Simulation and Control Motion Software Development for Micro Manufacturing

by

Abdolreza Bayesteh
BSc, National Technical University of Ukraine, 2009,
MSc, National Technical University of Ukraine, 2011,

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Mechanical Engineering

Supervisory Committee

Simulation and Control Motion Software Development for Micro Manufacturing

by

Abdolreza Bayesteh
BSc, National Technical University of Ukraine, 2009,
MSc, National Technical University of Ukraine, 2011,

Supervisory Committee

Dr. Martin Byung-Guk Jun, Department of Mechanical Engineering
Supervisor

Dr. Zuomin Dong, Department of Mechanical Engineering
Departmental Member

Abstract

Supervisory Committee
Dr. Martin Byung-Guk Jun, Department of Mechanical Engineering
Supervisor
Dr. Zuomin Dong, Department of Mechanical Engineering
Departmental Member

Due to increasing trends of miniaturization, components with microscale features are in high demand. Accordingly, manufacturing and measurement of small components as small as a few microns became new challenges. Micro milling and femtosecond laser machining are the most common in use cutting operations providing high accuracy and productivity. Micro milling has unique features different from traditional milling including high ratio of tool size to feature size, and constant ratio of tool edge radius to tool size [1]. Due to the mentioned differences, low stiffness of the micro mill and the complexity of the cutting mechanism at the macroscale, selection of cutting parameters are difficult [2]. Therefore, process performance in micro milling, which affects surface quality and tool life, depends on the selected cutting parameters. Also, for measuring micro components, the available dimensional control systems in the market are atomic force microscopes (AFMs) and a combination of coordinate measuring machines (CMMs) and vision systems. These are confined to the scopes of nanoscale and macroscale parts, respectively. It is difficult to justify the high cost and large size of these systems for measurement of mesoscale/microscale features and components and dimensional verification of miniature parts with 3D features. Therefore, a new cost-effective way is needed for measuring components and features in these scales. Additionally, lack of advanced CAD/CAM software for micro laser machining providing constant velocity along the tool path, is the main problem in femtosecond laser machining. In this thesis, to address the mentioned challenges, different software packages are presented to improve micro machining productivity, to provide an accurate

and cost effective way of micro scanning and to bring CAD/CAM capability for micro laser machining.

Table of Contents

List of Figures

Acknowledgments

I would like to express my deep and sincere gratitude to my supervisor Professor Martin B.G. Jun for his endless support, understanding, kindness, and great supervision. During my MASc studies in Victoria, I have learned a lot from him not only about reasearch but also about life.

I would also like to thank my friends and colleagues in University of Victoria, especially Salah Erfurjani, Aditya Chandurkar, Amin Cheraghi, Afshin Joshesh, Mohammad Plaschi, Farid Ahmed, Max Rukosuyev, Junghyuk Ko, Shan Luo and Yanqiao Zhang, from whom I learned a lot over the past two years.

Finally, I would like to thank my family Fatemeh, Hossein and Alireza for their love, all-time support, and guidance during my whole life.

Chapter 1: Introduction

Due to the revolution of miniaturized technologies the demand of manufacturing products of dimension as low as a few microns has been increasing every day. Micro-milling is one of the best methods to fabricate miniature parts in a wide range of sectors including biomedical, electronic, and aerospace. Micro milling tools with diameters as small as 50 μm are used to fabricate micro-scale parts in wide varieties of materials such as stainless steel, titanium, brass, aluminum, platinum, ceramics, etc. Micro milling has unique features different from traditional milling including high ratio of tool size to feature size, and constant ratio of tool edge radius to tool size [1]. Due to the mentioned differences, low stiffness of the micro mill and the complexity of the cutting mechanism at the macroscale, selection of cutting parameters are difficult [2]. Therefore, process performance in micro milling, which affects surface quality and tool life, depends on the selected cutting parameters. Vibration due to flexible tool and minimum chip thickness effects is also an issue associated with the micro milling process. At low feed rates, when chip thickness is lower than the required minimum chip thickness, rubbing or ploughing occurs instead of cutting [1]. As a result, increased cutting energy and stresses on the cutting edge lead to decreasing surface quality and tool life.

Increasing product miniaturization and the need for complex three-dimensional (3D) geometry with a relative accuracy of 10-3~10-5 demand appropriate quality controls of the fabricated microscale features and components. The available dimensional control systems in the market are atomic force microscopes (AFMs) and a combination of coordinate measuring machines (CMM) and vision systems. These are confined to the scopes of nanoscale and macroscale parts, respectively. It is difficult to justify the high cost and large size of these systems for measurement of mesoscale/microscale features and components and dimensional verification of miniature parts with 3D features. Therefore, a new cost-effective way is needed for measuring components and features in these scales. A new probing system that consists of a wire probe and an acoustic emission (AE) sensor for touch detection has recently been reported [3]. It was demonstrated that the use of a rotating wire and an AE sensor could be quite promising as a cost-effective and accurate probing system.

Laser based micro-machining is becoming increasingly popular because intense laser radiation can machine virtually any material with micron precision and high repeatability. Precision and advanced material fabrication is achievable when laser energy is deposited on the work piece efficiently (No damage or heat-affected zone to adjacent material) and a sample stage is translated with a constant velocity even for a complex scanning trajectory. Therefore, developing an efficient laser processing computer aided design (CAD)/ computer aided manufacturing (CAM) system is required to intelligently deliver laser energy to micro-machine a part. The CAD software packages are powerful design tools to create design drawings and geometrical modeling of product and component. CAM uses computer technologies to assist in all key phases of manufacturing such as process and production planning, machining final parts and quality control. The precision and repeatability in laser based material processing significantly depend on the capability of CAD/CAM to describe the tool path for various operations involved in complex parts manufacturing.

This thesis focuses on software development for micro manufacturing by considering the aforementioned challenges, resulting in a micro milling simulation software for feed rate adjustment based on chip volume, a CMM software for micro probing, a 2-Dimensional ploughing volume simulation for micro flat end milling and a CAD/CAM Software package for micro laser machining.

Chapter 2: Literature Review

2.1 Optimization for milling

Feed rate adjustment for conventional milling has been researched for more than two decades. The aim of the feed rate optimization is to achieve the best and optimum cutting parameters to be able to reduce the machining time and improve surface quality and tool life. Many studies have been investigated during the last two decades to predict cutting forces, vibration, chatter and etc. to find a reliable method for solving the problem. All of those methods can be divided into the following two major groups:

- On-line optimization
- Off-line optimization

2.1.1 Online optimization

In this method cutting parameters such as feed and spindle speed are adjusted based on feedback from the sensor(s) installed on NC machine tools to maintain a constant load during the machining. The following sensors can be used for on-line monitoring system:

- ***Internal sensors***

Most of CNC machine tools have servo motors or spindle motors which are controlled by current sensors. When torque causes motor disturbance, current is modified by the servo controller to compensate its influence [4]. Therefore, cutting force can be estimated based on servo motor's current which is the cheapest way to monitor the machining process. Cutting force calculation by servo motor current method was used by Altintas [5] and Lee et al. [6] for tool breakage detection. In many CNC machines, spindle load is displayed on the screen, as the result of cutting torque calculation.

- ***External sensors***

Sensors installed on machine tools to monitor cutting force are called external sensors [7]. They are classified into the following groups:

- ***Force sensors***

Dynamometers are the most accurate and reliable force sensors that can be mounted on any machine tool table. Dynamometers use quartz piezoelectric transducers to measure cutting force [8] [9] but they are not cost effective and their use in machining process control is limited [10].

- ***Torque Sensors***

Piezoelectric-based dynamometers are able to measure cutting torques on spindles [9]. Cutting torque also can be calculated using strain gauges [11]. Torque sensor installation on the spindle is difficult because of machine tool space restrictions [4]. To solve this problem, torque sensors are integrated into tool holders [12] to measure torque, axial, and radial forces.

- ***Cutting Force Calculation by Spindle Displacement***

Non-contact displacement measurement (Figure 2-1) is easier than mounting a sensor on the spindle to measure cutting forces or torque. Calculating cutting force from spindle displacement was studied by Kim et al. [13] and Jeong et al. [14] where they measured spindle displacement in turning using three displacement sensors. A high-speed spindle with sensors integrated to monitor spindle conditions such as vibration, run-out, and temperature, is becoming popular in new high speed CNC machines [15]. However this technique has two major issues: thermal influence and spindle stiffness.

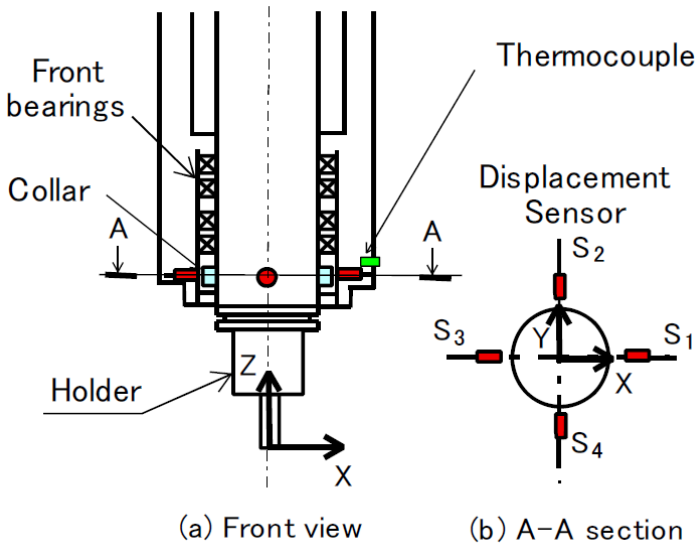


Figure 2-1 : Spindle-integrated displacement sensor [16]

- ***Acoustic emission sensor***

AE sensor is designed to detect stress waves motion caused a local dynamic material displacement and to convert this displacement to an analog signal. AE sensors are typically piezoelectric sensors with elements made of special ceramic elements like lead zirconate titanate (PZT) which generates electric signals.

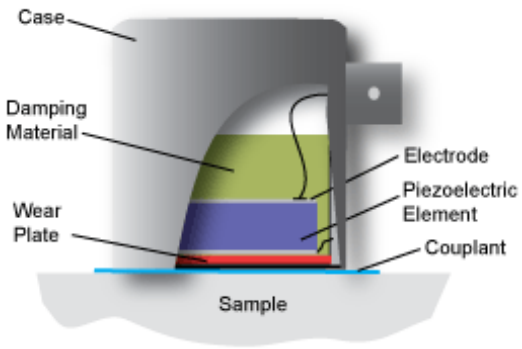


Figure 2-2 : Typical AE sensor mounted on a test object

Many studies have been researched on acoustic emission (AE) in the last three decades to use AE sensor for monitoring and controlling of cutting processes such as

drilling, milling, turning, reaming and grinding. Tanaka et al. suggested a new monitoring system based on AE signal for automatic feedrate adjustment in milling.

2.1.2 Off-line optimization

The aim of this method is to adjust cutting parameters based on predicted cutting forces by using virtual simulation software packages prior of performing the machining operation. Therefore the challenge is how to predict cutting forces, which helps to figure out the dynamic behaviors during machining. The feedrate optimization process in general consists of two stages:

- Feedrate optimization with respect to parameters:
  - Cutting force
  - Material removal Rate (MRR)
  - Chip Thickness
  - Chip volume
- G-code Modification to apply optimized feedrate values.

Feedrate adjustment approaches in milling are used to keep constant only one of the machining constraints including chip thickness, MRR, surface roughness Ra, deflection and resultant cutting force R [17]. Feedrate adjustment based on MRR is commonly used by many researchers. In this method, feedrate is considered to be proportional to either average or instantaneous material removal rate and machining productivity is increased by maximizing the MRR [18].

$$MRR = a \bullet B \bullet c \bullet n \bullet N \tag{2.1}$$

where a is the axial depth of cut, B is the radial width of cut, c is the feed per tooth, n is the spindle speed, and N is the number of flutes on the milling cutter as shown in Figure 2-3.

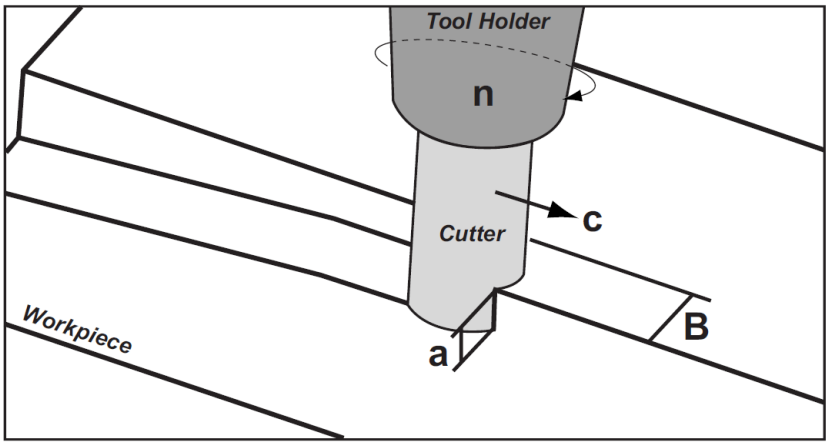


Figure 2-3 : MRR based optimization variables [18]

Many researchers in the last two decades used MRR based optimization. Donggo et al. [19] developed a voxel-based simulator for multi-axis CNC machining. The workpiece was represented as a voxel model and milling simulation was performed by continually subtracting tool swept volumes from the workpiece. MRR values were calculated by counting the number of removed voxels and for feedrate adjustment, the MRR model was used. Wang [20] was one of the first researchers who have developed a feedrate adjustment algorithm based on MRR for milling. His developed solid modeller simulation software was capable of maximizing MRR for 3-5 axis end milling which could minimize chatter and tool breakage. Doruk et al. [21] developed a virtual milling system to predict and improve the performance of three-axis milling operations by using a constraint-based optimization approach to maximize the MRR. Bailey et al. [22] introduced an approach based on chip thickness for feedrate optimization that could reduce the machining time.

Guzel et al. [23] used a methodology for increasing productivity using a force system model. The introduced cutting force model can predict the cutting forces in all X, Y and Z directions. Simulated cutting forces had a good match with the measured forces during

machining and machining time was increased as the result of optimization. Liqiang et al. [24] presented a force prediction model for feedrate scheduling in 5 axis milling by combination of geometric and mechanistic milling models. The developed model was implemented to determine a desired feedrate by considering the maximum allowable cutting force which resulted in significantly decreasing the machining time.

2.1.3 Material removal simulation methods

2.1.3.1 Image space (z-map)

An image space (z-map) approach is one of the popular methods for NC simulation. This method was first used by; Anderson [25] where he developed a 3D histogram simulation software to detect collusion for three axis machining. This method was limited to geometries that do not have undercutting, since there must be a unique z value for a given x, y value. Wang et al. [26] suggested a scan-rendering algorithm using raster graphics for displaying images of solid models. In Wang's approach vectors are drawn for normal pixels to the screen. Intersections between created vectors and toolpath geometries are found using a scan-line algorithm. The buffer is provided for each pixel and for each cutter movement, pixel information is updated. After moving the tool, the workpiece Z-buffer is subtracted from the tool swept volume by performing Boolean operation. Real time simulation is achieved by re-rendering the workpiece after each tool movement.

The image-based approach can simulate the cutting process in 3D but a 3D solid model cannot be obtained since, the simulation data only contains a Z-buffer image. This method also can be used for fast computation and approximate calculation of volume removal rates, but calculation time and memory consumption will be increased by

enhancing accuracy. Among CNC simulators VeriCUT (CGTech, USA) uses an image space approach.

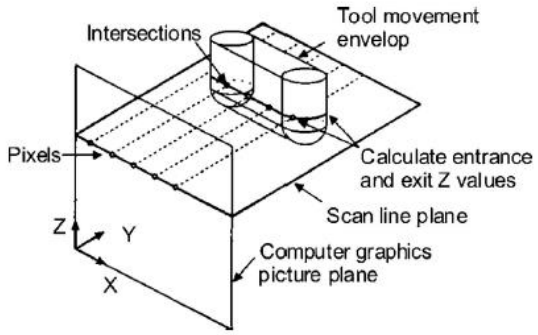


Figure 2-4 : Intersection between z-map image and swept volume [26]

2.1.3.2 Vector-based

Chappel [27] was first to introduce a machining simulation method based on a vector clipping method. In this method, at points on the workpiece, vectors are created normal to the workpiece surfaces and these vectors are extended in outward and inward directions. The milling cutter is presented as a randomly oriented cylinder. By moving the cutter along the toolpath, material removal is simulated by trimming all intersecting vectors. In contrast to the image space approach, this method can provide high resolution verification and it has the ability to handle five-axis machining. The disadvantages of this method are that it cannot generate a solid model for the machined part after the simulation process and it cannot be used for models with wide vector normal angles.

