the Future,” *Procedia Manuf*, vol. 38, pp. 333–340, 2019, doi: 10.1016/j.promfg.2020.01.043.

[47] D. Kruse, J. T. Wen, and R. J. Radke, "A Sensor-Based Dual-Arm Tele-Robotic System," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 1, pp. 4–18, Jan. 2015, doi: 10.1109/TASE.2014.2333754.

[48] D.-E. Kim, D.-J. Park, J.-H. Park, and J.-M. Lee, "Collision and Singularity Avoidance Path Planning of 6-DOF Dual-Arm Manipulator," 2018, pp. 195–207. doi: 10.1007/978-3-319-97589-4_17.

[49] A. Forcellese, T. Mancia, A. C. Russo, M. Simoncini, and A. Vita, "Robotic automated fiber placement of carbon fiber towpregs," *Materials and Manufacturing Processes*, vol. 37, no. 5, pp. 539–547, Apr. 2022, doi: 10.1080/10426914.2021.1885706.

[50] L. Li, D. Xu, X. Wang, and M. Tan, "A survey on path planning algorithms in robotic fibre placement," in *Proceedings of the 2015 27th Chinese Control and Decision Conference, CCDC 2015*, Institute of Electrical and Electronics Engineers Inc., Jul. 2015, pp. 4704–4709. doi: 10.1109/CCDC.2015.7162756.

[51] L. Yan, Z. C. Chen, Y. Shi, and R. Mo, "An accurate approach to roller path generation for robotic fibre placement of free-form surface composites," *Robot Comput Integr Manuf*, vol. 30, no. 3, pp. 277–286, 2014, doi: 10.1016/j.rcim.2013.10.007.

[52] K. Fayazbakhsh, M. Arian Nik, D. Pasini, and L. Lessard, "Defect layer method to capture effect of gaps and overlaps in variable stiffness laminates made by Automated Fiber Placement," *Compos Struct*, vol. 97, pp. 245–251, Mar. 2013, doi: 10.1016/j.compstruct.2012.10.031.

[53] C. S. Lopes, Z. Gürdal, and P. P. Camanho, "Variable-stiffness composite panels: Buckling and first-ply failure improvements over straight-fibre laminates," *Comput Struct*, vol. 86, no. 9, pp. 897–907, May 2008, doi: 10.1016/j.compstruc.2007.04.016.

[54] "Simulator for industrial robots and offline programming - RoboDK".

[55] S. Pieskä, J. Kaarela, and J. Mäkelä, "Simulation and programming experiences of collaborative robots for small-scale manufacturing," in *2018 2nd International Symposium on Small-Scale Intelligent Manufacturing Systems, SIMS 2018*, Institute of Electrical and Electronics Engineers Inc., May 2018, pp. 1–4. doi: 10.1109/SIMS.2018.8355303.

[56] P. Sivasankaran* and R. Karthikeyan, “Simulation of Robot Kinematic Motions using Collision Mapping Planner using Robo Dk Solver,” *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 11, pp. 21–27, Sep. 2020, doi: 10.35940/ijitee.J7588.0991120.

[57] M. Sága, V. Bulej, N. Čuboňova, I. Kuric, I. Virgala, and M. Eberth, “Case study: Performance analysis and development of robotized screwing application with integrated vision sensing system for automotive industry,” *Int J Adv Robot Syst*, vol. 17, no. 3, May 2020, doi: 10.1177/1729881420923997.

[58] “Fanuc M-20iA/20M Robot | Robots.com.” Accessed: Oct. 19, 2024. [Online]. Available: https://www.robots.com/industrial-robots/fanuc-m-20ia-20m

[59] A. Saboukhi, Y. Hedayatnasab, S. Van Hoa, W.-F. Xie, and F. Shadmehri, “Toward a compact AFP head capable of performing V-shape structures:Design and Implementation,” in *2024 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, IEEE, Jul. 2024, pp. 1038–1043. doi: 10.1109/AIM55361.2024.10637034.

[60] I. Zaplana, H. Hadfield, and J. Lasenby, “Closed-form solutions for the inverse kinematics of serial robots using conformal geometric algebra,” Sep. 2021, doi: 10.1016/j.mechmachtheory.2022.104835.

[61] J. Lenarcic, T. Bajd, and M. M. Stanišić, *Robot Mechanisms*, vol. 60. in Intelligent Systems, Control and Automation: Science and Engineering, vol. 60. Dordrecht: Springer Netherlands, 2013. doi: 10.1007/978-94-007-4522-3.

[62] “KUKA KP1-V 500 robot - RoboDK.” Accessed: Oct. 19, 2024. [Online]. Available: https://robodk.com/robot/KUKA/KP1-V-500

[63] “KP1-V single-axis positioner | KUKA AG.” Accessed: Oct. 19, 2024. [Online]. Available: https://www.kuka.com/en-us/products/robotics-systems/robot-periphery/positionierer/kp1-v

[64] “IRBP L Workpiece positioner - ABB Robotics - PDF Catalogs | Technical Documentation | Brochure.” Accessed: Oct. 19, 2024. [Online]. Available:

https://pdf.directindustry.com/pdf/abb-robotics/irbp-l-workpiece-positioner/30265-660655.html

[65] "H-840, 6-Axis Hexapod Positioning System, Precision 6-Axis Stage." Accessed: Oct. 19, 2024. [Online]. Available: https://www.physikinstrumente.com/en/products/parallel-kinematic-hexapods/h-840-6-axis-hexapod-700810/

[66] T. H. G. Megson, "Fuselages," *Aircraft Structures for Engineering Students*, pp. 643–651, 2013, doi: 10.1016/B978-0-08-096905-3.00040-1.

[67] "Why do airliners have 'pressure bulkheads'? - Aviation Stack Exchange." Accessed: Oct. 19, 2024. [Online]. Available: https://aviation.stackexchange.com/questions/31292/why-do-airliners-have-pressure-bulkheads

[68] G. B. Arfken, H. J. Weber, and F. E. Harris, "More Special Functions," *Mathematical Methods for Physicists*, pp. 871–933, 2013, doi: 10.1016/B978-0-12-384654-9.00018-9.

[69] J. Chen, T. Chen-Keat, M. Hojjati, A. Vallee, M. A. Octeau, and A. Yousefpour, "Impact of layup rate on the quality of fiber steering/cut-restart in automated fiber placement processes," *Science and Engineering of Composite Materials*, vol. 22, no. 2, pp. 165–173, Mar. 2015, doi: 10.1515/secm-2013-0257.

[70] C. Krombholz, M. Perner, M. Bock, and D. Röstermundt, "IMPROVING THE PRODUCTION QUALITY OF THE ADVANCED AUTOMATED FIBER PLACEMENT PROCESS BY MEANS OF ONLINE PATH CORRECTION".