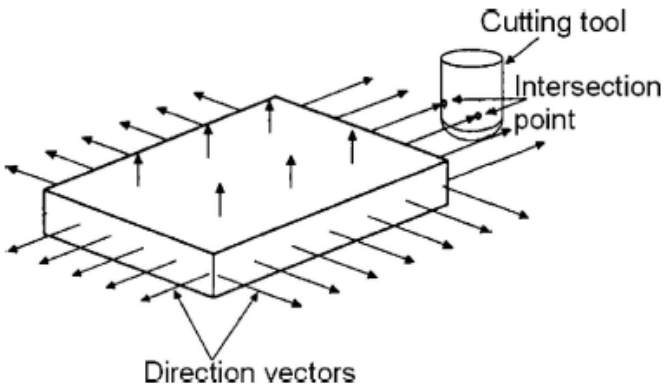


Figure 2-5 : Vectors normal to part surface to simulate material removal process [27]

2.1.3.3 Solid modeling-based

In solid modeling-based method the workpiece is presented using boundary representation (B-rep). By performing Boolean subtraction between the workpiece and the cutter swept volume, the milling process can be simulated. Sungurtekin et al. [28] developed a simulation system for milling which was the first attempt to use CSG modeling technique for material removal progress. Altintas et al. [29] developed a milling simulation software in the part was represented using a CSG solid modeller. Simplicity and high accuracy of the CSG approach was its reason to be popular but this technique has a high computational expense. Despite the fact that, solid modeling allows high resolution verification, the computation time increases quickly as the number of cutter movements increases [30]. The simulation complexity was estimated to be $O(n^4)$, where n is the number of tool movements [31]. The B-rep based milling simulation technique is able to provide very precise machining simulation, but the disadvantages of the method are high computational time, high data storage requirement, and complexity [30]. B-rep model stores lists of vertices, edges, and faces in the memory. Therefore, the amount of

memory required to store new vertices, edges, faces and all the connectivity information increases with the number of tool motions [30].

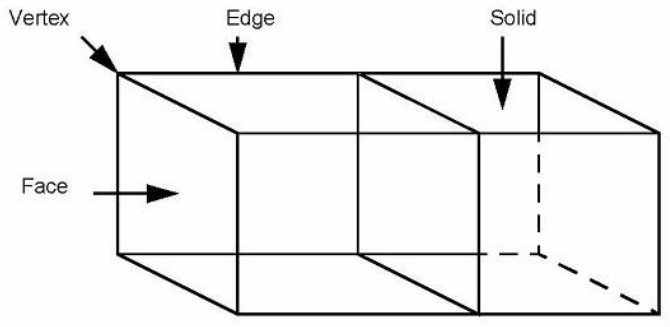


Figure 2-6 : B-rep solid model entities [32]

2.1.3.4 Spatial partitioning

In this approach, the workpiece is converted into simple geometric elements, using spatial partitioning methods such as:

- ❖ Ray casting [33] [34]
- ❖ Dexel [35] [36] [37]

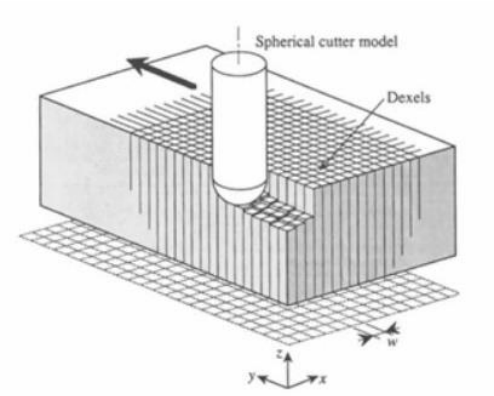


Figure 2-7 : Dexel based machining simulation [38]

- ❖ Graf-tree [39]
- ❖ Voxel [40]
- ❖ Octree [41]

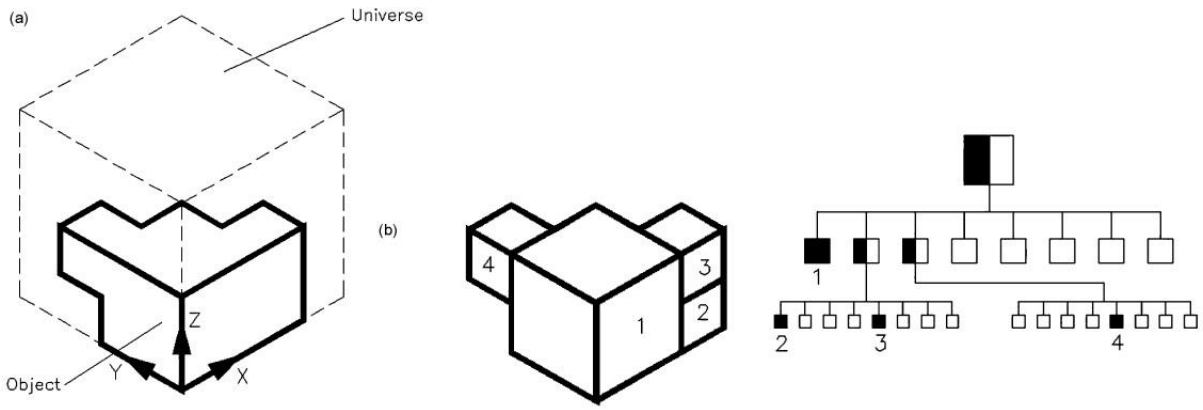


Figure 2-8 : Octree representation of an object (a) object and universe. (b) Octree of the object [41]

These partitioning approaches calculate the tool swept volume based on the defined accuracy that depends on the size and shape of the cells (Dexel, voxel). This method is computationally very efficient but by increasing the accuracy (by decreasing the size of the cells), calculation time and also memory consumption are dramatically increased.

Off-line in comparison to the on-line optimization is more cost effective but requires simulating NC-code based on 3D-model geometry which can be a little time-consuming for a user. On the other hand an on-line optimization approach is not cost effective and it will take time to be implemented in industries.

2.1.4 Existing commercial systems

Implementing feedrate scheduling in free-form surface milling has become popular and it is also used in some commercial CAM software packages such as the following:

- Vericut
- NCSimul
- NCSpeed
- Mastercam

- Powermill
- VegaCNC
- Ezcam

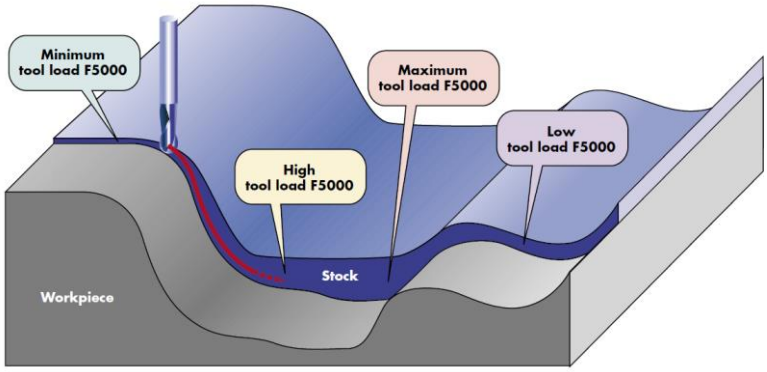


Figure 2-9 : G-code adjustment in NcSpeed [42]

2.1.5 Optimization in Micro milling

Only a few studies have been conducted to optimize the cutting parameters for micro milling. Optimization of cutting parameters in micro milling was studied in [43] by using genetic algorithm and depth of cut, cutting speed and feed rate was considered as the decision variables to improve the tool life. Dimov et al. [44] studied the stability of different strategies for micro-milling, the way they affect the resultant surface finish, and the way to avoid cutting tool breakages. Dynamic and physical simulation of micro milling also was investigated by [2] and [45] where, Jun et al. [2] investigated the dynamics of micro end milling and developed a chip thickness and cutting force model to predict cutting forces and vibration in micro-milling. Finite element modeling of micro-milling based on rigid-plastic deformation was also conducted to predict chip formation [45]. However, although feed rate adjustment for improved productivity is a common practice, issues in feed rate optimization for micro milling have not been well dealt with in the literature.

2.2 Micro probing systems

In general probing technology can be classified into two main methods; contact and non-contact. In the contact-based methods, sensing is done by touching an object surface with sensing elements such as a probe tip. Non-contact methods scan objects using laser-based optical sensors. Contact-based in contrast to the non-contact based probing system has a simplified structure. On the other hand in tactile probing the probing force, generated by the contact between the probe tip and the part surface, is inevitable and fabrication of micro scale probes is a challenge. Non-contact probing systems avoid surface damage during measurement but for increasing the accuracy more optical lenses need to be used, which increases the complexity and cost of the system. Here, various tactile probing technologies are introduced, which are applied for the measurement of the miniaturized components.

2.2.1 Optic fiber based probing system

2.2.1.1 Touch probe using fiber optic displacement sensors

This touch probe system was introduced by Takaaki et al. [46] for providing better sensitivity and repeatability. In this method a ball is mounted to the probe stylus tip with an elastic structure for free movement in three directions (Figure 2-10). When the tip ball touches the workpiece, the ball is elastically moved by the external force. After moving the ball three fibre optic sensors measure the ball's displacement and direction. The contact can be detected by monitoring the change in the sensor's output voltage.

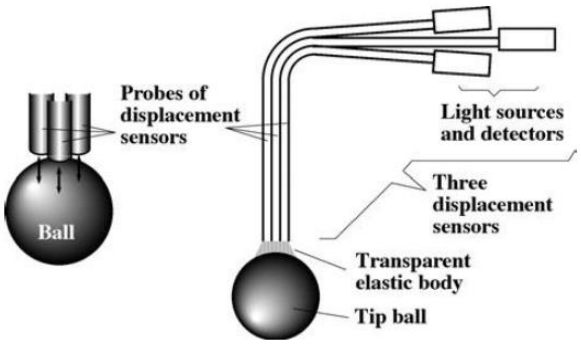


Figure 2-10 : Design of a 3D probe consisting of three displacement sensors [46]

The measurement resolution of the tip ball displacement was reported to be less than 12 nm and the measurement force mentioned was approximately 0.6 mN in the horizontal direction and 4.5 mN in the vertical direction. Low probing force was achieved due to the selected optic elastic adhesive.

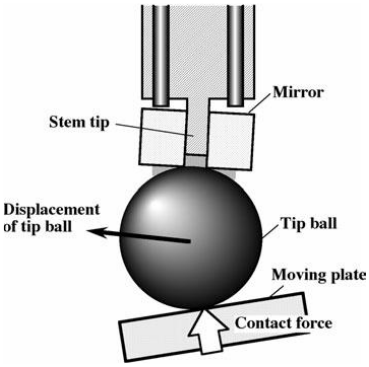


Figure 2-11 : Motion of the tip ball at approach angle [46]

2.2.1.2 Contact-based probe using fiber Bragg grating (FBG) sensor

Ji et al. [47] developed an optical fibre based micro contact probe system with high sensitivity and repeatability. The introduced optical fibre probe had a fused spherical tip which was utilized with a fibre Bragg grating as a strain sensor in the probe stem. When the probe tip touches the surface of the part, a strain will be induced along the probe stem resulting in a Bragg wavelength shift. The contact signal triggers when the wavelength shift signal is produced. Additionally, the authors claimed that fibre grating sensor

integration with the probe provides a high sensitivity probe system. Probing resolution of the developed probe system was analyzed and reported to be about 60 nm in the axial loading.

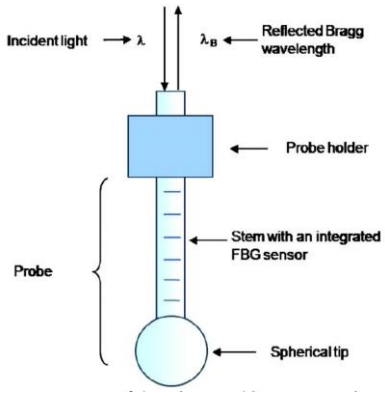


Figure 2-12 Micro-probing system design using FBG sensor [47]

2.2.2 Probing system based on silicon membrane

Butefisch et al. [48] designed a microprobe which consists of sensing elements and a probe. The sensor was based on a silicon boss membrane made of single crystalline silicon and thickness of 20–30 μm. The membrane contains piezo-resistive semiconductor elements which change their resistance dependent on the mechanical strain. A ruby ball with a diameter of 300 μm was mounted on the tip of the stylus. Ball deflection deforms the membrane, resulting in change of the resistances of the Piezo-resistive elements. These Piezo-resistive elements are wired to four Wheatstone-bridges. Based on these four sensor signals, the deflection of the probing sphere can be calculated.

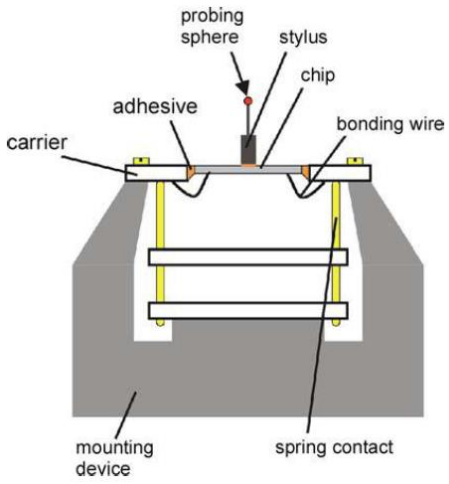


Figure 2-13 : Silicon membrane based probing concept [48]

2.2.3 Probing system using fizeau interferometry

Liebrich et al. [49] introduced a new type of probing system, which had a complicated mechanical setup with high sensitivity, low probing force and high price. The designed probing system consists of a laser interferometer, adjustment unit and probing head. The micro-probe is attached to the hinge providing the probe unit elasticity, as shown in Figure 2-14. The design hinge was fabricated from a thin foil to decrease the probing force. The probing concept is based on a Fizeau interferometer to sense the probe deflection produced by touching of the probe tip and a part surface.

The repeatability of the probing system was reported to be 0.19 μm. The measurement accuracy of this method is not satisfactory because of the probe thermal drift. For a measurement period of four hours the thermal drift was measured to be 0.16 μm. By calibrating the probing system every 30 min, the drift can be reduced.

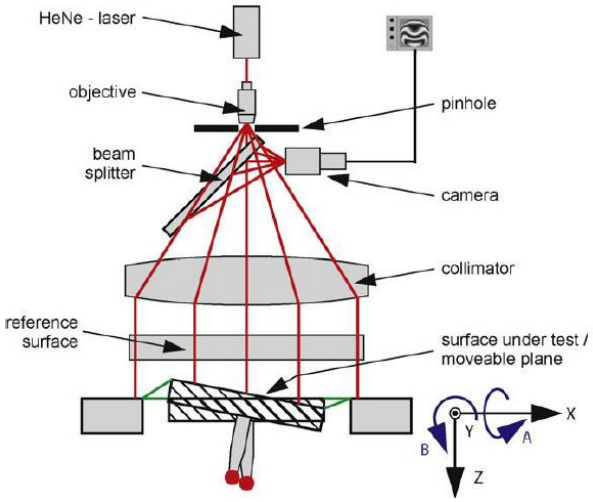


Figure 2-14 : Mechanical setup for probing system using Fizeau interferometry [49]

2.2.4 Piezoelectric-based probing system

The proposed micro-probing system by Hidaka [50] consists of a micro-probe, two sensing elements, a driving and a sensing electrode as shown in Figure 2-15. The stylus is axially vibrated by a bulk-Lead zirconium titanate (PZT) at the resonant frequency excited by sinusoidal voltage. When the probe tip (a sphere attached to the end of the stylus) touches the part surface, vibration reduces due to the change of voltage amplitude. A voltage signal is picked up by the sensing electrode. In general, the contacting force was calculated to be 0.15 mN and the repeatability was reported to be less than 2 μm.

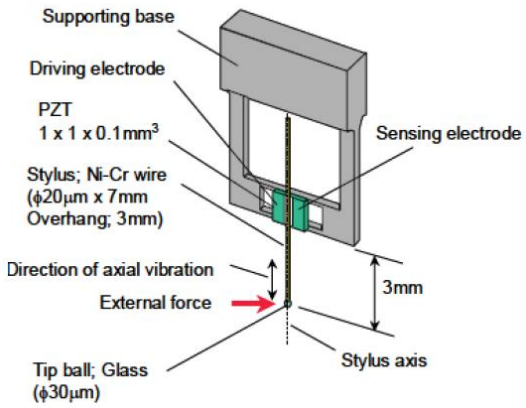


Figure 2-15 : Structure of the piezoelectric-based probing system [50]

2.3 Effect of minimum chip thickness in Micro milling

Micro milling has unique features different from traditional milling including high ratio of tool size to feature size, and constant ratio of tool edge radius to tool size [1]. Due to the mentioned differences, low stiffness of the micro mill and the complexity of the cutting mechanism at the macroscale, selection of cutting parameters are difficult [2]. Therefore, process performance in micro milling, which affects surface quality and tool life, depends on the selected cutting parameters.

Vibration due to tool flexibility and minimum chip thickness effects is also an issue associated with the micro milling process. At low feed rates, when chip thickness is lower than the required minimum chip thickness, rubbing or ploughing occurs instead of cutting [1]. As a result, increased cutting energy and stresses on the cutting edge lead to decreasing surface quality and tool life. In order to avoid ploughing in the micro milling process, the uncut chip thickness must be greater than the minimum chip thickness [51].