Appendix A

Fanuc M20-iA and the KUKA KP1-V 500

```
from robodk import robolink, robomath
import csv
import numpy as np
import random
import math
import time

# Connect to RoboDK
RDK = robolink.Robolink()
robot = RDK.Item('Fanuc M-20iA')

# Delete previously generated paths
for i in range(1, 25):
    path_name = f"TPath{i}"
    path_item = RDK.Item(path_name)
    if path_item.Valid():
        path_item.Delete()

if robot.Valid():
    print('Robot is valid')
else:
    print('Robot is not valid')
    exit()
# store the final paths after rotation
base_path = "C:/Users/ASUS/Desktop/Paths/"
path_points_dict = {}
final_paths_dict = {}

# Read and transform path points
for i in range(1, 19):
    file_path = f"{base_path}Path{i}.csv"
    data_tuples = []

    try:
        with open(file_path, 'r') as csvfile:
            csvreader = csv.reader(csvfile)
            for row in csvreader:
                tuple_row = list(map(float, row))
                data_tuples.append(tuple_row)
    except Exception as e:
        print(f"Failed to read {file_path}: {e}")
```

```
        continue

    # Transformation matrix
    tx, ty, tz, alpha, beta, gamma = 1090, -10, 280, 90, 0, 90
        Rx   =   np.array([[1,   0,   0],   [0,   np.cos(np.radians(alpha)),   -
np.sin(np.radians(alpha))],            [0,            np.sin(np.radians(alpha)),
np.cos(np.radians(alpha))]])
    Ry = np.array([[np.cos(np.radians(beta)), 0, np.sin(np.radians(beta))], [0, 1,
0], [-np.sin(np.radians(beta)), 0, np.cos(np.radians(beta))]])
     Rz  =  np.array([[np.cos(np.radians(gamma)),  -np.sin(np.radians(gamma)),  0],
[np.sin(np.radians(gamma)), np.cos(np.radians(gamma)), 0], [0, 0, 1]])
    R = Rz @ Ry @ Rx
    T = np.eye(4)
    T[:3, :3] = R
    T[:3, 3] = [tx, ty, tz]

    transformed_points = []
    for value in data_tuples:
        point = np.array([value[0], value[1], value[2], 1])
        vector = np.array([value[3], value[4], value[5], 0])
        transformed_point = T @ point
        transformed_vector = (R @ vector[:3]) * 180 / np.pi
                  transformed_Tuple    =    np.concatenate((transformed_point[:3],
transformed_vector[:3]))
        transformed_Tuple_rounded = np.round(transformed_Tuple)
        transformed_points.append(list(transformed_Tuple_rounded[:6]))

    path_name = f"TPath{i}"
    adjusted_transformed_points = [[x, y, z - 525, rx, ry, rz] for x, y, z, rx,
ry, rz in transformed_points]
    path_points_dict[path_name] = adjusted_transformed_points

# Display paths
for i in range(1, 18):
    path_key = f"TPath{i}"
    F = path_points_dict.get(path_key, [])
    F = [[x, y, z + 530, rx, ry, rz] for x, y, z, rx, ry, rz in F]

                    path_item                    =                    RDK.AddPath(F,
projection_type=robolink.PROJECTION_ALONG_NORMAL_RECALC)
    path_item.setName(path_key)
    path_item.setVisible(True)

time.sleep(5)
```

```
# Delete the paths after showing them
for i in range(1, 18):
    path_key = f"TPath{i}"
    path_item = RDK.Item(path_key)
    if path_item.Valid():
        path_item.Delete()

########################################################################
# calculate sphere center and angles
def calculate_angles(F):
    selected_points = random.sample(F, 3)

    P1 = np.array(selected_points[0][:3])
    P2 = np.array(selected_points[1][:3])
    P3 = np.array(selected_points[2][:3])

    midpoint_P1_P2 = (P1 + P2) / 2
    midpoint_P1_P3 = (P1 + P3) / 2

    vec_P1_P2 = P2 - P1
    vec_P1_P3 = P3 - P1

    normal_P1_P2 = np.cross(vec_P1_P2, P3 - P2)
    normal_P1_P3 = np.cross(vec_P1_P3, P2 - P3)

    A = np.array([normal_P1_P2, normal_P1_P3])
      b  =  np.array([np.dot(normal_P1_P2,  midpoint_P1_P2),  np.dot(normal_P1_P3,
midpoint_P1_P3)])

    sphere_center, _, _, _ = np.linalg.lstsq(A, b, rcond=None)

    # Calculate the normal vectors
    normalized_vectors = []
    comparison_vector = np.array([-1.0, 0.0, -1.0])
    comparison_vector /= np.linalg.norm(comparison_vector)
    angles = []

    for point in F:
        point = np.array(point[:3])
        vector = point - sphere_center
        normalized_vector = vector / np.linalg.norm(vector)
        normalized_vectors.append(np.array(normalized_vector))

    # Compute reversed normal
    reversed_normals = [-vector for vector in normalized_vectors]
```

```
    for reversed_normal in reversed_normals:
        dot_product = np.dot(reversed_normal, comparison_vector)
        angle = np.arccos(dot_product)
        angles.append(angle)

    o = []
    for i in range(1, len(angles)):
        if angles[i] > angles[i-1]:
            o.append(angles[i]/2-math.pi)
        else:
            o.append(-math.pi-angles[i]/2)

    return o

##########################################################################
# Function to create the target pose name
def create_target_pose_name(point_index, path_index):
    return f'TP{point_index}C{path_index}'

# rotate a path around the Z-axis
def rotate_path_z_tool_frame(path_points, angle_deg, tool_frame):
    angle_rad = np.radians(angle_deg)
    Rz = np.array([
        [np.cos(angle_rad), -np.sin(angle_rad), 0],
        [np.sin(np.radians(angle_deg)), np.cos(np.radians(angle_deg)), 0],
        [0, 0, 1]
    ])
    rotated_points = []
    tool_position = np.array(tool_frame[:3])
    for point in path_points:
        position = np.array(point[:3])
        rotated_position = Rz @ (position - tool_position) + tool_position
        rotated_points.append(list(rotated_position) + point[3:])
    return rotated_points

# Sort points by Y-coordinate in descending order
def sort_points_by_y(points):
    return sorted(points, key=lambda point: point[1], reverse=True)

# Define specific rotations
fixed_rotations = {
    12: 70,
    13: 100,
    14: 90,
```

```
    15: 80,
    16: 60,
}

# Function to move the robot along the paths with collision avoidance
def move_robot_along_path(path_points, angles, path_index, rotated=False):
    for point, angle in zip(path_points, angles):
        x, y, z, rx, ry, rz = point
        z -= 530
        A = angle

        # Avoid collisions
        if path_index == 8 or path_index == 10:
            print(f"Adjusting for collision on path {path_index}")
            A = A + np.radians(45)

        if rotated:
                target_pose = robomath.transl(x, y, z) * robomath.rotx(A) *
robomath.roty(0) * robomath.rotz(math.pi / 2)
        else:
                target_pose = robomath.transl(x, y, z) * robomath.rotx(A) *
robomath.roty(0) * robomath.rotz(math.pi)

        # Move the robot to the target pose
        robot.setSpeed(5)
        try:
            robot.MoveJ(target_pose, blocking=True)
            print(f"Moved to {point[:3]} on path {path_index}")
        except Exception as e:
            print(f"Cannot reach point {point[:3]} on path {path_index}: {e}")
            return False
    return True

all_target_poses = {}
tool_frame = [1090, -10, 280, 90, 0, 90]

# Show and process paths 1 to 11 (without rotation)
for j in range(1, 12):
    path_key = f"TPath{j}"
    F = path_points_dict.get(path_key, [])
    F = sort_points_by_y(F)  # Sort points by Y-coordinate
    o = calculate_angles(F)
    F = [[x, y, z + 530, rx, ry, rz] for x, y, z, rx, ry, rz in F]  # Correct Z
adjustment
```

```
    # Add path to RoboDK for visualization
                                path_item                =                RDK.AddPath(F,
projection_type=robolink.PROJECTION_ALONG_NORMAL_RECALC)
    path_item.setName(path_key)
    path_item.setVisible(True)

    # Move the robot along this path
    success = move_robot_along_path(F, o, path_index=j, rotated=False)
    if not success:
        print(f"Path {j} could not be fully processed.")
    else:
        print(f"Successfully moved along Path {j}")

path_key = f"TPath12"
F = path_points_dict.get(path_key, [])
F = sort_points_by_y(F)

# Apply 70-degree rotation to path 12
F_rotated = rotate_path_z_tool_frame(F, fixed_rotations[12], tool_frame)
F_rotated_adjusted = [[x, y, z + 530, rx, ry, rz] for x, y, z, rx, ry, rz in
F_rotated]

o = calculate_angles(F_rotated_adjusted)

# Move the robot along the rotated path 12
success    =    move_robot_along_path(F_rotated_adjusted,    o,    path_index=12,
rotated=True)
if not success:
    print(f"Path 12 could not be fully processed.")
else:
    print(f"Successfully moved along Path 12 with 70-degree rotation")

# Show and process paths
for j in range(13, 19):
    path_key = f"TPath{j}"
    F = path_points_dict.get(path_key, [])
    F = sort_points_by_y(F)

    # Apply the fixed rotation
       F_rotated   =   rotate_path_z_tool_frame(F,   fixed_rotations.get(j,   80),
tool_frame)


    F_rotated_adjusted = [[x, y, z + 530, rx, ry, rz] for x, y, z, rx, ry, rz in
F_rotated]
```

```
    o = calculate_angles(F_rotated_adjusted)

    # Move the robot along this rotated path
       success   =   move_robot_along_path(F_rotated_adjusted,   o,   path_index=j,
rotated=True)
    if not success:
        print(f"Rotated Path {j} could not be fully processed.")
    else:
        print(f"Successfully moved along Rotated Path {j}")

    # Add the rotated path to RoboDK for visualization (after rotation)
                final_path_item        =        RDK.AddPath(F_rotated_adjusted,
projection_type=robolink.PROJECTION_ALONG_NORMAL_RECALC)
    final_path_item.setName(f"Rotated_TPath{j}")
    final_path_item.setVisible(True)

print("Robot has moved along all paths, and final rotated paths are displayed.")

 # Plotting the manipulability

for path_key, values in manipulability_data.items():
    plt.plot(values, label=path_key)

plt.title("Robot Manipulability Along the Paths")
plt.xlabel("Point Index")
plt.ylabel("Manipulability")
plt.legend()
plt.savefig("finalmanplotresults.png")
plt.show()
```