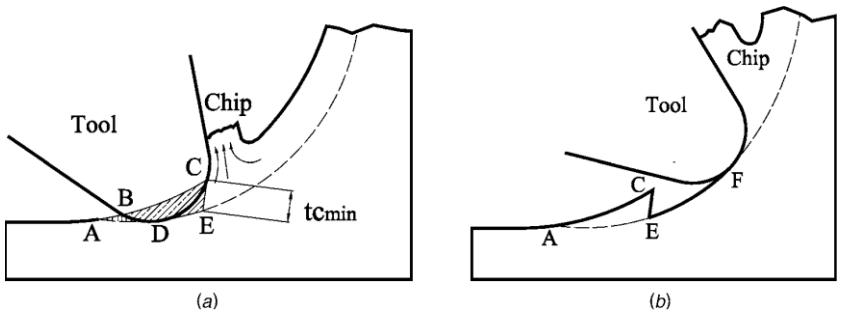


Figure 2-16 : Illustration of the minimum chip thickness effect [52]

Micro deformation during ploughing increases significantly cutting forces [53], burr formation [54], and surface roughness [55] and [56]. Therefore, the minimum chip thickness value must to be considered for selection of appropriate cutting parameters.

Minimum chip thickness depends on many parameters associated with tool geometry and material properties. Liu et al. [56] estimated the minimum chip thickness based on tool cutting edge radius as:

$$t_{\min} = \frac{f}{N \times n} > 0.3 r_e \tag{2.2}$$

where n is spindle speed, N is number of teeth and r_e is tool cutting edge.

Several studies focusing on chip formation mechanism in micro milling have been found. Malekian M. et al. [51] studied the minimum uncut chip thickness and it was determined that the average minimum chip thickness was approximately 0.23 of the edge radius for aluminum. Özel et al. [7] also simulated the micro milling process and estimated minimum chip thickness values for different materials. Kim et al. [54] investigated the chip formation and cutting forces in micro milling processes. The developed chip formation model includes minimum chip thickness and edge-radius effects. The proposed model was experimentally verified by analyzing cutting forces obtained during machining and the relationship between feed per tooth and cutting forces was achieved. Liu et al. [52] developed an analytical model to predict the minimum chip thickness value for the process development and optimization in micro milling. The developed model considers the thermal softening and strain hardening effects on the minimum chip thickness. Additionally, the influence of cutting velocity and tool edge radius on the minimum chip thickness was taken into account. Bissacco et al. [57] presented a theoretical model for cutting force prediction in micro milling by considering the cutting edge radius size effect, the tool run out and the deviation of the chip flow angle from the inclination angle. A parameterization based on the uncut chip thickness to cutting edge radius ratio was used for the cutting force calculation parameters.

However, there is limited work reported on ploughing volume simulation for a given tool path in micro milling based on minimum chip thickness.

2.4 Review of micro laser machining CAM software packages

Laser based micro-machining is becoming increasingly popular because intense laser radiation can machine virtually any material with micron precision and high repeatability. The major advantages of laser materials processing include precision and atomization in micromachining, sharp cutting edge, localization of the heat affected zone and low crack **[58] [59] [60]**. Precision and advanced material fabrication is achievable when laser energy is deposited on the work piece efficiently (No damage or heat-affected zone to adjacent material) and a sample stage is translated with a constant velocity even for a complex scanning trajectory. Therefore, developing an efficient laser processing computer aided design (CAD)/ computer aided manufacturing (CAM) system is required to intelligently deliver laser energy to micro-machine a part. The CAD software packages are powerful design tools to create design drawings and geometrical modeling of product and component. CAM uses computer technologies to assist in all key phases of manufacturing such as process and production planning, machining final parts and quality control. The precision and repeatability in laser based material processing significantly depend on the capability of CAD/CAM to describe the tool path for various operations involved in complex parts manufacturing. Considerable work has been accomplished to develop CAM software and bring automation in micro- fabrication. 2D and 3D CAM tool have been demonstrated to generate tool paths according to a CAD drawing of a microelectromechanical systems (MEMS) device **[61] [62]**. Tool path generation of 2D profiles has also been reported that focuses in laser based contour machining **[63] [64]**.

Efficient tool path generation involves flexibility of importing drawing exchange format (DXF) files, arbitrary 3D tool path (2D, 3D contour, spiral, helix etc.) generation, offset generation of the path elements in a trajectory, maintaining constant feed for all trajectory elements, and 3D visualization to monitor laser machining process. In addition, typically the CAD/CAM software is either computer specific or controller specific.

Chapter 3: Feed Rate Optimization Software for Micro Milling

3.1 Model for Chip Volume Calculation

Feed rate optimization for milling at the macro scale has been used by many researchers to decrease machining time and increase tool life and surface quality. In recent years, the idea as feedrate scheduling became an important feature of CAM software including Mastercam and Powermill. In most of the optimization methods, force predictions are based on MRR, chip volume or chip thickness. Volume calculation is the initial step and there are several methods for volume calculation, including Voxel, Dexel, imaged surface based, and Boolean operation, each with different advantages and disadvantages. Due to the high accuracy calculation of micro-milling, the Boolean operation method was implemented to calculate chip thickness. Boolean operation is not suitable for real time simulation and causes memory leak issues for long time simulations. Therefore, a combination of Voxel and solid Boolean operation was used as shown in Figure 3-1 to improve the calculation time and avoid memory leak problems.

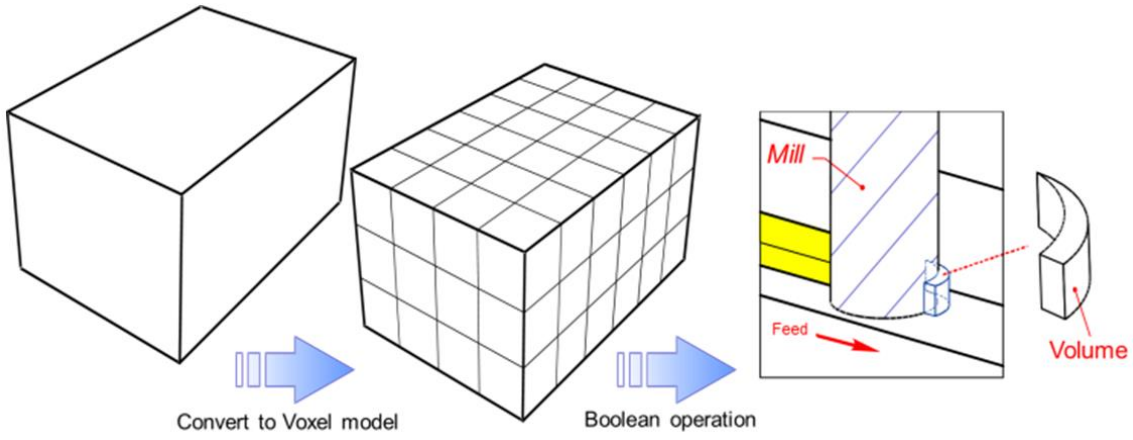


Figure 3-1: Geometrical Chip volume calculation procedure

For chip volume calculation, after importing and parsing the G-code program, each segment (line or arc) of the toolpath was divided in smaller portions based on feed per tooth value (equation 3.1). The feed per tooth can be calculated as,

$$f_t = f / (RPM \times Number\ of\ teeth) \quad (3.1)$$

Where f is the feed rate in mm/sec. Performing Boolean operations for each created segment by feed per tooth (f_t) resulted in having many subsegments. Therefore, the chip load value was multiplied by a fixed scale number to decrease the number of subsegments and to improve the calculation time.

To use Boolean operation, Computational Geometry Algorithms Library (CGAL www.cgal.org), which is an open source C++ library, was used. In order to communicate with the CGAL functions, a C-Sharp wrapper Dll has been developed to call 3D Boolean subtraction, intersection and mesh volume calculator functions and return the results to the main program in C-Sharp. Boolean operations were performed for each created subsegment in 3D by using OpenGL to calculate the intersected volume and to visualize the cutting process. Results of the simulation and the interface of the program are shown in Figure 3-2.

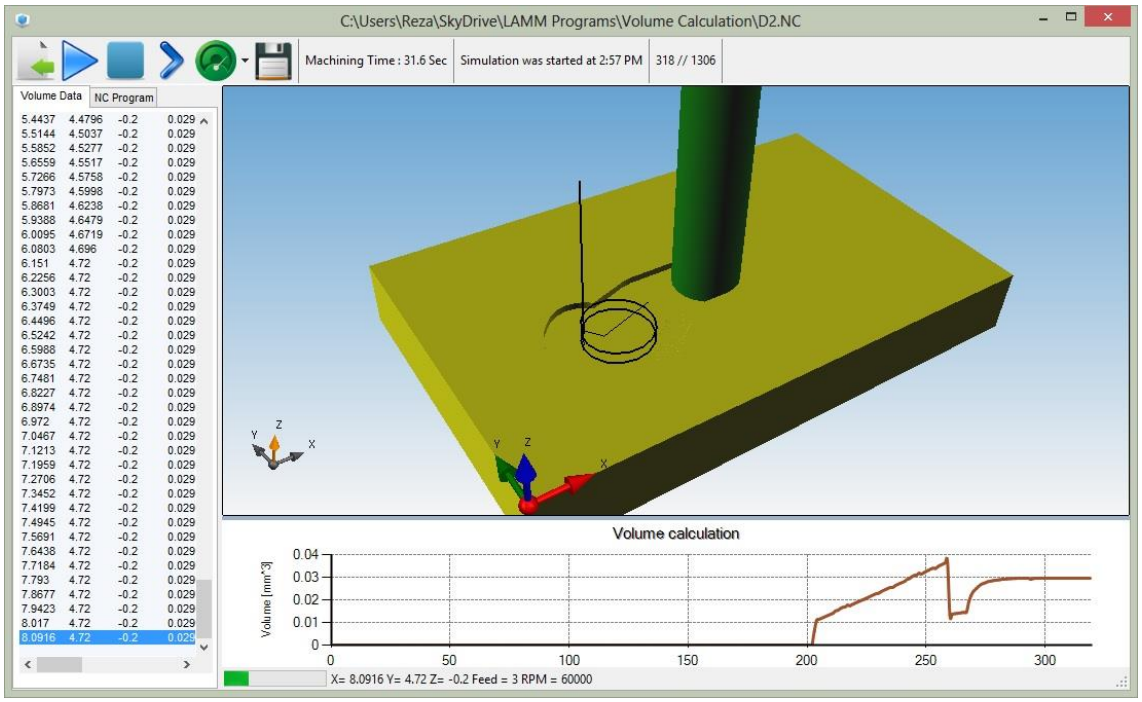


Figure 3-2 : Interface of the micro-milling simulator

3.2 Feed Rate Optimization Algorithm

In this project off-line feed scheduling method has been used. Therefore, the first step is to calculate an optimized feedrate based on chip volume (V_C). After finding all the values of V_C along the tool path for each position, the maximum and minimum value of V_C for the entire tool path can be determined. The values of the maximum feed rate (f_{tmax}) and the minimum feed rate (f_{tmin}) are chosen according to the manufacturer's recommendation or based on the experience. In our case, for machining of Al6061, a minimum feed rate of 1 µm per flute is chosen in order to be well above the minimum chip thickness. The minimum chip thickness for Al6061 is found to be around $0.3r_e$, where r_e is the edge radius [1]. For an edge radius of 2 µm, which is typically the case, the minimum chip thickness (h_c) comes out to be 0.6 µm. It has been found that around $1.5h_c$ is needed to avoid the dynamic effect of the minimum chip thickness [65]. Thus,

the minimum feed rate of 1 μm per flute, which is above the minimum thickness is chosen. For the maximum feed rate, 1.5 μm per flute was selected in order to avoid vibrations due to low stiffness of the tool. The optimum feed for each position can be established as:

$$f_{topt} = \frac{(f_{t\max} - f_{t\min}) \times (V_C - V_{C\min})}{V_{C\max} - V_{C\min}} \tag{3.2}$$

where Vc is the current value of chip volume, V_{Cmin} and V_{Cmax} are the minimum and maximum values of the V_C along the entire tool. Figure 1 shows a schematic representing Equation (3.2) for determining the f_{topt}. As shown, the value of the optimum feed rate proportionally increases as the magnitude of chip volume decreases.

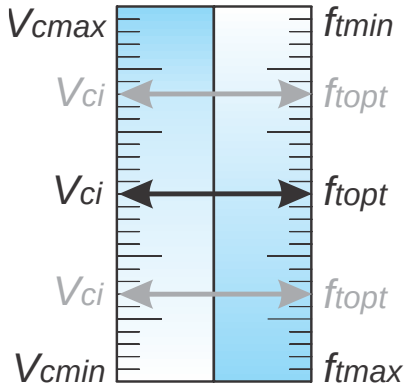


Figure 3-3 : Feed rate optimization scheme

In the optimization algorithm, firstly, required input parameters include tool diameter, number of teeth, workpiece width (*W*), workpiece height (*H*), workpiece thickness (*T*), chip thickness calculation scale factor, minimum feed, maximum feed. The G-code tolerance and G-code program are set by user. The Voxel model of the workpiece was created based on the *W, H* and *T* values. Secondly, after importing the G-code, all of the segments were read and saved in the memory. The Value of f_t based on current feed and spindle speed of each segment was calculated and multiplied by the scale factor. The gain

value from the previous step is used to split the segment into smaller portions. Then, the cutter tool could be moved to the positions of created points and Boolean subtraction and intersection were used to calculate the chip thickness volume. Current position and chip thickness volume were stored in the *DS* list to calculate f_{topt} for each position by using equation 3.2. After the optimum feed rates are determined for the entire tool path, the profile of the feed rates needs to be smoothed. For this purpose moving average method was used to smooth the values of new feed rates. This is because feed rates constantly vary and sudden change in feed rates can cause jerk in the stage motion. Finally, new G-code based on consideration of the input tolerance was generated. The flowchart of the algorithm with more details is presented in Figure 3-4.

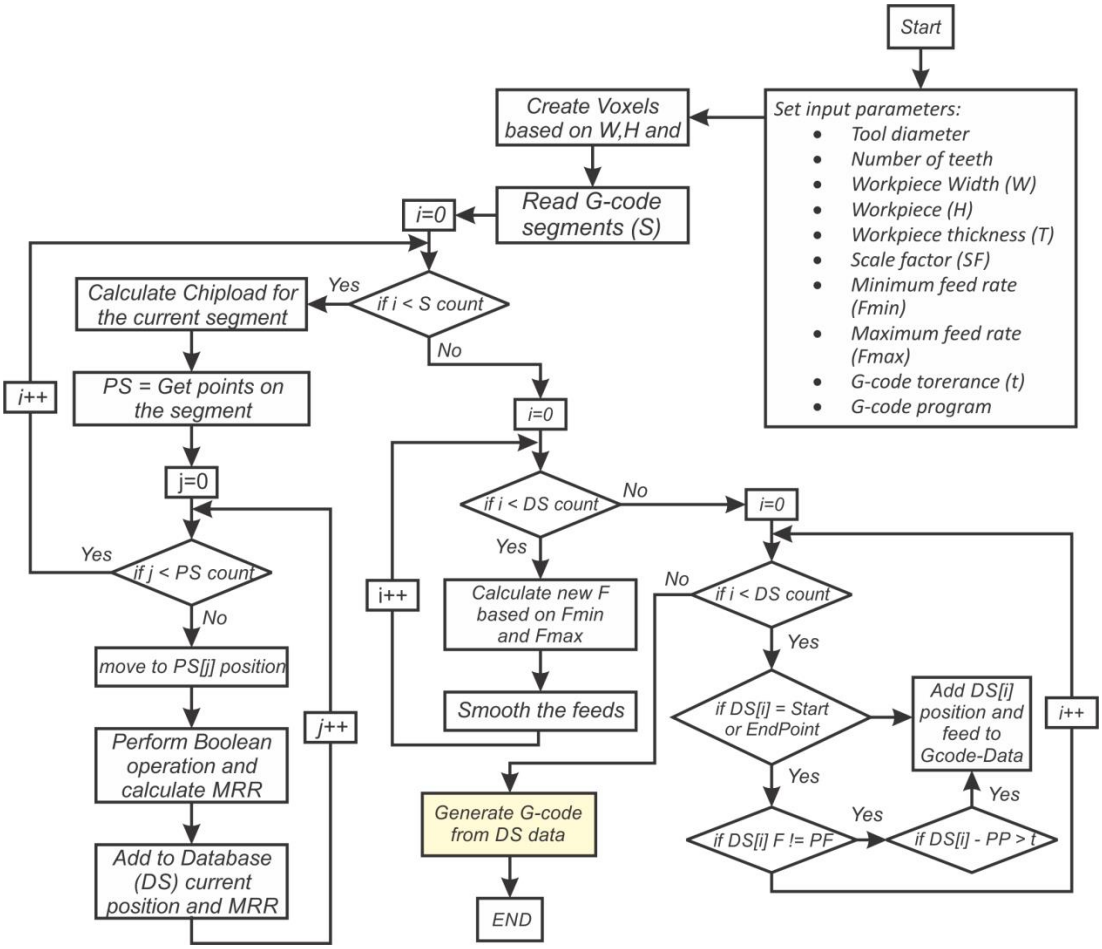


Figure 3-4 : Feed rate optimization flowchart

3.3 Experimental Setup

For feed rate optimization experiments, machining of a pocket shown in Figure 3-5 was performed. Two different tools are considered: 4-fluted 2 mm flat end mill and two-fluted 0.794 mm micro end mill. The nominal feed rate of 1 μm/tooth and 30,000 rpm for tool diameter of 2 mm and the nominal feed rate of 1 μm/tooth and 60,000 rpm for tool diameter of 0.794 mm were used before optimization. A higher rotational speed for the 0.794 mm diameter tool was used due to the decrease in surface speed on the cutting edge. For optimization, 1 μm/tooth (2 mm/sec) and 1.5 μm/tooth (3 mm/sec) for both tools were selected as f_{tmin} and f_{tmax}, respectively.