Appendix B

Fanuc M20-iA and ABB IRBP L600 L1250 Rotations

```
from robodk import robolink, robomath  # RoboDK API and utilities
import csv
import numpy as np
import math
import sys
import matplotlib.pyplot as plt  # For plotting manipulability

# ============================
# 1. Connect to RoboDK
# ============================
RDK = robolink.Robolink()

# ============================
# 2. Define and Validate Robots
# ============================
# Define the two robots: Fanuc M-20iA and ABB IRBP L600 L1250
fanuc_robot = RDK.Item('Fanuc M-20iA')
abb_robot = RDK.Item('ABB IRBP L600 L1250')

# Delete previously generated paths
for i in range(1, 25):
    path_name = f"TPath{i}"
    path_item = RDK.Item(path_name)
    if path_item.Valid():
        path_item.Delete()

# Validate robots
if fanuc_robot.Valid() and abb_robot.Valid():
    print('Both robots are valid')
else:
    print('One or both robots are not valid')
    sys.exit()

# ============================
# 3. Define Rotation Axis
# ============================
axis_point1 = np.array([1200, 640, 950])
axis_point2 = np.array([1200, -590, 950])

# ============================
```

```
# 4. Create Rotation Matrix Function For ABB
# ============================
def create_rotation_matrix(axis_p1, axis_p2, angle_deg):
    # Define the rotation axis
    axis_vector = axis_p2 - axis_p1
    axis_length = np.linalg.norm(axis_vector)
    if axis_length == 0:
        raise ValueError("The two points defining the rotation axis cannot be the
same.")
    axis_unit = axis_vector / axis_length
    ux, uy, uz = axis_unit

    # Convert angle to radians
    angle_rad = math.radians(angle_deg)

    # Rodrigues' rotation formula
    cos_theta = math.cos(angle_rad)
    sin_theta = math.sin(angle_rad)
    one_minus_cos = 1 - cos_theta

    # Rotation matrix components
    R = np.array([
        [cos_theta + ux**2 * one_minus_cos,
         ux * uy * one_minus_cos - uz * sin_theta,
         ux * uz * one_minus_cos + uy * sin_theta],

        [uy * ux * one_minus_cos + uz * sin_theta,
         cos_theta + uy**2 * one_minus_cos,
         uy * uz * one_minus_cos - ux * sin_theta],

        [uz * ux * one_minus_cos - uy * sin_theta,
         uz * uy * one_minus_cos + ux * sin_theta,
         cos_theta + uz**2 * one_minus_cos]
    ])

    # Homogeneous rotation matrix
    Rotation = np.eye(4)
    Rotation[:3, :3] = R

    #rotate around the axis
    Translation_to_origin = np.eye(4)
    Translation_to_origin[:3, 3] = -axis_p1

    Translation_back = np.eye(4)
    Translation_back[:3, 3] = axis_p1
```

```
    # Combined rotation matrix
    Rotation_full = Translation_back @ Rotation @ Translation_to_origin

    return Rotation_full

# ============================
# 5. Define Initial Transformation
# ============================
tx, ty, tz = 1200, 30, 520
alpha, beta, gamma = 90, 0, 90

# Create the initial transformation matrix
Rx = np.array([
    [1, 0, 0],
    [0, np.cos(np.radians(alpha)), -np.sin(np.radians(alpha))],
    [0, np.sin(np.radians(alpha)), np.cos(np.radians(alpha))]
])
Ry = np.array([
    [np.cos(np.radians(beta)), 0, np.sin(np.radians(beta))],
    [0, 1, 0],
    [-np.sin(np.radians(beta)), 0, np.cos(np.radians(beta))]
])
Rz = np.array([
    [np.cos(np.radians(gamma)), -np.sin(np.radians(gamma)), 0],
    [np.sin(np.radians(gamma)),  np.cos(np.radians(gamma)), 0],
    [0, 0, 1]
])
R_initial = Rz @ Ry @ Rx
T_initial = np.eye(4)
T_initial[:3, :3] = R_initial
T_initial[:3, 3] = [tx, ty, tz]

# ============================
# 6. Define Transformation and Rotation Functions
# ============================
def apply_transformations(point, vector, T_initial, Rotation_matrix):
    """
     Applies the initial transformation and then rotates the point around the
specified axis.
    """
    # Convert point to homogeneous coordinates
    point_homogeneous = np.array([point[0], point[1], point[2], 1])
    transformed_point = T_initial @ point_homogeneous
```

```
    # Convert orientation vector (Rx, Ry, Rz) to a direction vector
    transformed_vector = R_initial @ np.array(vector)

    # Apply rotation around the axis
    transformed_point_rotated = Rotation_matrix @ transformed_point
    # For vectors, only apply rotation (no translation)
    transformed_vector_rotated = Rotation_matrix[:3, :3] @ transformed_vector

    return transformed_point_rotated[:3], transformed_vector_rotated

def draw_rotation_axis(p1, p2, RDK_instance):
    """
    Draws the rotation axis in RoboDK for visualization.
    """
    line_name = "RotationAxis"
    line_points = [p1.tolist(), p2.tolist()]
    axis_path = RDK_instance.AddPath(line_points, None, robolink.PROJECTION_NONE)
    axis_path.setName(line_name)
    axis_path.setColor([1, 0, 0])
    axis_path.setVisible(True)

# ============================
# 7. Visualize Rotation Axis (Optional)
# ============================
draw_rotation_axis(axis_point1, axis_point2, RDK)

# ============================
# 8. Read and Store Original Paths
# ============================
base_path = "C:/Users/ASUS/Desktop/Paths/"
original_path_points_dict = {}

for i in range(1, 17):  # Loop through 16 paths
    file_path = f"{base_path}Path{i}.csv"
    data_tuples = []

    try:
        with open(file_path, 'r') as csvfile:
            csvreader = csv.reader(csvfile)
            for row in csvreader:
                if len(row) < 6:
                       print(f"Row in {file_path} does not have enough columns:
{row}")
                    continue
                # Extract position and orientation vectors
```

```
                tuple_row = list(map(float, row))
                point_xyz = tuple_row[:3]
                vector_rpy = tuple_row[3:6]

                data_tuples.append((point_xyz, vector_rpy))
    except Exception as e:
        print(f"Failed to read {file_path}: {e}")
        continue

    path_name = f"TPath{i}"
    original_path_points_dict[path_name] = data_tuples

print("Original paths data stored.")

# ============================
# 9. Define Helper Functions for Conversions and Movement
# ============================
def mat_to_numpy(mat_obj):
    """Converts a RoboDK Mat object to a NumPy array."""
    return np.array([[mat_obj[i, j] for j in range(4)] for i in range(4)])

def numpy_to_mat(numpy_array):
    """Converts a NumPy array to a RoboDK Mat object."""
    mat_obj = robomath.Mat(numpy_array.tolist())
    return mat_obj

# Function to calculate manipulability
def calculate_manipulability(robot):
    J = robot.Jacobian()
    if J is not None:
        J = np.array(J)
        if J.shape[0] >= 6 and J.shape[1] >= 6:
            manipulability = np.sqrt(np.linalg.det(J @ J.T))
            return manipulability
    return 0

# Function to move the robot along the paths
def move_robot_along_path(robot, path_points, path_index):
    manipulability_indices = []
    path_point_indices = []
    for idx, point_data in enumerate(path_points):
        position = point_data[:3]
        vector = point_data[3:6]

        # No Z adjustment needed
```

```
        position[2] -= 530

        # Convert position and vector to standard Python floats
        position = [float(pos) for pos in position]
        vector = [float(vec) for vec in vector]

        # The Z direction of the tool should be the vector at each point (reverse
Z)
        z_axis = np.array(vector)
        if np.linalg.norm(z_axis) == 0:
            print(f"Zero vector encountered at point {idx+1} on path {path_index}.
Skipping this point.")
            manipulability_indices.append(0)
            path_point_indices.append(len(manipulability_indices))
            continue
        z_axis = z_axis / np.linalg.norm(z_axis)

        # Define the Y direction (arbitrary, but perpendicular to Z)
        x_temp = np.array([1.0, 0.0, 0.0])
        if np.allclose(z_axis, x_temp) or np.allclose(z_axis, -x_temp):
            x_temp = np.array([0.0, 1.0, 0.0])
        y_axis = np.cross(z_axis, x_temp)
        y_axis = y_axis / np.linalg.norm(y_axis)

        # Recalculate X-axis to ensure orthogonality
        x_axis = np.cross(y_axis, z_axis)

        # Create rotation matrix
        R = np.column_stack((x_axis, y_axis, z_axis))

        # Create the pose matrix with standard floats
        pose_matrix = np.eye(4, dtype=float)
        pose_matrix[:3, :3] = R
        pose_matrix[:3, 3] = position

        # Convert to RoboDK pose
        robodk_pose = robomath.Mat(pose_matrix.tolist())

        # Move the robot using MoveJ
        try:
            robot.MoveJ(robodk_pose)
            print(f"Moved to point {idx+1} on path {path_index}")

            # Calculate manipulability
            manipulability = calculate_manipulability(robot)
```

```
            manipulability_indices.append(manipulability)
            path_point_indices.append(len(manipulability_indices))
        except Exception as e:
            print(f"Failed to move to point {idx+1} on path {path_index}: {e}")
            manipulability_indices.append(0)
            path_point_indices.append(len(manipulability_indices))
            continue
    return manipulability_indices, path_point_indices

# ============================
# 10. Store Original ABB Robot Pose
# ============================
abb_original_pose = abb_robot.Pose()
abb_original_pose_array = mat_to_numpy(abb_original_pose)

# ============================
# 11. Loop Over Rotation Angles
# ============================
rotation_angles = [0, 10, 20, 30, 40, 50]

for rotation_angle_deg in rotation_angles:
    print(f"\nProcessing rotation angle: {rotation_angle_deg} degrees")
    # Create the rotation matrix for this angle
        Rotation_matrix   =   create_rotation_matrix(axis_point1,   axis_point2,
rotation_angle_deg)

    # Rotate ABB robot
    # Reset ABB robot to original pose
    abb_robot.setPose(abb_original_pose)
    abb_pose_array = abb_original_pose_array.copy()

    # Apply rotation to ABB robot
    abb_pose_rotated = Rotation_matrix @ abb_pose_array
    abb_pose_rotated_mat = numpy_to_mat(abb_pose_rotated)

    abb_robot.setPose(abb_pose_rotated_mat)

    # Transform, Rotate, and Visualize Paths
    path_points_dict = {}

    for path_name, data_tuples in original_path_points_dict.items():
        transformed_data_tuples = []

        for point_xyz, vector_rpy in data_tuples:
            # Apply initial transformation and rotation
```

```
            transformed_xyz, transformed_vector = apply_transformations(point_xyz,
vector_rpy, T_initial, Rotation_matrix)

            transformed_tuple = list(transformed_xyz) + list(transformed_vector)
            transformed_data_tuples.append(transformed_tuple)

        path_points_dict[path_name] = transformed_data_tuples

        # Add the transformed path to RoboDK for visualization
        # Remove existing path if it exists
        path_item = RDK.Item(path_name)
        if path_item.Valid():
            path_item.Delete()

        # Add new path
        path_points_xyz = [tuple[:3] for tuple in transformed_data_tuples]
        if path_points_xyz:
                            path_item    =    RDK.AddPath(path_points_xyz,    None,
robolink.PROJECTION_ALONG_NORMAL_RECALC)
            path_item.setName(path_name)
            path_item.setVisible(True)
        else:
            print(f"No valid points to add for {path_name}.")

    print("Paths rotated and visualized in RoboDK.")

    # Move Fanuc Robot Along Paths and Calculate Manipulability
    total_manipulability_indices = []
    total_path_point_indices = []

    # Loop through the paths
    for path_name, data_points in path_points_dict.items():
        print(f"Processing {path_name}...")
                            manipulability_indices,      path_point_indices      =
move_robot_along_path(fanuc_robot, data_points, path_name)
        total_manipulability_indices.extend(manipulability_indices)
        total_path_point_indices.extend(path_point_indices)

      print(f"Finished  moving  the  robot  along  all  paths  for  rotation  angle
{rotation_angle_deg} degrees.")

    # Plot Manipulability for this rotation angle
    plt.figure()
             plt.plot(total_path_point_indices,     total_manipulability_indices,
label=f'Rotation {rotation_angle_deg}°')
```

```
    plt.xlabel('Path Point Index')
    plt.ylabel('Manipulability')
          plt.title(f'Robot    Manipulability    Along    the    Path    (Rotation
{rotation_angle_deg}°)')
    plt.legend()
    plt.grid(True)
    plt.show()

    # Optionally, you can save the plot
    # plt.savefig(f'Manipulability_Rotation_{rotation_angle_deg}.png')

    # Reset Fanuc robot to home position if necessary
    fanuc_robot.setJoints([0, 0, 0, 0, 0, 0])

print("All rotations processed.")
```