For the 2 mm diameter tool, the stiffness of the tool may be considered to be close to the tools at the conventional scale. For the 0.794 mm tool, due to the decrease of the diameter, low stiffness of the tool can lead to more vibration even though the pocket size is proportionally decreased. For the two tools, two numerical control (NC) code programs were created by MasterCAM X6 using the same tool path as shown in Figure 3-6. Note that the tool paths are almost the same except that the tool path for the smaller tool has fewer segments. The dots on the tool path represent the start and the end of each segment.

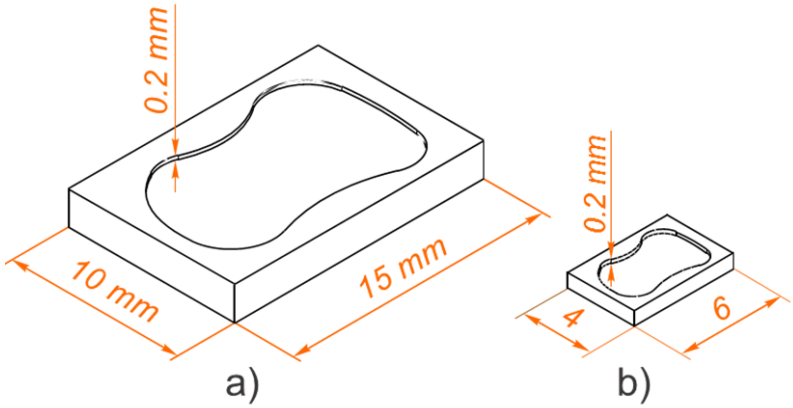


Figure 3-5 : Pocket drawings for (a) 2 mm diameter and (b) 0.794 mm diameter tool

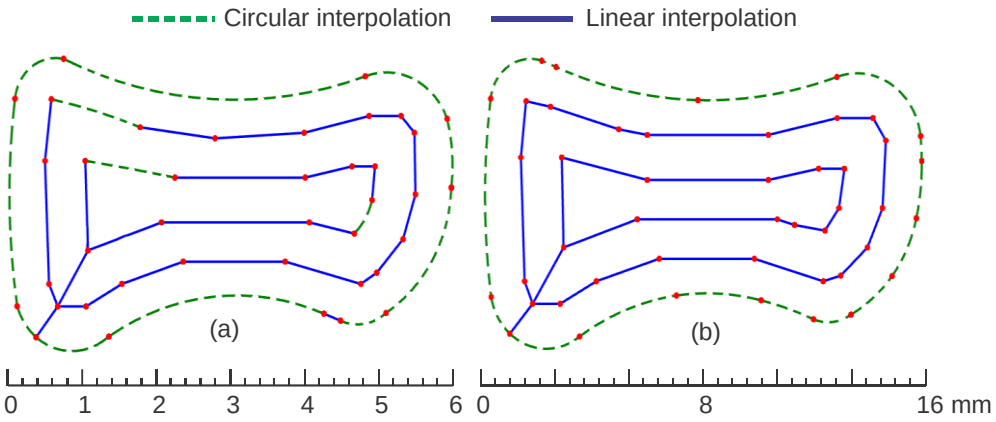


Figure 3-6 : Generated tool paths by Mastercam X6

a) Tool path for tool diameter of 2mm, b) tool path for smaller tool

The machine used in this project was an ALIO vertical micro-milling machine which has a spindle speed range from 10,000 to 80,000 rpm. The experimental setup is

presented in Figure 3-7. The instantaneous cutting force was measured during the machining operation using a Kistler table dynamometer (MiniDyn 9256C1). The material of the workpiece was Aluminium 6061. The workpiece was clamped on the dynamometer on the feed table of the machining center. Machining was carried out under dry conditions. A data acquisition (DAQ) board from National Instruments (NI-P/N-USB-6251 BNC) was used to acquire the measured forces from the dynamometer. A software program using C# was developed to receive data from the DAQ board as well as the coordinates of tool from the controller during the machining operation. A sampling rate of 100 kHz was used to process the measured force data, and the tool position coordinates were acquired at 100 Hz, which is the maximum capacity of the controller.

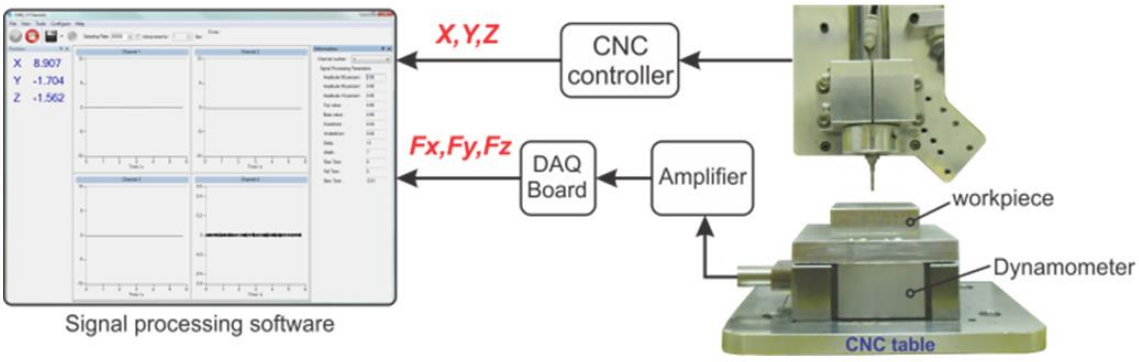


Figure 3-7 : Experimental setup of micro milling operations

The recorded data from the dynamometer and the controller consisted of seven parameters: cutting forces (F_X, F_Y, and F_z) in the X, Y, Z directions, respectively, as well as the current positions (X, Y, Z) with respect to the micro-milling machine and actual feed rate (f) of the stages. The sampling rate of 100 Hz for the tool position is too low for force measurement. Thus, 1000 force readings are received while the micro milling tool position is recorded only once in that interval. Therefore, in order to plot the cutting forces in 3D, the maximum value of the measured 1000 data points of cutting forces was

taken for each position. The heights of 3D plots of cutting forces represent the magnitude of cutting forces for each position. The resultant force is used as the measured cutting force, which can be found from Equation (3.3):

$$R = \sqrt{F_x^{\,2} + F_y^{\,2} + F_z^{\,2}} \tag{3.3}$$

The pockets were first machined at the nominal feed rate 1 μm/tooth (2 mm/sec) and the cutting forces were measured. As the chip volume model of the entire toolpath was simulated, measured cutting forces were compared with chip volume simulation models. The optimized feed rates were calculated and the resulting NC codes were prepared based on the optimization algorithm described in Figure 3-4. Two different optimization algorithms were used for the smaller pocket, which resulted in two optimized G-code programs. Then, experiments were conducted again using the new NC programs with optimum feed rates and the cutting forces were measured. Finally, the chip simulation models were compared to the measured cutting forces to evaluate the optimization method.

3.4 Experimental Results

Measured resultant cutting forces and chip volume data obtained during machining of the pocket using a four-fluted 2.0 mm diameter tool are shown in Figure 3-8 and Figure 3-9 respectively. As shown in Figure 3-8, forces vary slightly above 8 N to as low as 1 N. Figure 3-9 shows that the simulated chip volume profile looks very similar to the measured resultant forces. This indicates that the use of chip volume to obtain optimum feed rates can lead to improve productivity. The optimum feed rates calculated based on chip volume within the range are shown in Figure 3-10.

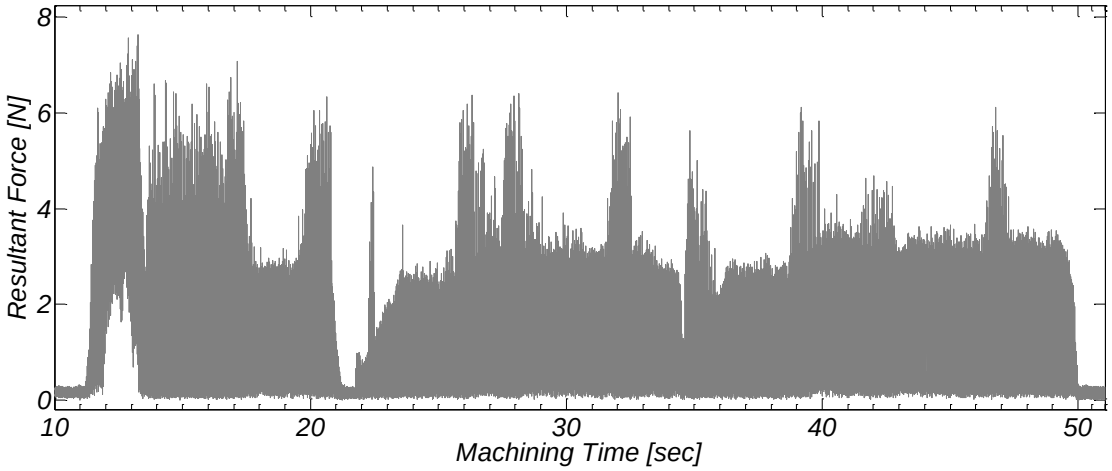


Figure 3-8 : Resultant forces with mill diameter of 2 mm

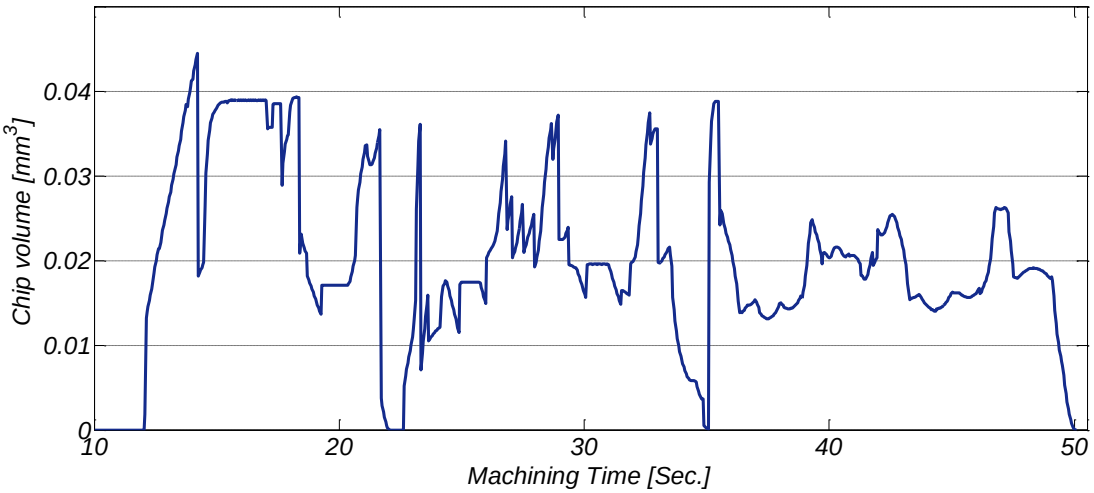


Figure 3-9 : Chip volume simulation with mill diameter of 2 mm

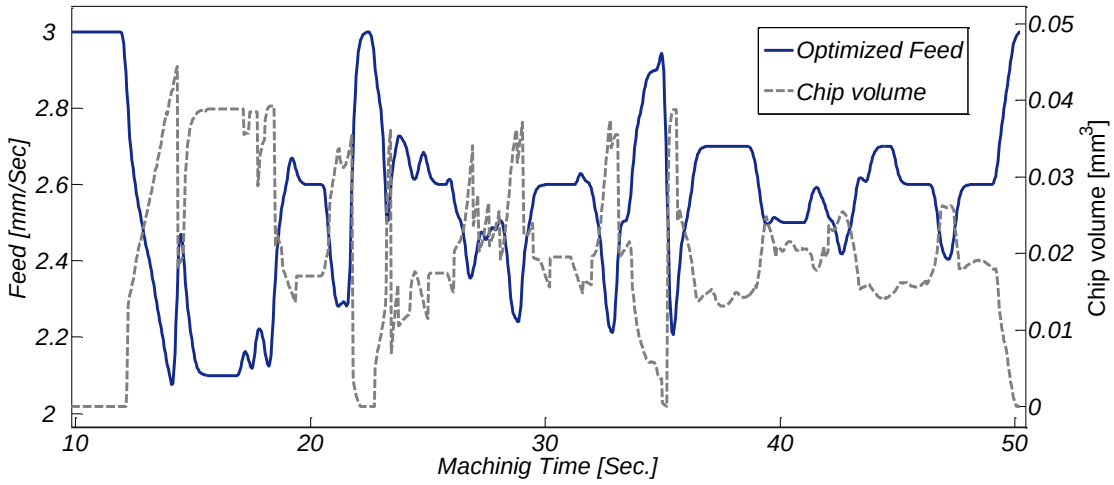


Figure 3-10 : Chip volume during the pocket machining and optimized feed rates for mill diameter of 2 mm

The resultant forces before and after optimization with mill diameter of 2 mm of the pocket are shown in Figure 3-11. It shows that the average resultant forces remained the same although they look more distributed, and the machining time was reduced 16% as a result of the optimization. This indicates that productivity can be increased. Three dimensional profiles of the measured resultant forces along the tool path before and after optimization are shown in Figure 3-12, where the height represents the magnitude of the forces. It shows the locations where the resultant forces are high and low in magnitude. As can be seen, before optimization (Figure 3-12a), machining of the inner most loop is associated with high forces, this is likely because this loop is the first loop that the tool machines and thus full immersion cutting is involved with this loop. After optimization, the force magnitudes are seen to be more or less uniform over all tool locations as shown in Figure 3-12b. Note that in Figure 3-10, feed rates vary quite frequently because the simulated chip volume changes frequently along the tool path. These frequent changes of feed rates resulted in an increase in the number of segments, i.e., the number of G-code blocks. The increase in the number of segments can be observed from the increase in the

number of dots in Figure 3-12b compared to Figure 3-12a. Nonetheless, although the number of segments increased, feed rate adjustment successfully improved the machining performance.

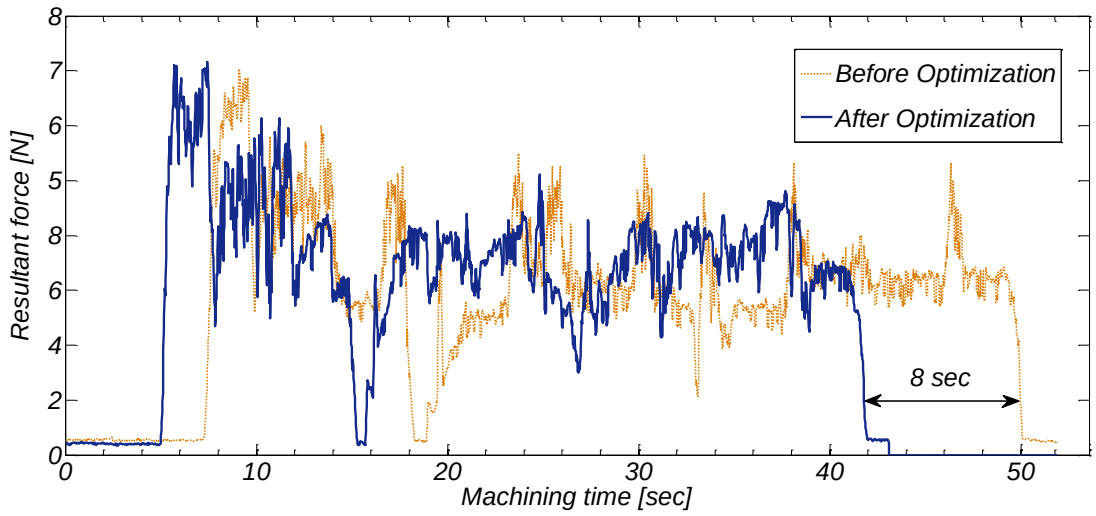


Figure 3-11 : Resultant forces before and after optimization with mill diameter of 2 mm

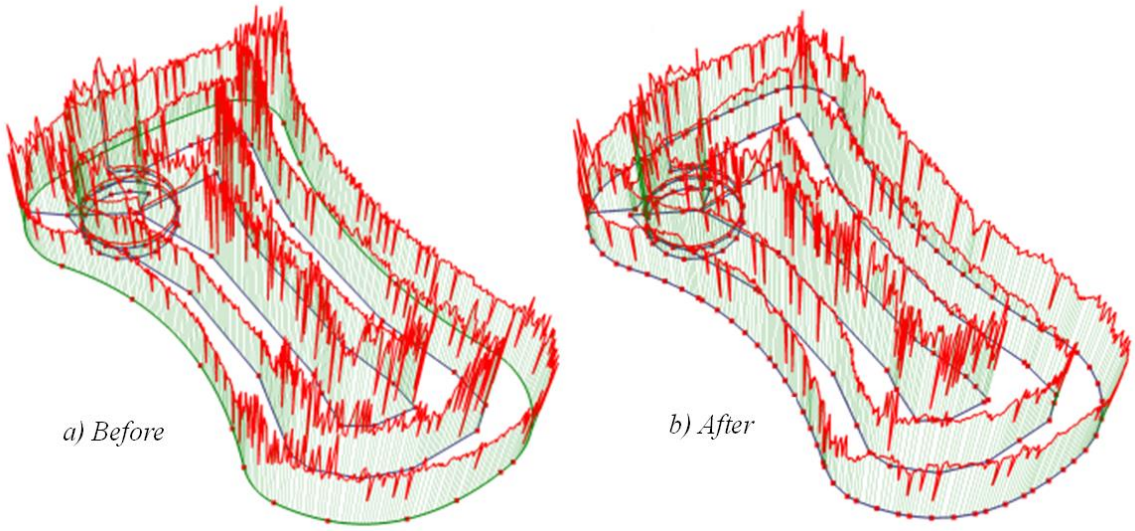


Figure 3-12 : 3D plot of resultant forces before and after optimization with mill diameter of 2 mm.

To evaluate the effect of different ranges of feed rates for optimization, f_{tmin} with a feed value of 0.5 µm/tooth (1 mm/sec) was selected, while f_{tmax} remained at 1.5 µm/tooth (3 mm/sec). The results of the experiment depicted in Figure 3-13Figure 3-14 show that

both the resultant forces and the machining time are increased by using the new range with lower feed rate. The increase in machining time is expected because a lower feed rate was used for the minimum feed (f_{tmin}). However, forces did not decrease although a lower feed rate was used, likely due to the size effect and increased ploughing associated with the low feed rate. Note that the feed rate of 0.5 µm/tooth is close to the minimum chip thickness. This shows that selection of a proper feed rate range for optimization is important for increased productivity as well as improved cutting process.

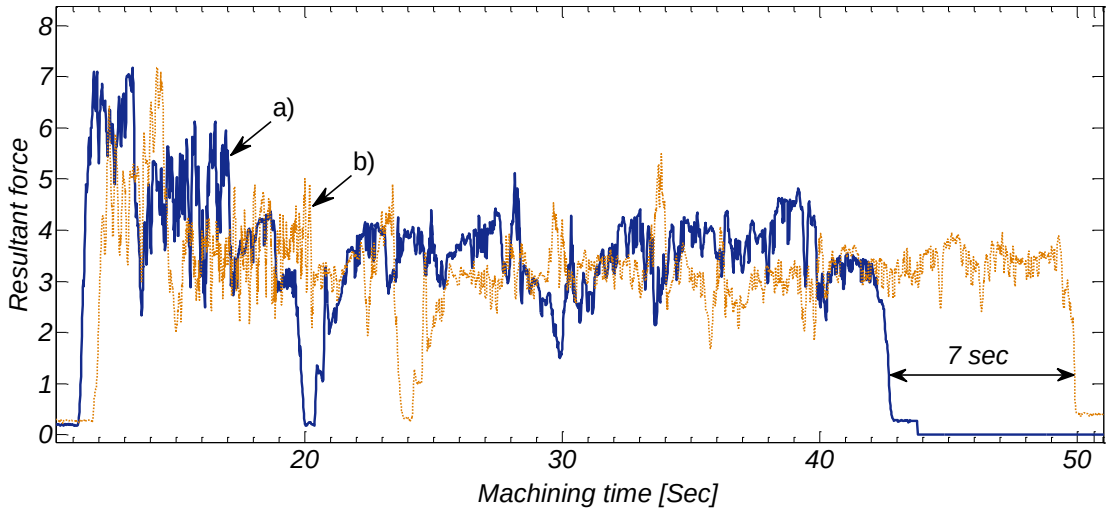


Figure 3-13 : Resultant forces after optimization with mill diameter of 2 mm using feed rate ranges of (a) 1 -1.5 µm/tooth and (b) 0.5-1.5 µm/tooth

The measured cutting forces and chip volume data during machining of the pocket using a two-fluted 0.794 mm diameter tool are represented in Figure 3-14 and Figure 3-15, respectively. As shown in Figure 3-14, forces vary from slightly above 5N to as low as 1N. Notice that the simulated chip volume profile seems to resemble that of the measured forces although chip volume fluctuates more excessively than the measured forces.

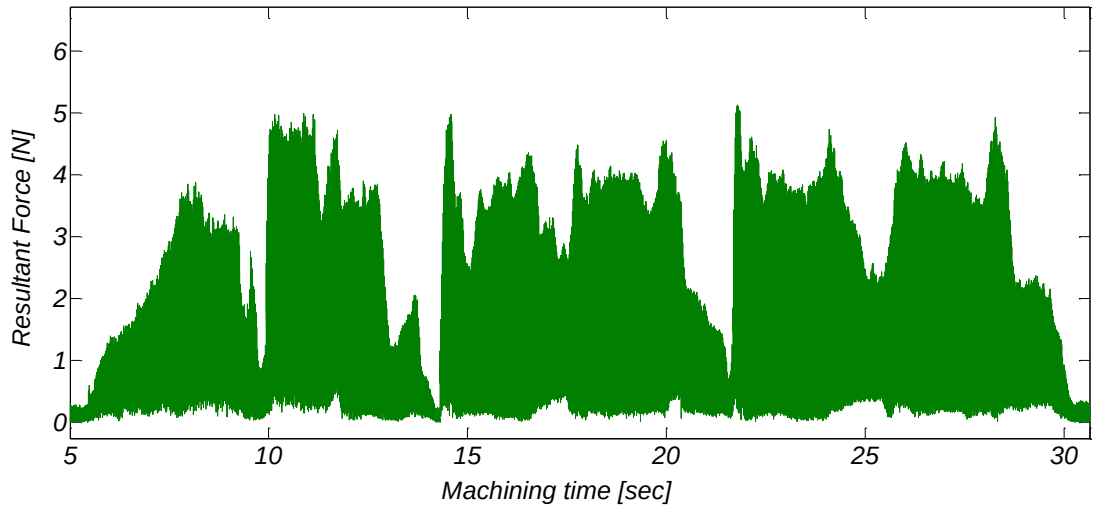


Figure 3-14 : Resultant forces with mill diameter of 0.794 mm

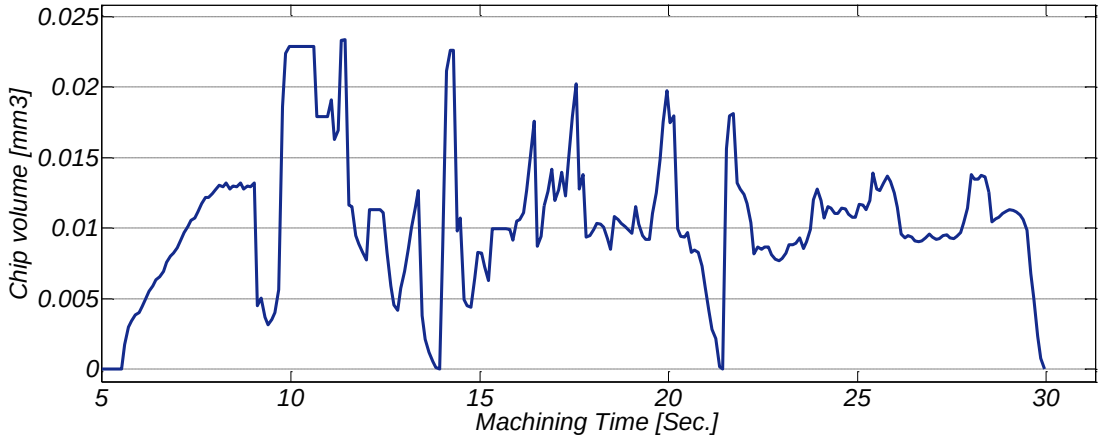


Figure 3-15 : Chip volume with mill diameter of 0.794 mm

The optimized feed rates using the simulated chip volume data during machining of the pocket with the 0.794 diameter micro end mill are shown in Figure 3-16. The comparison of resultant forces before and after optimization is given in Figure 3-17. The three dimensional profiles of the measured resultant forces along the tool path before and after optimization are shown in Figure 3-17.

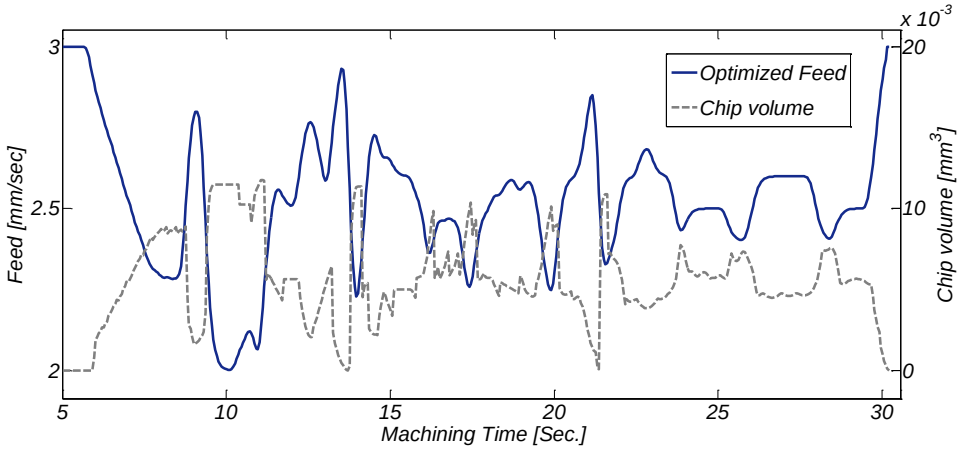


Figure 3-16 : Chip volume during the pocket machining and optimized feed rates for mill diameter of 0.794 mm

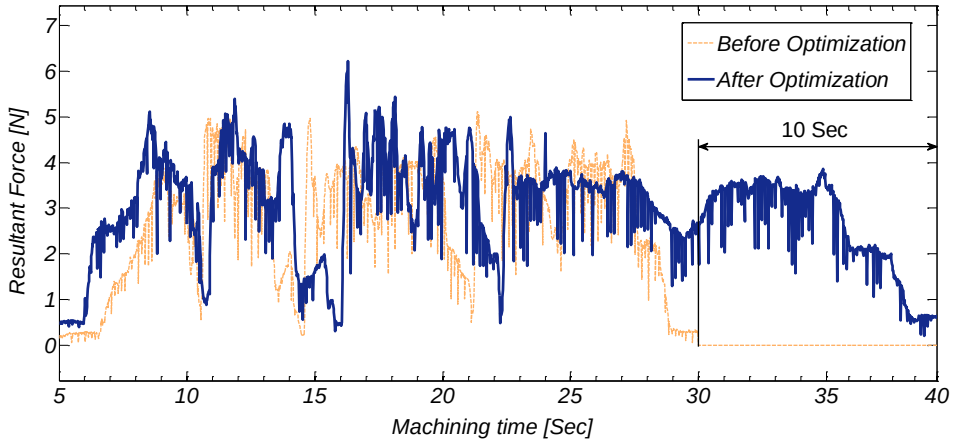


Figure 3-17 : Resultant forces during machining before and after optimization with mill diameter of 0.794 mm

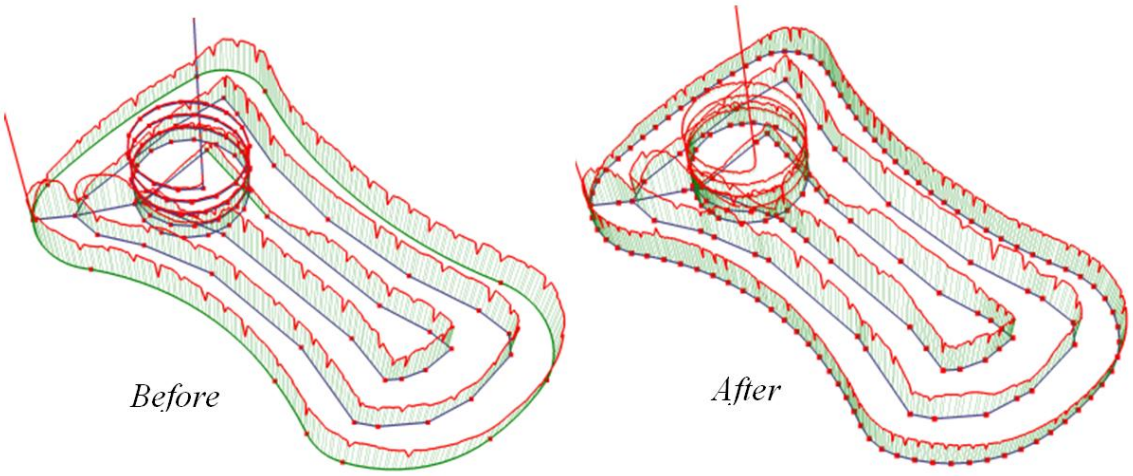


Figure 3-18 : 3D plot of Resultant forces before and after optimization with mill diameter of 0.794 mm

As shown in Figure 3-17, the average resultant force increased 7.5% after optimization. Also, the machining time was increased by 10 seconds. It is interesting that this occurs even though the optimized feed rates specified in the NC program are always higher than the nominal feed rate before optimization. It turned out that the optimized feed rate was actually lower than the specified feed rates in Figure 3-16. The actual feed rates recorded from the controller during machining are shown in Figure 3-19. It shows that feed rate fluctuates significantly and decreases even below the minimum feed rate (f_{tmin}) used for optimization. It turned out that this is because optimization resulted in more segmentation in the NC program, i.e., more position points added along the tool path. Thus, due to the increase in number of the G-code blocks, the length of the NC program with optimized feed rates is much longer than the program before optimization. For micro-milling, because each segment is too short in length, the controller is not able to accelerate the stage up to the specified feed rate within the given time, and the delay within the controller to process each block in the NC program seemed to have caused disruptions in maintaining the specific feed rate. Due to this decrease in feed rates, longer

machining time resulted. This kind of problem has already been reported for micro-milling in the literature [65], where the authors developed an intelligent segmentation method to increase feed rate and improve stability by using circular and NURBS interpolation for segments. This confirms that for proper feed rate optimization for micro-milling, appropriate tool path segmentation must also be considered.

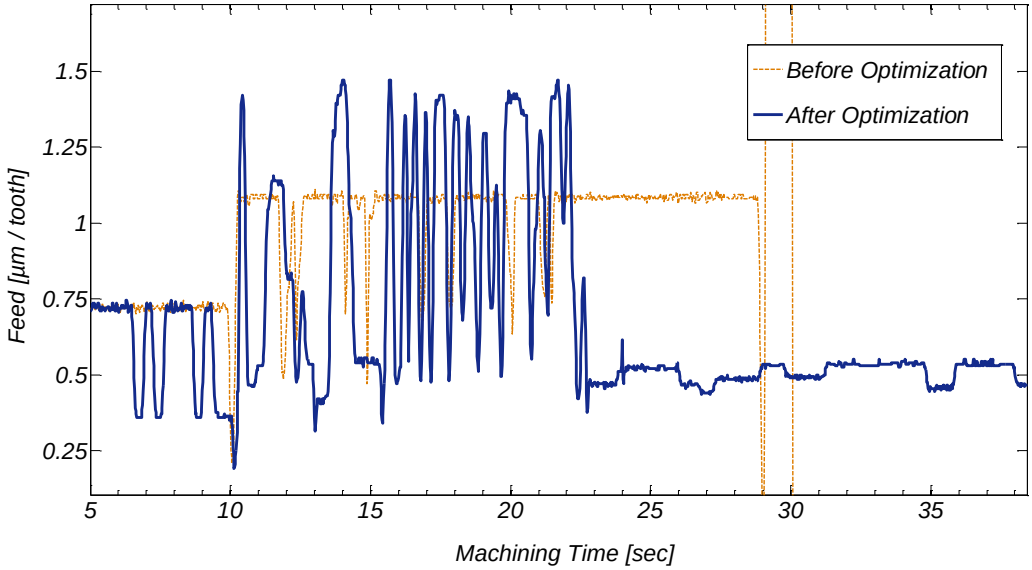


Figure 3-19 : Actual feed rates before and after optimization

To verify the feed rate variation issue with different segment lengths in micro-milling, the following experiments were performed. Different G-code programs were created for a straight path consisting of different numbers of segments with different lengths. A fixed feed rate of 3 mm/sec was set for all the programs, and the actual feed rates were obtained from the controller. Measurement results of the actual feed rates are shown in Figure 3-20, which shows that as segment length is decreased, the actual feed rate is also decreased. The actual feed rate is seen to be linearly proportional to the segment length. If the segment length is increased further, the actual feed rate will match the specified feed rate value. It can be seen in Figure 3-20 that a minimum of 0.4 mm

length segment is needed to reach the specified feed rate with the controller and stage used in this experiment.