Appendix C

Fanuc M20-ia and ABB IRBP L600 L1250

```
from robodk import robolink, robomath  # RoboDK API and utilities
import csv
import numpy as np
import math
import sys
import matplotlib.pyplot as plt  # For plotting manipulability

# ============================
# 1. Connect to RoboDK
# ============================
RDK = robolink.Robolink()

# ============================
# 2. Define and Validate Robots
# ============================
# Define the two robots: Fanuc M-20iA and ABB IRBP L600 L1250
fanuc_robot = RDK.Item('Fanuc M-20iA')
abb_robot = RDK.Item('ABB IRBP L600 L1250')

# Delete previously generated paths
for i in range(1, 25):
    path_name = f"TPath{i}"
    path_item = RDK.Item(path_name)
    if path_item.Valid():
        path_item.Delete()

# Validate robots
if fanuc_robot.Valid() and abb_robot.Valid():
    print('Both robots are valid')
else:
    print('One or both robots are not valid')
    sys.exit()

# ============================
# 3. Define Rotation Axis
# ============================
# Given Points defining the rotation axis, shifted 525 units higher in Z
axis_point1 = np.array([1200, 640, 950])        # Original [1200, 640, 425] + 525
in Z
```

```
axis_point2 = np.array([1200, -590, 950])       # Original [1200, -590, 425] + 525
in Z

# ============================
# 4. Create Rotation Matrix Function
# ============================
def create_rotation_matrix(axis_p1, axis_p2, angle_deg):
    """
    Creates a homogeneous rotation matrix to rotate points around the line defined
by axis_p1 and axis_p2 by angle_deg degrees.
    """
    # Define the rotation axis (unit vector)
    axis_vector = axis_p2 - axis_p1
    axis_length = np.linalg.norm(axis_vector)
    if axis_length == 0:
        raise ValueError("The two points defining the rotation axis cannot be the
same.")
    axis_unit = axis_vector / axis_length
    ux, uy, uz = axis_unit

    # Convert angle to radians
    angle_rad = math.radians(angle_deg)

    # Rodrigues' rotation formula
    cos_theta = math.cos(angle_rad)
    sin_theta = math.sin(angle_rad)
    one_minus_cos = 1 - cos_theta

    # Rotation matrix components
    R = np.array([
        [cos_theta + ux**2 * one_minus_cos,
         ux * uy * one_minus_cos - uz * sin_theta,
         ux * uz * one_minus_cos + uy * sin_theta],

        [uy * ux * one_minus_cos + uz * sin_theta,
         cos_theta + uy**2 * one_minus_cos,
         uy * uz * one_minus_cos - ux * sin_theta],

        [uz * ux * one_minus_cos - uy * sin_theta,
         uz * uy * one_minus_cos + ux * sin_theta,
         cos_theta + uz**2 * one_minus_cos]
    ])

    # Homogeneous rotation matrix
    Rotation = np.eye(4)
```

```
    Rotation[:3, :3] = R

    # To rotate around the axis, translate so that axis_p1 is at the origin, apply
rotation, then translate back
    Translation_to_origin = np.eye(4)
    Translation_to_origin[:3, 3] = -axis_p1

    Translation_back = np.eye(4)
    Translation_back[:3, 3] = axis_p1

    # Combined rotation matrix
    Rotation_full = Translation_back @ Rotation @ Translation_to_origin

    return Rotation_full

# ============================
# 5. Define Initial Transformation
# ============================
# Transformation matrix parameters
tx, ty, tz = 1200, 30, 520
alpha, beta, gamma = 90, 0, 90

# Create the initial transformation matrix
Rx = np.array([
    [1, 0, 0],
    [0, np.cos(np.radians(alpha)), -np.sin(np.radians(alpha))],
    [0, np.sin(np.radians(alpha)), np.cos(np.radians(alpha))]
])
Ry = np.array([
    [np.cos(np.radians(beta)), 0, np.sin(np.radians(beta))],
    [0, 1, 0],
    [-np.sin(np.radians(beta)), 0, np.cos(np.radians(beta))]
])
Rz = np.array([
    [np.cos(np.radians(gamma)), -np.sin(np.radians(gamma)), 0],
    [np.sin(np.radians(gamma)),  np.cos(np.radians(gamma)), 0],
    [0, 0, 1]
])
R_initial = Rz @ Ry @ Rx  # Combine rotations
T_initial = np.eye(4)  # Identity matrix for transformation
T_initial[:3, :3] = R_initial
T_initial[:3, 3] = [tx, ty, tz]  # Apply translation

# ============================
# 6. Define Transformation and Rotation Functions
```

```
# ============================
def apply_transformations(point, vector, T_initial, Rotation_matrix):
    """
     Applies the initial transformation and then rotates the point around the
specified axis.
    """
    # Convert point to homogeneous coordinates
    point_homogeneous = np.array([point[0], point[1], point[2], 1])
    transformed_point = T_initial @ point_homogeneous

    # Convert orientation vector (Rx, Ry, Rz) to a direction vector
    transformed_vector = R_initial @ np.array(vector)

    # Apply rotation around the axis
    transformed_point_rotated = Rotation_matrix @ transformed_point
    # For vectors, only apply rotation (no translation)
    transformed_vector_rotated = Rotation_matrix[:3, :3] @ transformed_vector

    return transformed_point_rotated[:3], transformed_vector_rotated

def draw_rotation_axis(p1, p2, RDK_instance):
    """
    Draws the rotation axis in RoboDK for visualization.
    """
    line_name = "RotationAxis"
    line_points = [p1.tolist(), p2.tolist()]
    axis_path = RDK_instance.AddPath(line_points, None, robolink.PROJECTION_NONE)
    axis_path.setName(line_name)
    axis_path.setColor([1, 0, 0])  # Red color for the axis
    axis_path.setVisible(True)

# ============================
# 7. Visualize Rotation Axis (Optional)
# ============================
draw_rotation_axis(axis_point1, axis_point2, RDK)

# ============================
# 8. Read and Store Original Paths
# ============================
base_path = "C:/Users/ASUS/Desktop/Paths/"
original_path_points_dict = {}

for i in range(1, 17):  # Loop through 16 paths
    file_path = f"{base_path}Path{i}.csv"
    data_tuples = []
```

```
    try:
        with open(file_path, 'r') as csvfile:
            csvreader = csv.reader(csvfile)
            for row in csvreader:
                if len(row) < 6:
                       print(f"Row in {file_path} does not have enough columns:
{row}")
                    continue
                # Extract position and orientation vectors
                tuple_row = list(map(float, row))
                point_xyz = tuple_row[:3]
                vector_rpy = tuple_row[3:6]

                data_tuples.append((point_xyz, vector_rpy))
    except Exception as e:
        print(f"Failed to read {file_path}: {e}")
        continue

    path_name = f"TPath{i}"
    original_path_points_dict[path_name] = data_tuples

print("Original paths data stored.")

# ============================
# 9. Define Helper Functions for Conversions and Movement
# ============================
def mat_to_numpy(mat_obj):
    """Converts a RoboDK Mat object to a NumPy array."""
    return np.array([[mat_obj[i, j] for j in range(4)] for i in range(4)])

def numpy_to_mat(numpy_array):
    """Converts a NumPy array to a RoboDK Mat object."""
    mat_obj = robomath.Mat(numpy_array.tolist())
    return mat_obj

# Function to move the robot along the paths
def move_robot_along_path(robot, path_points, path_index):