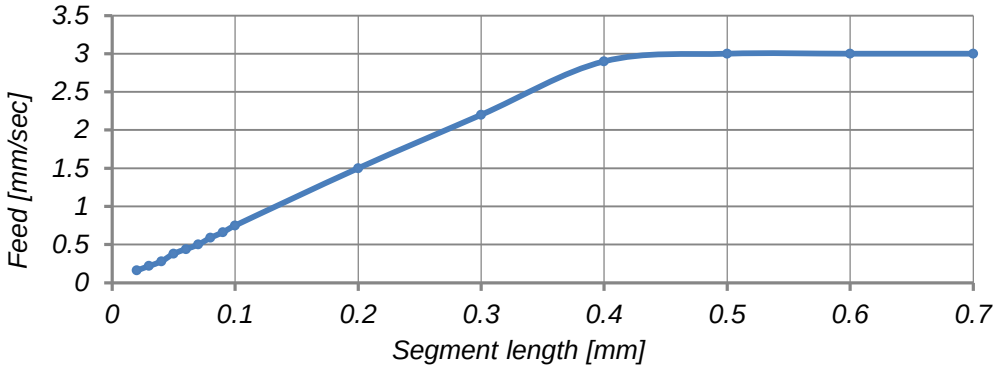


Figure 3-20 : Actual feed rate measurements with different segment lengths

Adding extra points in the program may cause the feed rate to decrease and as a result, the proper feed may not be achieved. Thus, optimization was performed while keeping the number of segments fixed. In this method, instead of breaking the segment in smaller portions and adding new segments with different feeds, average chip volume of each segment is calculated. Based on the average value, equation 3.2 is applied to calculate the optimum feed. Average chip volume values and optimized feed of each segment are depicted in Figure 3-21.

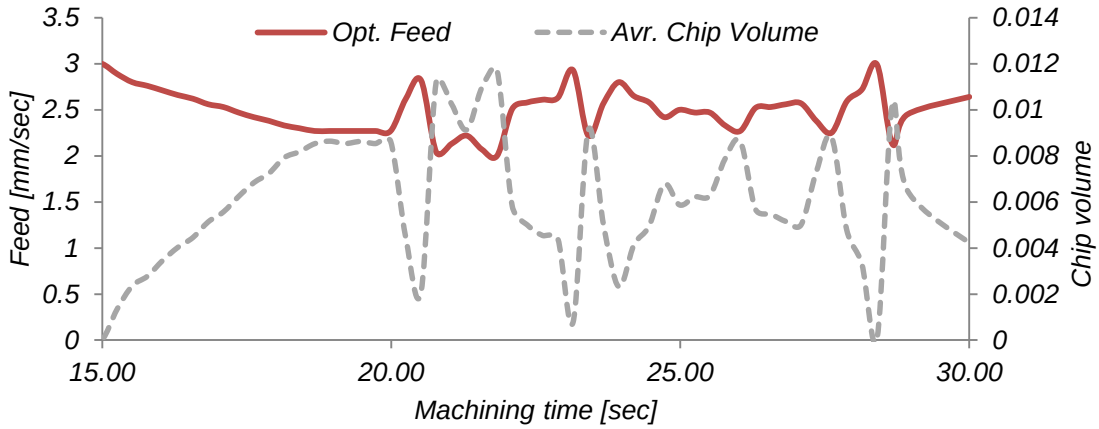


Figure 3-21 : Average Chip volume and optimized feed rates of NC block segment during the pocket machining for mill diameter of 0.794 mm

Pocket machining with tool diameter of 0.794 mm was repeated with the optimized program, generated by the method mentioned above. The measured resultant forces are shown in Figure 3-22 and three dimensional profiles of the measured resultant forces along the tool path before and after optimization are shown in Figure 3-23. It shows that the average resultant force decreased approximately 28% and the machining time was reduced 13% as a result of the optimization

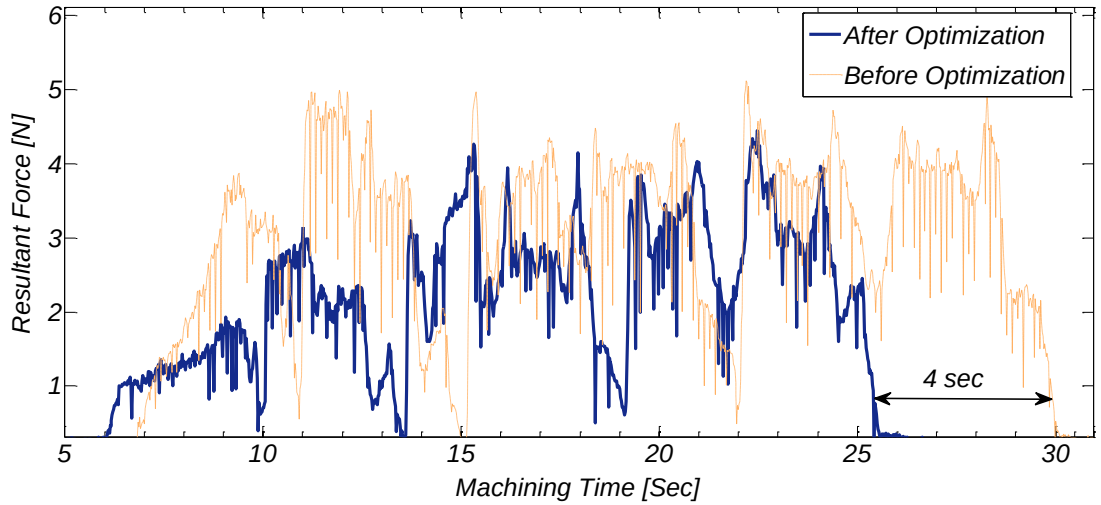


Figure 3-22 : Resultant forces during machining before and after optimization with mill diameter of 0.794 mm

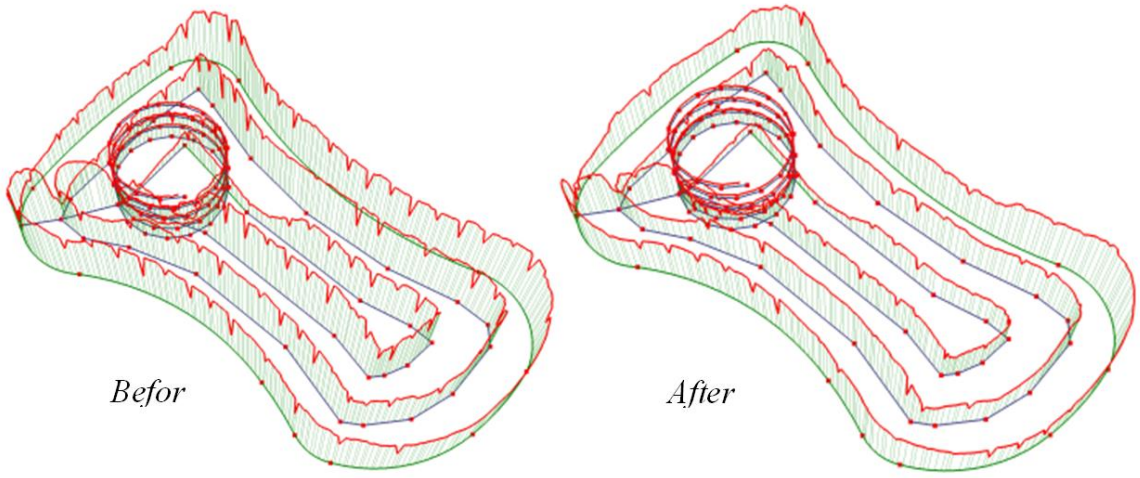


Figure 3-23 : 3D plot of Resultant forces before and after optimization with mill diameter of 0.794 mm

By calculating average chip volume values and defining the optimum feed of each segment in G-code program, the feed variation issue caused by adding extra points in the program could be solved. As a result, average cutting forces and machining time were decreased, leading to improved productivity and tool life. This confirms that smaller segment lengths would be an issue in feed rate optimization of the micro-milling process. It would be important to employ tool path strategies, where segment lengths are controlled or better interpolation methods are employed to maintain the specified feed rate.

3.5 Conclusion

Feed rate optimization has been conducted based on chip volume for micro milling. An algorithm has been developed to determine the optimum feed rate for each tool position using the chip volume at the corresponding positions. After optimization, the resultant forces and the machining time decreased substantially. However, it was observed that as the tool diameter and the size of the feature to be machined decrease, segmentation length along the tool path seems to play an important role in controlling the feed rate during machining. The actual feed rate measured turned out to be lower than the specified feed rate after optimization, possibly due to smaller segment size resulting from increased segmentations. By keeping the segment length unchanged and calculating average chip volume values for each segment, the resulting optimization decreased the average cutting forces and matching time.

Chapter 4: CMM Software development for Micro Probing Based on a Rotating Wire Probe and an AE Sensor

A new probing system that consists of a wire probe and an acoustic emission (AE) sensor for touch detection has recently been reported [3]. It was demonstrated that the use of a rotating wire and an AE sensor could be quite promising as a cost-effective and accurate probing system. A wire probe is relatively easy to fabricate, and the AE-based detection method is very cost-effective and highly sensitive. Moreover, the spinning probe overcomes the snap-in and adhesion of probes to the workpiece surface during the touching operation, which is a major problem for typical microscale probes. However, only the feasibility of the new probing system was reported, and an in-depth evaluation of the system and its automated scanning capability was not investigated.

In this section, an automated scanning system using a rotating wire probe and an AE sensor for touch detection is presented and the basic concept of the AE-based sensing mechanism and the wire probe geometry are addressed. An automatic part measurement algorithm is introduced, and reports on experiments conducted to evaluate the developed CMM software are made.

4.1 Rotating Wire Probe and Sensing Mechanism

Figure 4-1 shows a diagram of the wire probe structure. A stainless steel high strength wire with a diameter of 0.102 mm was bonded to the end of the stylus. The wire was then bent at an angle (β) of 45°, such that, as the stylus spins, the rotational diameter of the wire tip becomes the probe's effective diameter (D_e), which determines the smallest

feature size that can be measured with the probe. The effective diameter (D_e) can be written as:

$$D_e = 2R_e = 2L\sin\beta \tag{4.1}$$

As shown in Eq. 1, the effective diameter depends on the length of the wire. Fabrication of a small-diameter probe can be achieved simply by attaching a short length wire. As long as the wire extends farther than the edge of the stylus, the wire probe can be used for contact sensing. The effective diameter has to be calibrated to identify the position of the probing element.

A custom-built precision-motion system (Alio Industries) with three linear stages that have a resolution of 200 nm and a spindle (NSK E800Z) with a maximum speed of 80,000 rev/min were used to test the probing system. For the probe tip structure, a simple and low-cost interchangeable probe was fabricated, as shown in Figure 4-1. A thin stainless steel wire with a diameter of 102 μm was bonded into a stainless steel tube with an inner diameter of 127 μm.

Figure 4-2 shows a scanning electron microscopy (SEM) image of the probe and the wire tip. An edge can be seen at the end of the wire, because the wire was cut using a cutter. Some burrs are also visible on the wire tip. An AE sensor (PA Nano30) was used to detect contact between the probe and the surface.

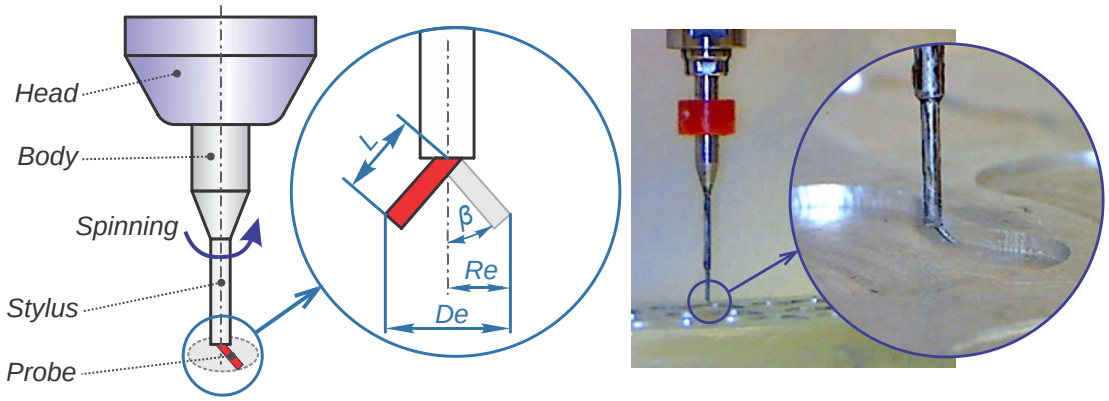


Figure 4-1 : Diagram of the wire probe tip and geometry

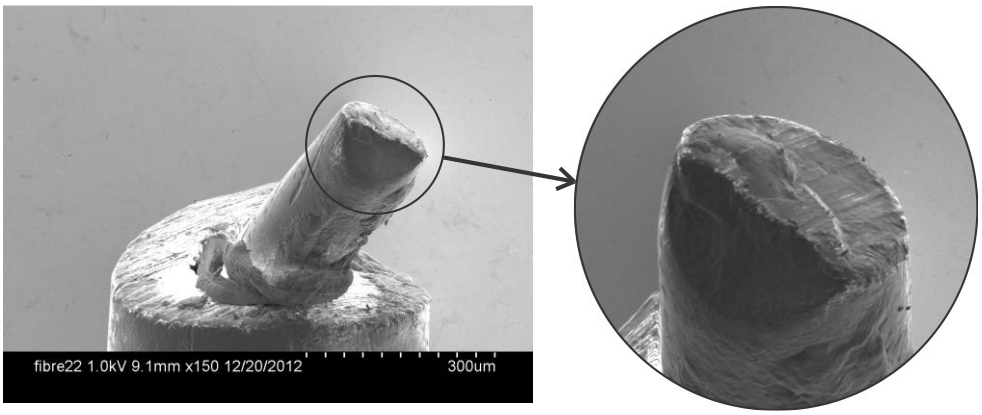


Figure 4-2 : SEM images of the probe tip

The contact sensing method using a wire probe and an AE sensor is illustrated in Figure 3. The probe was rotated and commanded to approach the object at a given approaching feed. As soon as the tip made contact with the part surface, the AE sensor picked up a burst of generated AE signals, which were generated due to the rubbing between rotating tip and the part surface.

There is a slight dependency of the distance between the contact location and the AE sensor on the generated AE signal magnitude. Thus, care must be taken in setting the threshold value used to detect the contact burst, so that the magnitude of the burst is larger than the AE noise, but small enough for highly sensitive contact detection.

For contact sensing, the AE signal was sampled at 20 kHz and sent to the designed control software. The input digital signals from the AE sensor were formed as an array of numbers from the data acquisition (DAQ) driver board. The signal processing module determined the point of contact between the probe and the component. When the values crossed the threshold point, the wire probe was stopped by generating a command for the computer numerically controlled (CNC) controller. Pulses generated from subsequent AE generation were neglected. The contact coordinates of the probe were recorded, and the probe immediately went back to the start position.

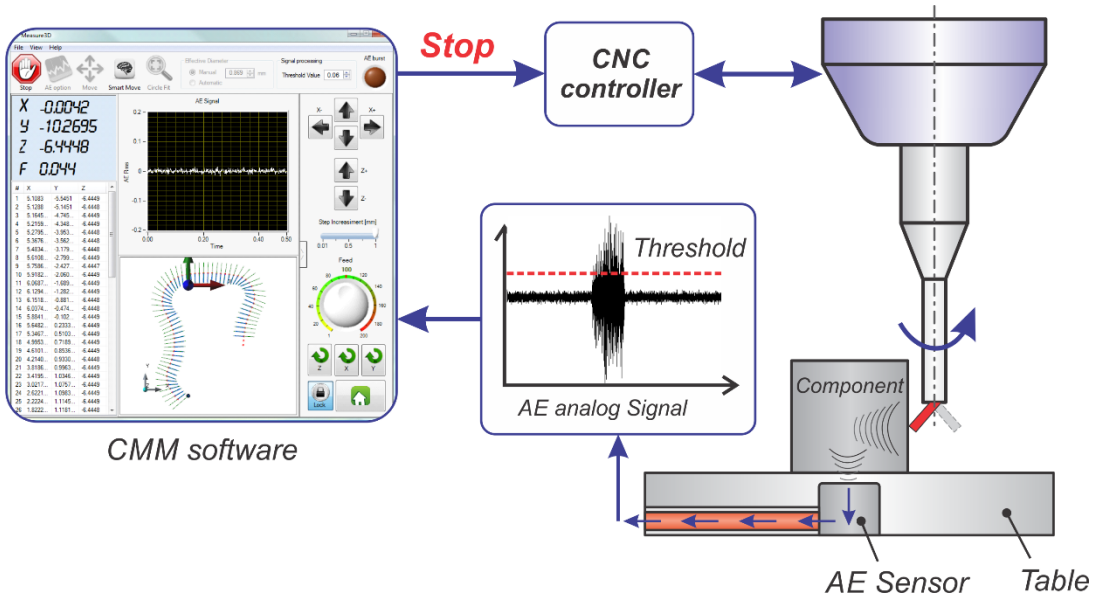


Figure 4-3 : Microprobe measurement touch sensing system

The probing system could detect contact regardless of the approaching direction. Thus, one sensor was used to detect contact in all three orthogonal directions (X, Y and Z). However, the contact mechanism was slightly different in the Z direction. In the Z direction, the wire tip was in contact with the surface for the entire revolution; therefore, the duration of wire rubbing against the object surface was much longer than in the X and Y directions.

The resolution of the probing system depended on the feed rate (μm/rev) determined by the rotating speed (rev/min) and the approaching feed (mm/min). When the microprobe was at the surface, the touch between the probe tip and the object surface was guaranteed after a full revolution of the probe, because only one probe tip was rotating.

4.3 Automated Scanning Algorithm and Control Software

Control software was developed to control the probing system with a user interface. The interface also allows for the translation of the axes. For manual contact position measurements, the probe is first brought to a position near the surface of the object and then commanded to approach the surface at a given feed rate. The interface also displays the probe's current position, and the recorded contact positions are graphically displayed on the interface along with the approach direction for the corresponding contact position.