    manipulability_indices = []
    path_point_indices = []

    # Retrieve joint limits
    joint_limits = robot.JointLimits()
    min_limits = np.array(joint_limits[0])
    max_limits = np.array(joint_limits[1])
```

```
    for idx, point_data in enumerate(path_points):
        position = point_data[:3]
        vector = point_data[3:6]

        # Adjust Z position by -530 (as per your requirement)
        position[2] -= 530

        # Convert position and vector to standard Python floats
        position = [float(pos) for pos in position]
        vector = [float(vec) for vec in vector]

        # The Z direction of the tool should be the vector at each point (reverse
Z)
        z_axis = np.array(vector)
        if np.linalg.norm(z_axis) == 0:
            print(f"Zero vector encountered at point {idx+1} on path {path_index}.
Skipping this point.")
            manipulability_indices.append(0)
            path_point_indices.append(len(manipulability_indices))
            continue
        z_axis = z_axis / np.linalg.norm(z_axis)

        # Define the Y direction (arbitrary, but perpendicular to Z)
        x_temp = np.array([1.0, 0.0, 0.0])
        if np.allclose(z_axis, x_temp) or np.allclose(z_axis, -x_temp):
            x_temp = np.array([0.0, 1.0, 0.0])
        y_axis = np.cross(z_axis, x_temp)
        y_axis = y_axis / np.linalg.norm(y_axis)

        # Recalculate X-axis to ensure orthogonality
        x_axis = np.cross(y_axis, z_axis)

        # Create rotation matrix
        R = np.column_stack((x_axis, y_axis, z_axis))

        # Create the pose matrix with standard floats
        pose_matrix = np.eye(4, dtype=float)
        pose_matrix[:3, :3] = R
        pose_matrix[:3, 3] = position

        # Convert to RoboDK pose
        robodk_pose = robomath.Mat(pose_matrix.tolist())

        # Move the robot using MoveJ
```

```
        try:
            robot.MoveJ(robodk_pose)
            print(f"Moved to point {idx+1} on path {path_index}")

            # Get the joint angles
            current_joints = np.array(robot.Joints().tolist())

            # Normalize the joints and calculate manipulability
            norm_joints = (current_joints - min_limits) / (max_limits - min_limits)
            manipulability = np.linalg.norm(norm_joints)
            manipulability_indices.append(manipulability)
            path_point_indices.append(len(manipulability_indices))

        except Exception as e:
            print(f"Failed to move to point {idx+1} on path {path_index}: {e}")
            manipulability_indices.append(0)
            path_point_indices.append(len(manipulability_indices))
            continue
    return manipulability_indices, path_point_indices

# ============================
# 10. Store Original ABB Robot Pose
# ============================
abb_original_pose = abb_robot.Pose()
abb_original_pose_array = mat_to_numpy(abb_original_pose)

# ============================
# 11. Define Specific Rotations for Paths
# ============================
# Updated mapping of paths to their rotation angles
paths_rotation = {
    1: 10,
    2: 20,
    3: 20,
    4: 20,
    5: 40,
    6: 50,
    7: 50,
    9: 50,
    10: 60,
    11: 60,
    12: 63,
    13: 75,
    14: 80,
    15: 80,
```

```
    16: 85,
}

# ============================
# 12. Process Each Path Individually
# ============================
manipulability_data = {}
path_indices_all = []
path_point_indices_all = []
manipulability_values_all = []

for path_num, rotation_angle_deg in paths_rotation.items():
    path_name = f"TPath{path_num}"
    data_tuples = original_path_points_dict.get(path_name, [])

    if not data_tuples:
        print(f"No data for {path_name}, skipping.")
        continue

     print(f"\nProcessing {path_name} with rotation angle: {rotation_angle_deg}
degrees")

    # Reset ABB robot to original pose
    abb_robot.setPose(abb_original_pose)

    # Create the rotation matrix for this angle
        Rotation_matrix = create_rotation_matrix(axis_point1, axis_point2,
rotation_angle_deg)

    # Apply rotation to ABB robot
    abb_pose_rotated = Rotation_matrix @ abb_original_pose_array
    abb_pose_rotated_mat = numpy_to_mat(abb_pose_rotated)
    abb_robot.setPose(abb_pose_rotated_mat)

    # Transform and Rotate Path Points
    transformed_data_tuples = []
    for point_xyz, vector_rpy in data_tuples:
        # Apply initial transformation and rotation
          transformed_xyz, transformed_vector = apply_transformations(point_xyz,
vector_rpy, T_initial, Rotation_matrix)

        transformed_tuple = list(transformed_xyz) + list(transformed_vector)
        transformed_data_tuples.append(transformed_tuple)

    # Update path_points_dict for this path
```

```
    path_points_dict = {path_name: transformed_data_tuples}

    # Remove existing path if it exists
    path_item = RDK.Item(path_name)
    if path_item.Valid():
        path_item.Delete()

    # Add new path to RoboDK for visualization
    path_points_xyz = [tuple[:3] for tuple in transformed_data_tuples]
    if path_points_xyz:
                        path_item     =     RDK.AddPath(path_points_xyz,     None,
robolink.PROJECTION_ALONG_NORMAL_RECALC)
        path_item.setName(path_name)
        path_item.setVisible(True)
    else:
        print(f"No valid points to add for {path_name}.")

    # Move Fanuc Robot Along the Path and Calculate Manipulability
    print(f"Moving robot along {path_name}...")
                    manipulability_indices,         path_point_indices         =
move_robot_along_path(fanuc_robot, transformed_data_tuples, path_name)

    # Store data for plotting
    manipulability_data[path_name] = {
        'manipulability': manipulability_indices,
        'path_indices': path_point_indices,
        'rotation_angle': rotation_angle_deg
    }

    # Collect data for overall plotting
    path_indices_all.extend([path_num] * len(manipulability_indices))
    path_point_indices_all.extend(path_point_indices)
    manipulability_values_all.extend(manipulability_indices)

    # Reset ABB robot to original pose (0 degrees rotation)
    abb_robot.setPose(abb_original_pose)
    print(f"Completed processing {path_name}.")

    #  reset Fanuc robot to home position if necessary
    #fanuc_robot.setJoints([0, 0, 0, 0, 0, 0])

# ============================
# 13. Plotting the Manipulability
# ============================
plt.figure(figsize=(12, 6))
```

```
# Define colors for the paths
color_map = plt.cm.get_cmap('tab20', len(paths_rotation))

# Plot each path's manipulability data
for idx, (path_name, data) in enumerate(manipulability_data.items(), start=1):
    plt.plot(
        data['path_indices'],
        data['manipulability'],
        label=f"{path_name} ({data['rotation_angle']}°)",
        color=color_map(idx - 1)
    )

plt.xlabel('Path Point')
plt.ylabel('Manipulability ')
plt.title('Robot Manipulability Along the Paths')
plt.legend()
plt.grid(True)
plt.show()


# plt.savefig('Final_Manipulability_Plot.png')

print("All paths processed and manipulability plotted.")
```

Appendix D

Best rotations check of the FANUC M20-ia and the Parallel robot

```
from robodk import robolink, robomath
import csv
import numpy as np
import math
import matplotlib.pyplot as plt
import random

# Connect to RoboDK
RDK = robolink.Robolink()
robot = RDK.Item('Fanuc M-20iA')
par = RDK.Item('PI H-840-D2A')

if robot.Valid():
    print('Item selected: ' + robot.Name())
    print('Item position: ' + repr(robot.Pose()))

target_ref = robot.Pose()
pos_ref = target_ref.Pos()
joints = robot.Joints()

robot.MoveJ(target_ref)
robot.setPoseFrame(robot.PoseFrame())
robot.setPoseTool(robot.PoseTool())

# Load path 3
base_path = "C:/Users/ASUS/Desktop/Rotary Python/p and s/"
path_index = 3
file_path = f"{base_path}Path{path_index}.csv"
data_tuples = []

with open(file_path, 'r') as csvfile:
    csvreader = csv.reader(csvfile)
    for row in csvreader:
        tuple_row = list(map(float, row))
        data_tuples.append(tuple_row)

# Transform points
tx, ty, tz, alpha, beta, gamma = 1050, 0, -49, 90, 0, 90
alpha_rad, beta_rad, gamma_rad = np.radians([alpha, beta, gamma])

Rx = np.array([[1, 0, 0],
```

```
                [0, np.cos(alpha_rad), -np.sin(alpha_rad)],
                [0, np.sin(alpha_rad), np.cos(alpha_rad)]])
Ry = np.array([[np.cos(beta_rad), 0, np.sin(beta_rad)],
                [0, 1, 0],
                [-np.sin(beta_rad), 0, np.cos(beta_rad)]])
Rz = np.array([[np.cos(gamma_rad), -np.sin(gamma_rad), 0],
                [np.sin(gamma_rad), np.cos(gamma_rad), 0],
                [0, 0, 1]])
R = Rz @ Ry @ Rx
T = np.eye(4)
T[:3, :3] = R
T[:3, 3] = [tx, ty, tz]

transformed_points = []
adjusted_transformed_points = []

for value in data_tuples:
    point = np.array([value[0], value[1], value[2], 1])
    vector = np.array([value[3], value[4], value[5], 0])
    transformed_point = T @ point
    transformed_vector = (R @ vector[:3]) * 180 / np.pi
              transformed_Tuple      =      np.concatenate((transformed_point[:3],
transformed_vector[:3]))
    transformed_Tuple_rounded = np.round(transformed_Tuple)
    transformed_points.append(list(transformed_Tuple_rounded[:6]))

    # Adjust Z value
    adjusted_point = list(transformed_Tuple_rounded[:3])
    adjusted_point[2] -= 525
                      adjusted_transformed_points.append(adjusted_point          +
list(transformed_Tuple_rounded[3:]))

path_points_dict = {}
adjusted_path_points_dict = {}
path_points_dict[f"TPath{path_index}"] = transformed_points
adjusted_path_points_dict[f"TPath{path_index}"] = adjusted_transformed_points

path_name = f"TPath{path_index}"
path_item                      =                     RDK.AddPath(transformed_points,
projection_type=robolink.PROJECTION_ALONG_NORMAL_RECALC)
path_item.setName(path_name)
print(f"Path created: {path_name}")

# Function to create the target pose name
def create_target_pose_name(point_index, path_index):
```

```
    return f'TP{point_index}C{path_index}'

# Function to check manipulability along a path
def check_path_manipulability(path_points, path_index):
    best_manipulability = -1
    best_pose = None
    best_point = None
    for idx, point in enumerate(path_points):
        x, y, z, rx, ry, rz = point
            target_pose = robomath.transl(x, y, z) * robomath.rotx(rx) *
robomath.roty(ry) * robomath.rotz(rz)
        robot.MoveJ(target_pose, blocking=True)
        manipulability = calculate_manipulability(robot, target_pose)
        if manipulability > best_manipulability:
            best_manipulability = manipulability
            best_pose = target_pose
            best_point = point
    return best_manipulability, best_pose, best_point

# Function to calculate manipulability
def calculate_manipulability(robot, pose):
    robot.MoveJ(pose, blocking=True)
    joints = robot.Joints().tolist()

    manipulability = sum(abs(joint) for joint in joints)
    return manipulability

# Function to calculate sphere center (assuming a method to calculate it is
provided)
def calculate_sphere_center(points):
    selected_points = random.sample(points, 3)
    P1 = np.array(selected_points[0][:3])
    P2 = np.array(selected_points[1][:3])
    P3 = np.array(selected_points[2][:3])

    midpoint_P1_P2 = (P1 + P2) / 2
    midpoint_P1_P3 = (P1 + P3) / 2

    vec_P1_P2 = P2 - P1
    vec_P1_P3 = P3 - P1

    normal_P1_P2 = np.cross(vec_P1_P2, P3 - P2)
    normal_P1_P3 = np.cross(vec_P1_P3, P2 - P3)

    A = np.array([normal_P1_P2, normal_P1_P3])
```

```
    b = np.array([np.dot(normal_P1_P2, midpoint_P1_P2), np.dot(normal_P1_P3,
midpoint_P1_P3)])

    sphere_center, _, _, _ = np.linalg.lstsq(A, b, rcond=None)
    return sphere_center

# Function to rotate a path around the Z-axis and apply pitch and yaw around the
sphere center
def rotate_path_with_stewart_platform(path_points, z_angle_deg, pitch_deg,
yaw_deg, center):
    z_angle_rad = np.radians(z_angle_deg)
    pitch_rad = np.radians(pitch_deg)
    yaw_rad = np.radians(yaw_deg)

    Rz = np.array([
        [np.cos(z_angle_rad), -np.sin(z_angle_rad), 0],
        [np.sin(z_angle_rad), np.cos(z_angle_rad), 0],
        [0, 0, 1]
    ])
    Rpitch = np.array([
        [1, 0, 0],
        [0, np.cos(pitch_rad), -np.sin(pitch_rad)],
        [0, np.sin(pitch_rad), np.cos(pitch_rad)]
    ])
    Ryaw = np.array([
        [np.cos(yaw_rad), 0, np.sin(yaw_rad)],
        [0, 1, 0],
        [-np.sin(yaw_rad), 0, np.cos(yaw_rad)]
    ])

    R = Rz @ Rpitch @ Ryaw
    rotated_points = []

    for point in path_points:
        position = np.array(point[:3])
        rotated_position = R @ (position - center) + center
        rotated_points.append(list(rotated_position) + point[3:])

    return rotated_points

# Sort points by Y-coordinate in descending order
def sort_points_by_y(points):
    return sorted(points, key=lambda point: point[1], reverse=True)

# Main processing to check manipulability and handle rotation
```

```
path_key = f"TPath{path_index}"
F = adjusted_path_points_dict.get(path_key, [])
F = sort_points_by_y(F)

# Calculate the sphere center
sphere_center = calculate_sphere_center(F)

# Varying angles for Z, pitch, and yaw
angles = range(0, 361, 10)
manipulabilities = []

for angle in angles:
    F_rotated = rotate_path_with_stewart_platform(F, angle, angle, angle,
sphere_center)
    best_manipulability, best_pose, best_point =
check_path_manipulability(F_rotated, path_index)
    manipulabilities.append(best_manipulability)
    print(f"Best manipulability for rotation {angle} degrees is
{best_manipulability} at point {best_point}")

# Plot manipulability vs. rotation angles
plt.figure(figsize=(10, 6))
plt.plot(angles, manipulabilities, marker='o')
plt.xlabel('Rotation Angle (degrees)')
plt.ylabel('Manipulability')
plt.title('Manipulability vs. Rotation Angle')
plt.grid(True)
plt.show()

# Simulate fiber placement at the best manipulability point
def simulate_fiber_placement_at_best_pose(robot, best_pose):
    if best_pose:
        robot.MoveJ(best_pose)

best_angle_index = np.argmax(manipulabilities)
best_angle = angles[best_angle_index]
F_best_rotated = rotate_path_with_stewart_platform(F, best_angle, best_angle,
best_angle, sphere_center)
best_manipulability, best_pose, best_point =
check_path_manipulability(F_best_rotated, path_index)

simulate_fiber_placement_at_best_pose(robot, best_pose)
```