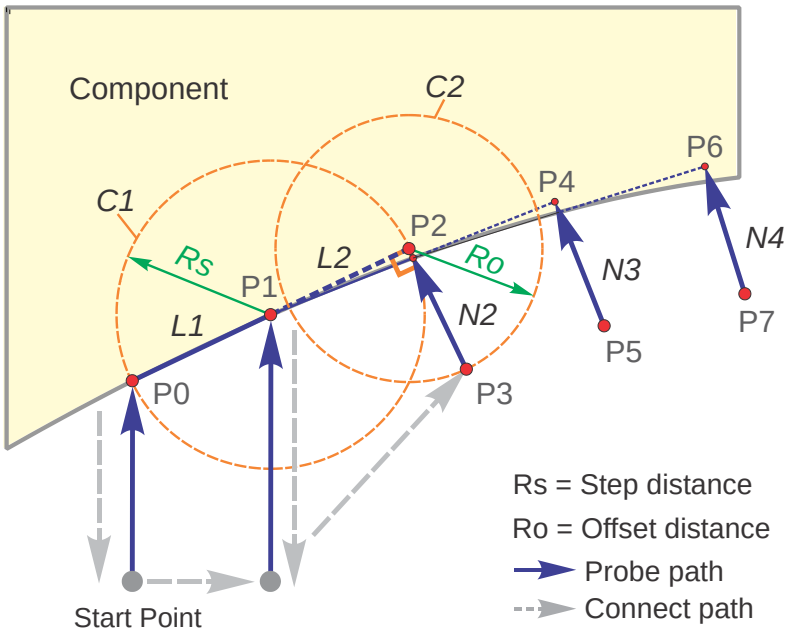


Figure 4-4 : Automated Probe path generation

The control software allows for automated scanning of an object in the X-Y plane, regardless of the object's geometry in the plane, based on the algorithm developed, as

shown in Figure 4. Automated scanning is currently limited to the X-Y plane, but scanning algorithms in other planes are under development. Two initial positions (*P0* and *P1*) need to be measured for the automated scanning.

The direction of the probe, the step-over distance (R_s) and the offset distance from the surface (R_o) are set by the user. The values of these parameters are required to start the scanning process. By connecting *P0* and *P1*, line segment *L1* is generated. Circle *C1* is then created with center *P1* and radius R_s. Line segment *L2* is then extended in the direction of *L1* to obtain *P2*, as shown in Figure 4. The X and Y coordinates of *P2* (X_{P2} and Y_{P2}) can be found as:

$$X_{P2} = 2X_{P1} + X_{P1}S^2 + \sqrt{-4X_{P1}{}^2S^6 + 4R_s{}^2S^4 + 4X_{P1}{}^2S^2 + 4R_s{}^2} \tag{4.2}$$

$$Y_{P2} = S(X_{P2} - X_{P1}) + Y_{P1} \tag{4.3}$$

where X_{P2} and Y_{P1} are the X and Y coordinates of *P1*, X_{P0} and Y_{P0} are the *X* and *Y* coordinates of *P0*, and

$$S = Y_{P1} - Y_{P0} / X_{P1} - X_{P0} \tag{4.4}$$

The approach direction at *P2* is set to be normal to *L2* and is denoted as *N2*. The length of line *N2* is determined by offset distance R_o. The probe then approaches the component surface from *P3*, which is the intersection of *C2* and *N2*. The X and Y coordinates of *P3* (X_{P3} and Y_{P3}) can be found as:

$$X_{P3} = 2X_{P2} + X_{P2}S'^2 + \sqrt{-4X_{P2}{}^2S'^6 + 4R_o{}^2S'^4 + 4X_{P2}{}^2S'^2 + 4R_o{}^2} \tag{4.5}$$

$$Y_{P3} = S'(X_{P3} - X_{P2}) + Y_{P2} \tag{4.6}$$

where S' is:

$$S' = -1/(Y_{P1} - Y_{P0} / X_{P1} - X_{P0}) . \quad (4.7)$$

The probe path is then set in the direction of *P3-P2*, which is approximately normal to the component surface, as shown in Figure 4-4. The subsequent approach paths (*N3*, *N4*, etc.) and start positions (*P5*, *P7*, etc.) can be determined in a similar manner.

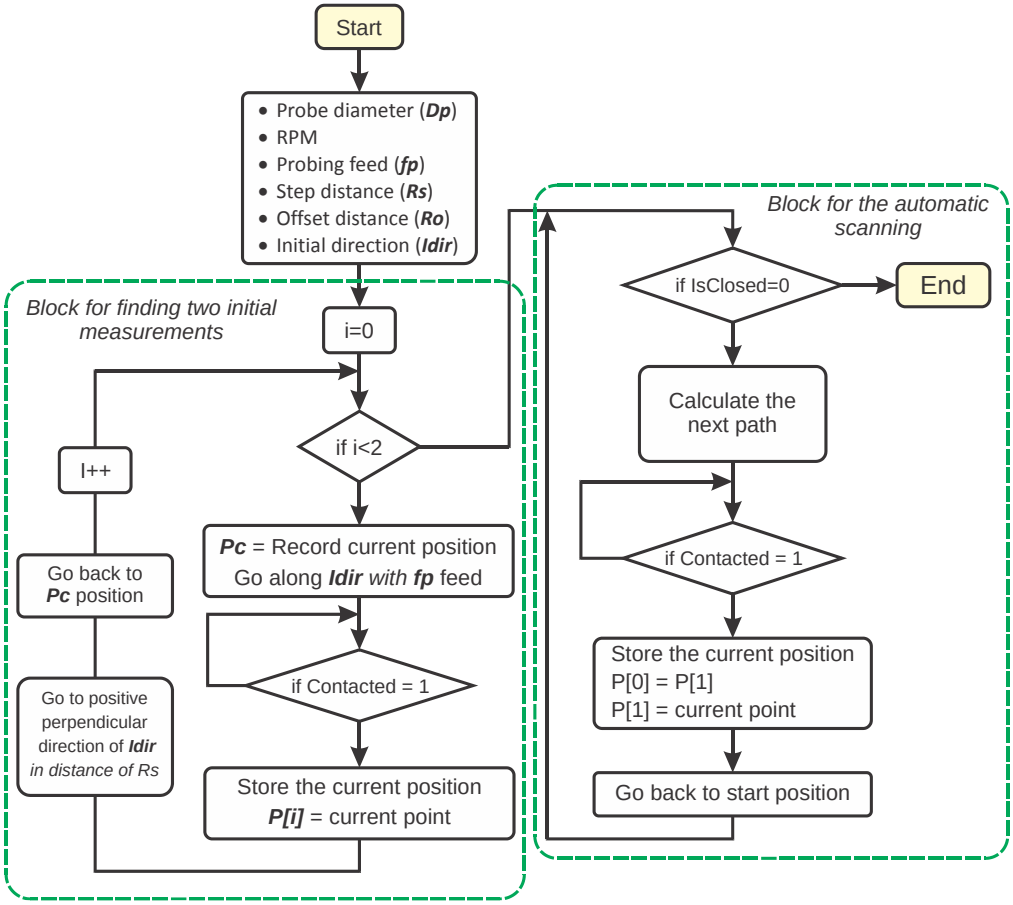


Figure 4-5 : Automated scanning algorithm

For automated scanning, the flow chart of the automated algorithm shown in Figure 4-5 is used. In the algorithm, the required input parameters, including probe diameter (D_p), spindle speed (RPM), probing feed (f_p), step-over distance (R_s), offset distance (R_o) and initial direction (I_{dir}), are set by the user. The algorithm takes two initial

measurements in the initial direction set by the user. In other words, when *i* (measured point counter) is less than two, the probe path is set along the initial direction.

The probe approaches the surface at the feed rate of f_p until the contact between the probe and the component surface is established. The first contact position is stored as the first item (i.e., $P[0]$) in the P list, which is an array of two points. The second contact point is then stored as $P[1]$. Once the two initial points are stored in the P list, a new probe path is calculated based on Eqs. 4.2-4.7; then the next contact point is obtained. The P list is then updated such that it stores the last two contact points in the list. This procedure continues until the measured points form a closed path. All the contact points of the surface are stored in a separate array as measurements. The scanning result of a cylinder and the interface of the developed software are depicted in Figure 4-6.

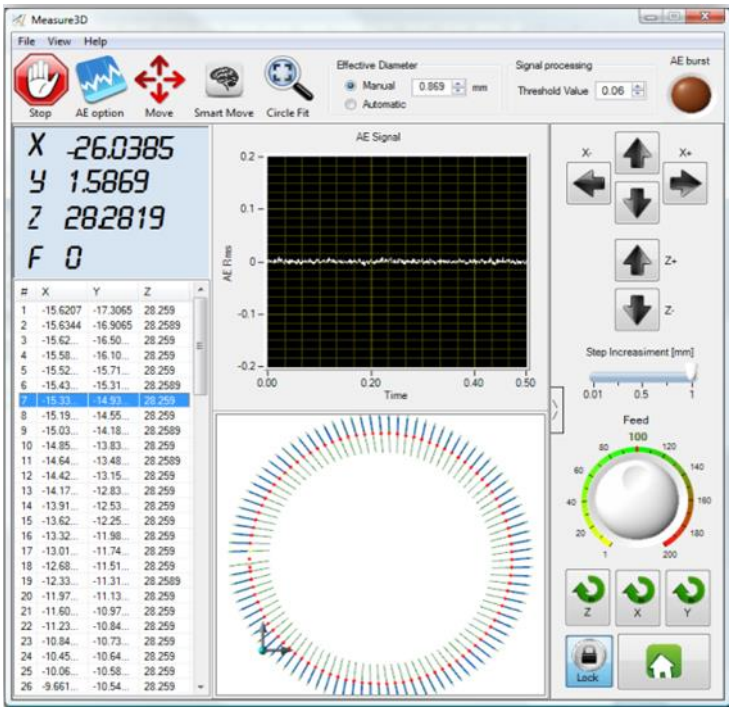


Figure 4-6 : User interface of the developed CMM software

4.4 Evaluation with Artifacts

Experiments were conducted to measure widths of different gauge blocks. All measurements in this section were performed using the automated scanning method described in Sections 4.2 and 4.3. In order to measure the gauge block's width, the effective diameter of the probe (D_e) was first determined using a 20 mm gauge block. One face of the block was scanned to determine the straight edge and a point on the other face was obtained. A normal line connecting the line and the point was taken as the summation of the block's width and the effective diameter. For the probe used in this experiment, D_e was measured to be 1.629 μm. The value of D_e can be easily reduced by cutting the wire shorter. As presented in our previous work [3], D_e is dependent on the spindle speed, due to the associated eccentricity and centrifugal forces. D_e is calibrated before measurement; therefore, as long as it stays consistent at the given conditions, the measurement accuracy is not compromised.

Three sets of experiments were conducted. The probe was used to measure the widths of three gauge blocks of different sizes (20, 12.7, and 9.525 mm), and each gauge block was measured five times. Figure 4-10 shows the results of the gauge block width measurements. Only the deviations from the corresponding gage block's width have been plotted. With each gauge block, the variation of the width measurements was less than 4 μm, with an average of 3.65 μm. As previously mentioned, this increased deviation may have been due to the wire tip's roughness, the AE sensor's locational dependence, or the stage accuracy. The deviation was consistent with the variation in the straightness measurements in Section 4.1. The probing conditions, wire tip conditions, process

dynamics, and machine accuracy need to be optimized to improve the probing system accuracy. Thermal effects were assumed to be negligible in this study.

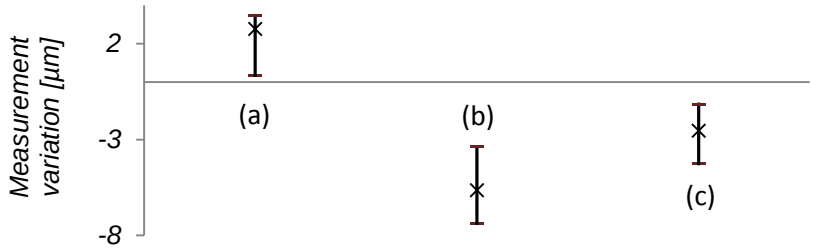


Figure 4-7 : Gauge block width measurements results for (a) 20, (b) 12.7, and (c) 9.525 mm blocks. Only width deviations are plotted

Measurements were also performed on a cylindrical plug gauge with a diameter of 12.7 mm (McMaster-Carr), which was scanned to obtain the block's diameter and the cylinder form. One hundred points around the cylinder were taken to measure the cylinder form. The plug gauge was 50.8 mm long and had a tolerance of 5 µm. A photo of the cylinder gauge and the measurement results are shown in Figure 4-8. The cylinder's diameter was measured to be 12.706 mm. The measured diameter and the deviation of the measurement points from the cylinder diameter were close to the tolerance of the cylinder.

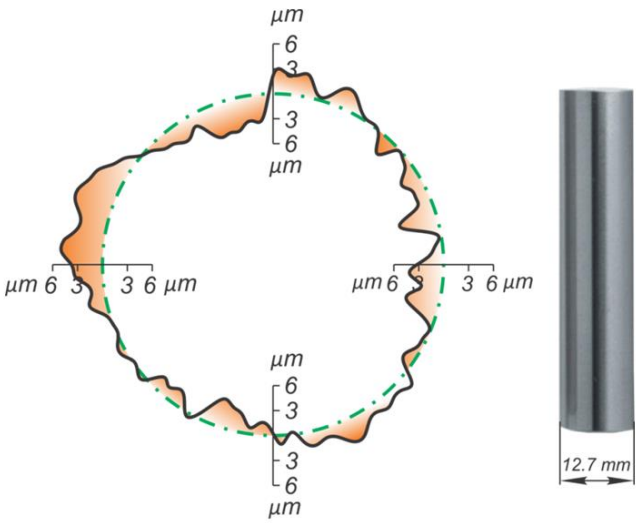


Figure 4-8 : Automated scan results of a cylinder gauge block

The probing system was used to measure and verify the dimensions of machined features. Figure 4-9 presents a photo of a milled cylinder with a diameter of 12.7 mm and the measurement results of the cylinder form. The measured diameter was 12.695 mm. Figure 4-9 shows a three-lobe type form error associated with the cylinder. The form error was also much greater than the form error of the cylindrical plug gauge.

A micro end-mill machined pocket was also scanned. The points taken during the automated scanning are shown in Figure 4-9. Deviations of the positions of the measured points from the pocket design geometry were as high as 80.0 μm because it was a rough milled pocket. Figure 4-10 shows that the probing system can be effectively used to measure and verify the dimensions of machined features.