Appendix E

Simulation of FANUC M20-ia and the Parallel robot

```
from robodk import robolink, robomath
import csv
import numpy as np
import math
import time

# Connect to RoboDK
RDK = robolink.Robolink()
robot = RDK.Item('Fanuc M-20iA')
par = RDK.Item('PI H-840-D2A')

if robot.Valid():
    print('Item selected: ' + robot.Name())
    print('Item position: ' + repr(robot.Pose()))

target_ref = robot.Pose()
pos_ref = target_ref.Pos()
joints = robot.Joints()

robot.MoveJ(target_ref)
robot.setPoseFrame(robot.PoseFrame())
robot.setPoseTool(robot.PoseTool())

# Load paths
base_path = "C:/Users/ASUS/Desktop/Rotary Python/p and s/"
path_points_dict = {}
adjusted_path_points_dict = {}

for i in range(1, 17):
    file_path = f"{base_path}Path{i}.csv"
    data_tuples = []

    with open(file_path, 'r') as csvfile:
        csvreader = csv.reader(csvfile)
        for row in csvreader:
            tuple_row = list(map(float, row))
            data_tuples.append(tuple_row)

    tx, ty, tz, alpha, beta, gamma = 1045, 0, -45, 90, 0, 90
    alpha_rad, beta_rad, gamma_rad = np.radians([alpha, beta, gamma])
```

```
    Rx = np.array([[1, 0, 0],
                   [0, np.cos(alpha_rad), -np.sin(alpha_rad)],
                   [0, np.sin(alpha_rad), np.cos(alpha_rad)]])
    Ry = np.array([[np.cos(beta_rad), 0, np.sin(beta_rad)],
                   [0, 1, 0],
                   [-np.sin(beta_rad), 0, np.cos(beta_rad)]])
    Rz = np.array([[np.cos(gamma_rad), -np.sin(gamma_rad), 0],
                   [np.sin(gamma_rad), np.cos(gamma_rad), 0],
                   [0, 0, 1]])
    R = Rz @ Ry @ Rx
    T = np.eye(4)
    T[:3, :3] = R
    T[:3, 3] = [tx, ty, tz]

    transformed_points = []
    adjusted_transformed_points = []

    for value in data_tuples:
        point = np.array([value[0], value[1], value[2], 1])
        vector = np.array([value[3], value[4], value[5], 0])
        transformed_point = T @ point
        transformed_vector = (R @ vector[:3]) * 180 / np.pi
                  transformed_Tuple    =    np.concatenate((transformed_point[:3],
transformed_vector[:3]))
        transformed_Tuple_rounded = np.round(transformed_Tuple)
        transformed_points.append(list(transformed_Tuple_rounded[:6]))

        # Adjust Z value
        adjusted_point = list(transformed_Tuple_rounded[:3])
        adjusted_point[2] -= 525
                          adjusted_transformed_points.append(adjusted_point     +
list(transformed_Tuple_rounded[3:]))

    path_points_dict[f"TPath{i}"] = transformed_points
    adjusted_path_points_dict[f"TPath{i}"] = adjusted_transformed_points

    path_name = f"TPath{i}"
                      path_item          =          RDK.AddPath(transformed_points,
projection_type=robolink.PROJECTION_ALONG_NORMAL_RECALC)
    path_item.setName(path_name)
    print(f"Path created: {path_name}")
time.sleep(5)

# Hide the normal paths before proceeding
items = RDK.ItemList()
```

```
for item in items:
    if item.Type() == robolink.ITEM_TYPE_OBJECT and 'TPath' in item.Name():
        item.Delete()

# 2. Read the final paths from CSV
final_paths_file = "C:/Users/ASUS/Desktop/Rotary Python/p and s/final_paths.csv"
best_results_file = "C:/Users/ASUS/Desktop/Rotary Python/p and s/best_results.csv"

path_points_dict = {}
best_results = {}

# Load the final paths
with open(final_paths_file, 'r') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        path_key = row['Path']
           point  =  list(map(float,  [row['X'],  row['Y'],  row['Z'],  row['Rx'],
row['Ry'], row['Rz']]))

        if path_key not in path_points_dict:
            path_points_dict[path_key] = []
        path_points_dict[path_key].append(point)

# Load the best results to get the Z angles
with open(best_results_file, 'r') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        path_key = row['Path']
        z_angle = float(row['Best Z Angle'])
        if z_angle > 180:
            z_angle = z_angle - 180
           z_angle_radians = z_angle * (np.pi / 180)  # Convert Z angle to radians
                 best_results[path_key]  =  {'z_angle_radians':  z_angle_radians,
'reverse': True}
        else:
           z_angle_radians = z_angle * (np.pi / 180)  # Convert Z angle to radians
                 best_results[path_key]  =  {'z_angle_radians':  z_angle_radians,
'reverse': False}

# 3. Bring paths into correct position and show them
RDK = robolink.Robolink()
robot = RDK.Item('Fanuc M-20iA')

# Load the part (Product8withoutroller)
part = RDK.Item('Product8withoutroller')
```

```
if not part.Valid():
    print("Product8withoutroller not found. Please ensure it is loaded in RoboDK.")
else:
    part.setVisible(True)
    print('Product8withoutroller loaded and visible.')

# Clear previous paths
items = RDK.ItemList()
for item in items:
    if item.Type() == robolink.ITEM_TYPE_OBJECT and 'TPath' in item.Name():
        item.Delete()

if robot.Valid():
    print('Robot selected: ' + robot.Name())

# Display the original paths
for path_key, points in path_points_dict.items():
    path_item = RDK.AddPath(points,
projection_type=robolink.PROJECTION_ALONG_NORMAL_RECALC)
    path_item.setName(f"Original_{path_key}")
    path_item.setVisible(True)
    print(f"Original Path {path_key} displayed.")

# Get the orientation (Rx, Ry, Rz) from the part
part_pose = part.Pose()
part_orientation = robomath.Pose_2_TxyzRxyz(part_pose)[3:]  # Extracting the
orientation Rx, Ry, Rz

# Z adjustment value
z_adjustment = -530

# Adjust the paths and apply orientation adjustment
for path_key, points in path_points_dict.items():
    num_points = len(points)
    angle_step = 0.8 * (np.pi / 180)  # 5 degrees in radians
    start_rx_offset = (num_points - 1) * angle_step / 2

    adjusted_points = []
    z_angle_radians = best_results[path_key]['z_angle_radians']
    reverse = best_results[path_key]['reverse']

    for i, point in enumerate(points):
        # Apply Z adjustment
        x, y, z = point[:3]
        z += z_adjustment
```

```
            # Apply the orientation adjustment for Rx in radians
            rx_offset = start_rx_offset + i * angle_step
            rx = part_orientation[0] + rx_offset

            # Adjust Rz with the Z angle in radians
            ry = part_orientation[1]
            rz = part_orientation[2] + z_angle_radians

            if reverse:
                # Reverse the orientation if Z angle was over 180 degrees
                rz = -rz

            adjusted_point = [x, y, z, rx, ry, rz]
            adjusted_points.append(adjusted_point)

        path_points_dict[path_key] = adjusted_points

# 4. Move the robot along the adjusted paths
def move_robot_on_path(robot, points):
    for point in points:
        x, y, z, rx, ry, rz = point
        target_pose = robomath.transl(x, y, z) * robomath.rotx(np.pi-rx) *
robomath.roty(ry) * robomath.rotz(rz)
        robot.MoveJ(target_pose, blocking=True)
        print(f"Robot moved to position {x:.2f}, {y:.2f}, {z:.2f} with
orientation {rx:.2f} rad, {ry:.2f} rad, {rz:.2f} rad")

# Move the robot on each adjusted path
for path_key, points in path_points_dict.items():
    if path_key == "TPath9":
        print(f"Skipping fiber placement on {path_key}")
        continue
    print(f"Moving robot on adjusted path {path_key}")
    move_robot_on_path(robot, points)

print("Robot has moved along all adjusted paths, excluding TPath9.")
```