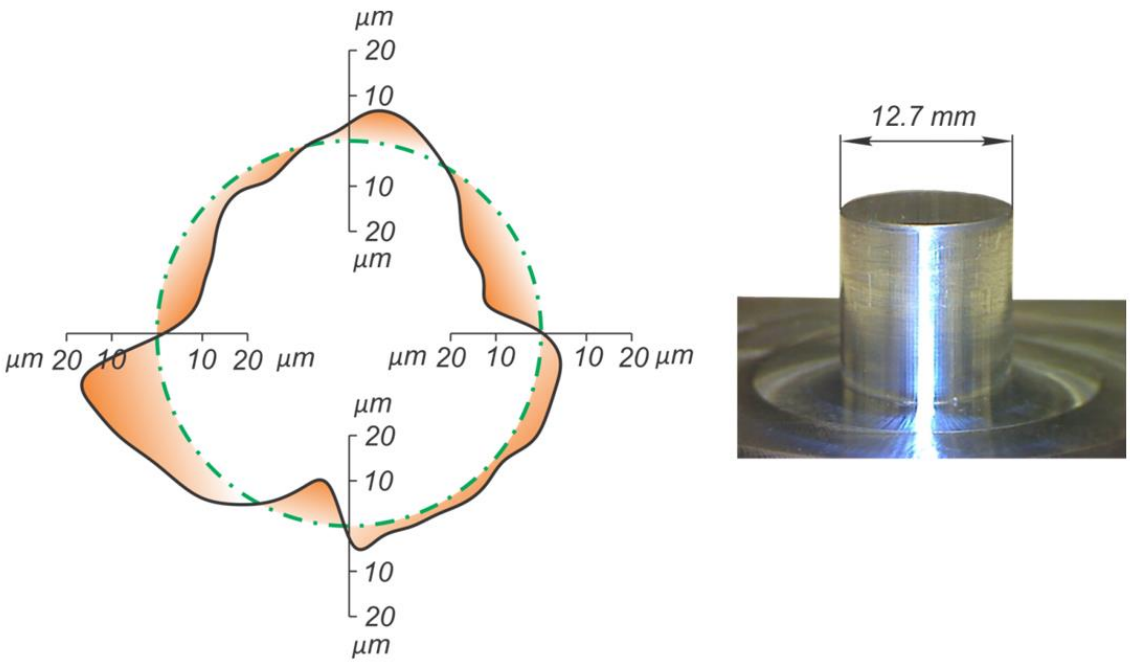


Figure 4-9 : Automated scan results of a rough machined cylinder

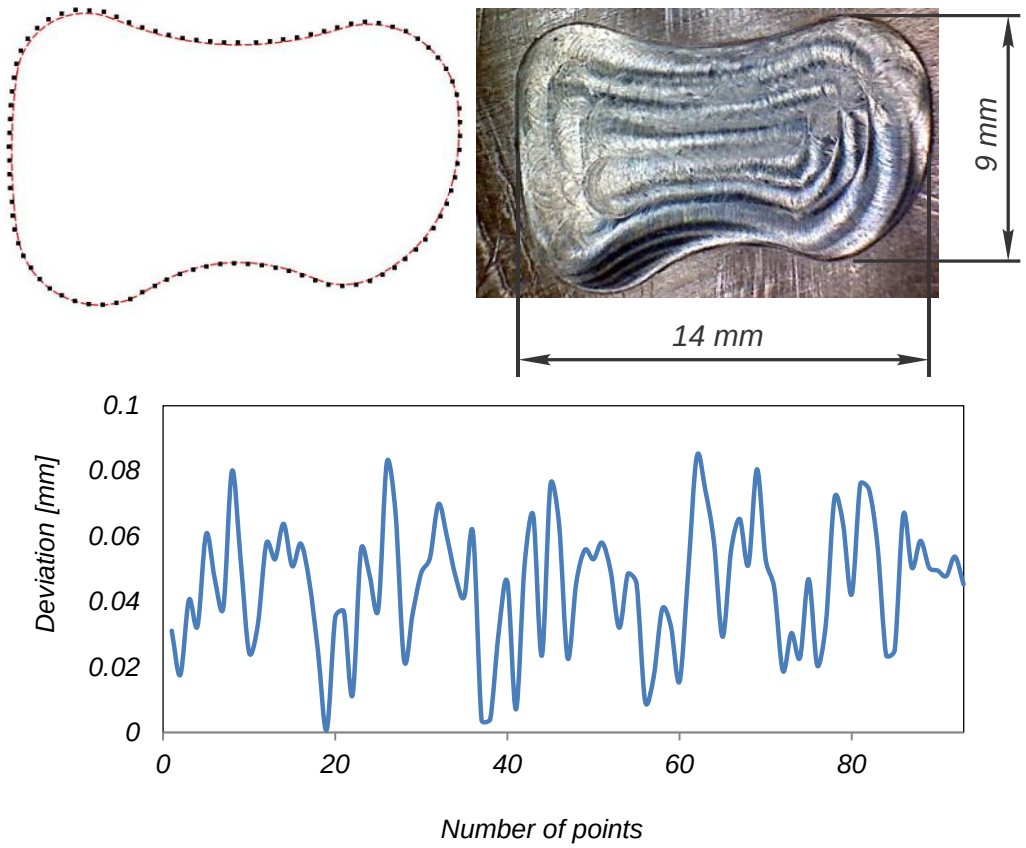


Figure 4-10 : Automated scanning of a rough machined pocket

4.5 Conclusions

The following conclusions can be drawn from this section:

- The automated scanning algorithm works on the X-Y plane.
- Automated scanning of calibration artifacts and machined features have been achieved. The system can be mounted on a conventional micro-milling machine to measure the component dimensions. Only a platform with embedded AE sensors, a spindle with the probe and a DAQ card are needed to set up a measurement system.

Chapter 5: 2-Dimensional Ploughing Simulation Software for Micro Flat End Milling

5.1 Ploughing Calculation Model

To simulate 2-D milling, the workpiece needs to be represented as a geometrical model and the material removal also needs to be represented geometrically. There are a number of ways to do this such as 2-D Boolean, using Boolean operations on 2-D shapes; Point cloud, using a large, dense matrix of individual points; 3-D Boolean with Voxel, using Boolean operations on segmented 3-D objects; and Dexel, an array of lines parallel to one axis (X, Y or Z), beginning and ending at an object's boundaries. Dexel was chosen to represent the workpiece, as it allowed for filling large areas without requiring data on every section of the area, resulting in highly efficient computation. In this section, a new implementation of Dexel was created, called Dual-Dexel, in which grids of lines in two perpendicular directions were created.

To simulate the milling process, line segments, which intersected with the mill's boundaries, were trimmed. These lines, originally intruding into the boundaries of the mill, were given new endpoints, at the intersection of the mill and the lines. From here, line segments were connected at the newly-created endpoints of the clipped line segments to create a polyline arc-shape along the edge of the mill.

Figure 5-1 shows the main flowchart describing the total and ploughing area calculation procedure. Initially, required input parameters include stock size (width W and height H), resolution R, tool diameter D, number of teeth n, minimum chip thickness T_{min}, scale factor S and ploughing calculation accuracy T_p which are set by the user.

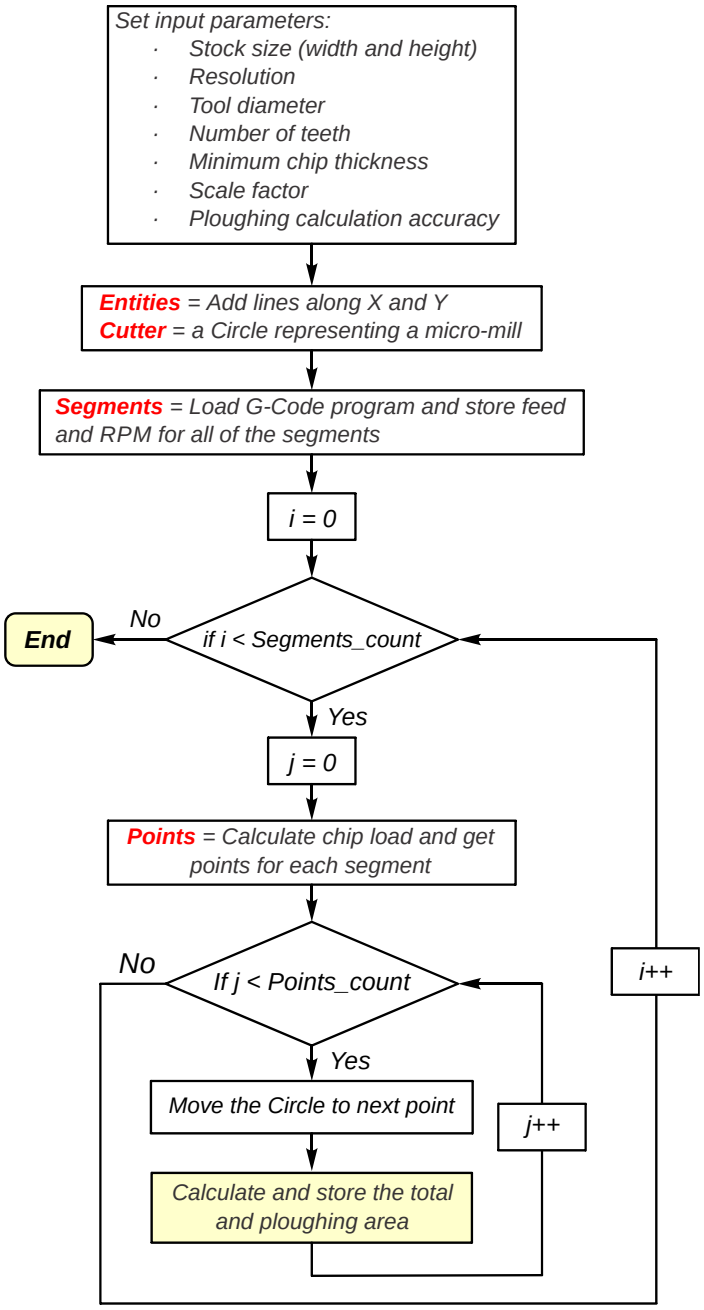


Figure 5-1 : General ploughing simulation flowchart

Lines are added to *Entities* array along X and Y axis in distance of any *R* and depending on *W* and *H*. Additionally, *Cutter*, which is a circle representing a micro mill, was created.

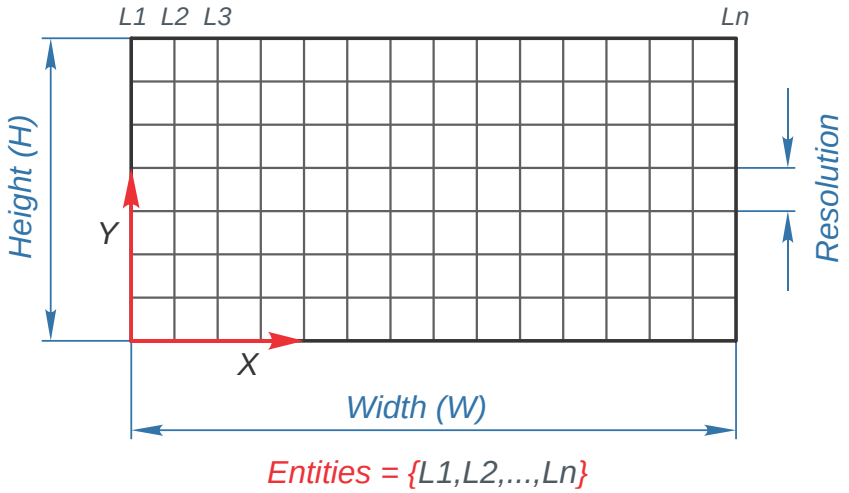


Figure 5-2 : workpiece representing with dexels

The G-code program was imported and read by the developed CNC program parser engine. Then, program segments (line or arc) of the toolpath were stored in array *Segments*, which corresponded to feed and spindle speed.

For each element in *Segments*, points were achieved and stored in array *PointList* in distance of *ft (*feed per tooth value) as shown in Figure 5-3. Feed per tooth for each toolpath segment values was calculated from equation (5.1).

$$f_t = (f / RPM \times N) \times SF \tag{5.1}$$

where f is the feed rate in mm/sec. Performing the area calculation for each created segment by feed per tooth (f_t) resulted in many points. Therefore, the chip load value was multiplied by a fixed scale number (SF) to decrease the number of points and improve the calculation time.

For each point P in array *PointList*, *Cutter* was moved to the coordinate of P wherein the total and ploughing areas were to be calculated.

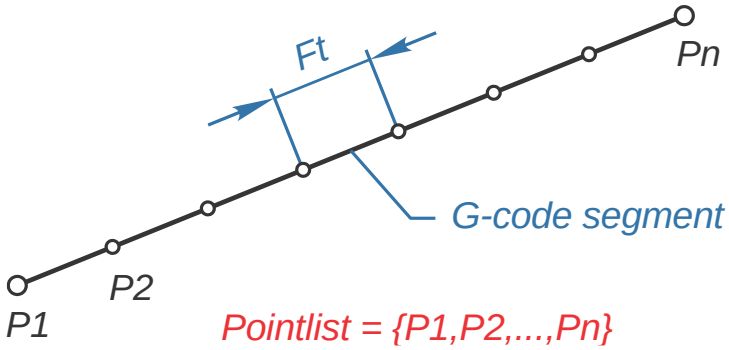


Figure 5-3 : G-code toolpath segment converting to an array of points

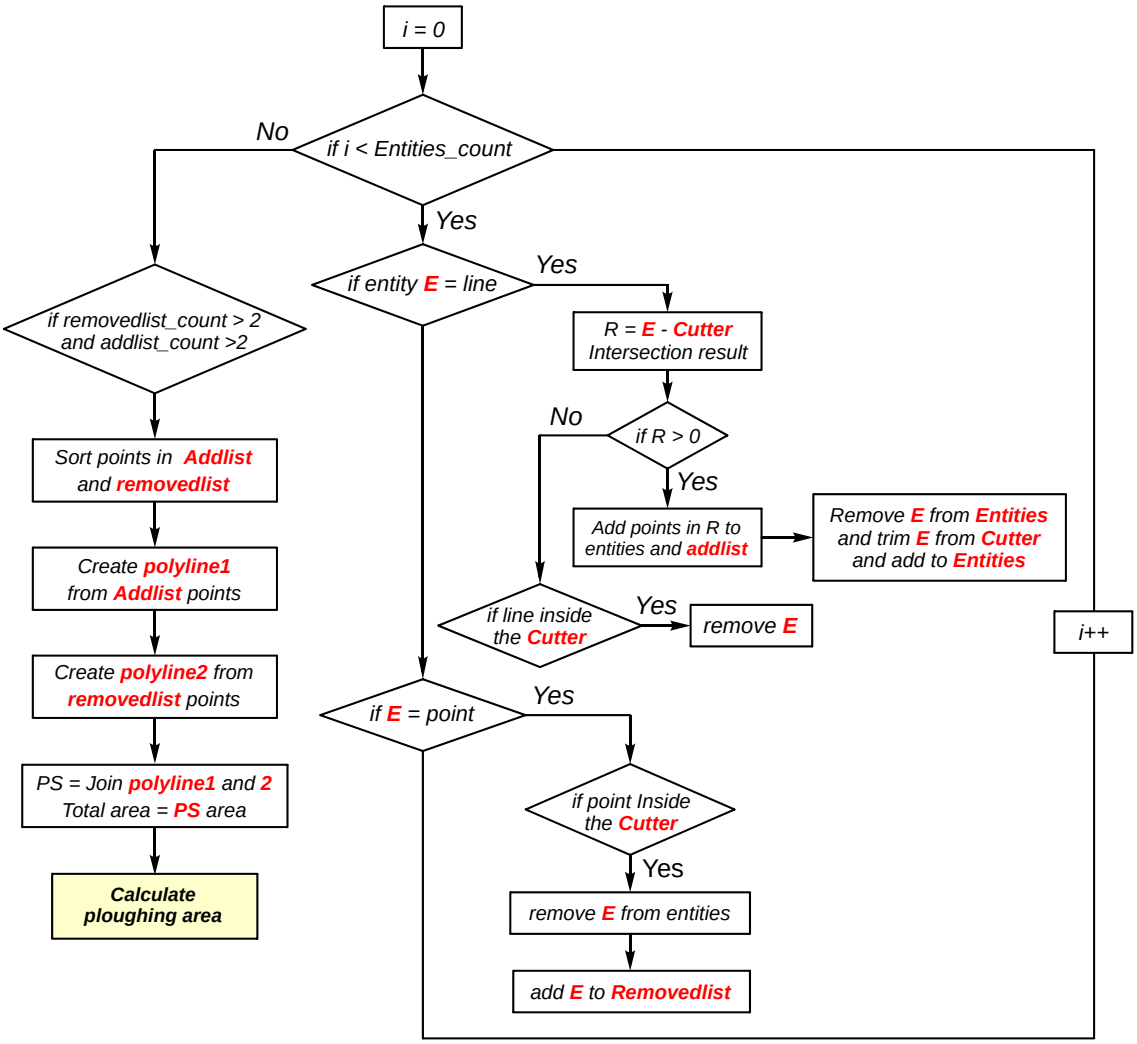


Figure 5-4 : Intersection area calculation flowchart

Figure 5-4 describes the procedure for the total area calculation. When *i* (entity counter) was less than the total number of elements in the array *Entities*. Entity *E* (line or

point) was equal to the corresponding element *i* in *Entities*. If *E* was a line, intersecting with the *Cutter* at particular points, the resultant intersecting points were stored in R (array of points). If numbers of points in *R* were more than zero, points in R were added to *Entities* and *addlist* arrays. Then *E* had to be removed from *Entities* and trimmed from the C*utter*. New trimmed line or lines were added to *Entities* as shown in Figure 5-5.

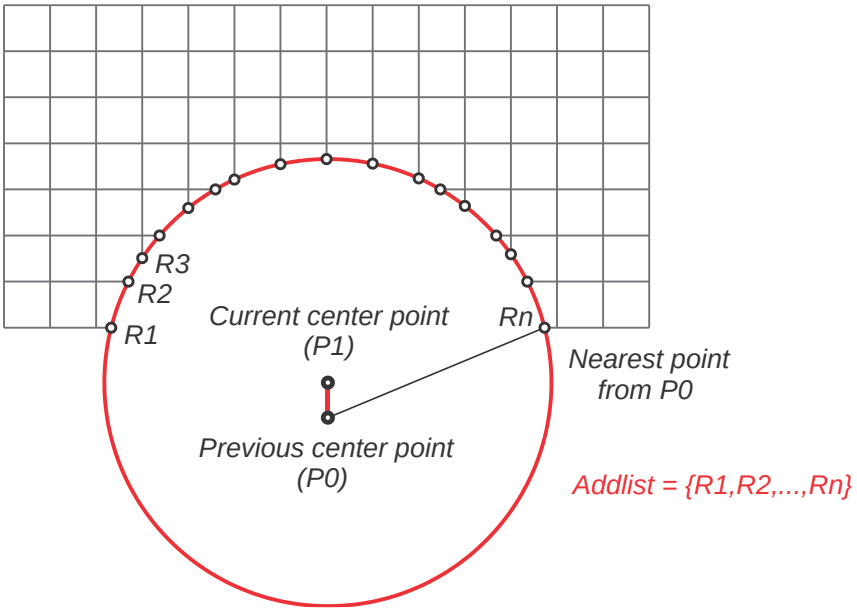


Figure 5-5 : Dexel trimming and finding intersected points

If numbers of points in R were zero and E was located inside the cutter, then *E* was removed from *Entities*.

When E was a point located inside the cutter, *E* was removed from *Entities* and added to array *removedlist* as depicted in Figure 5-6.

When *i* was more than the number of elements in *Entities* and if the numbers of points in *addlist* and *removedlist* were more than two: points in *addlist* and *removedlist* were sorted.

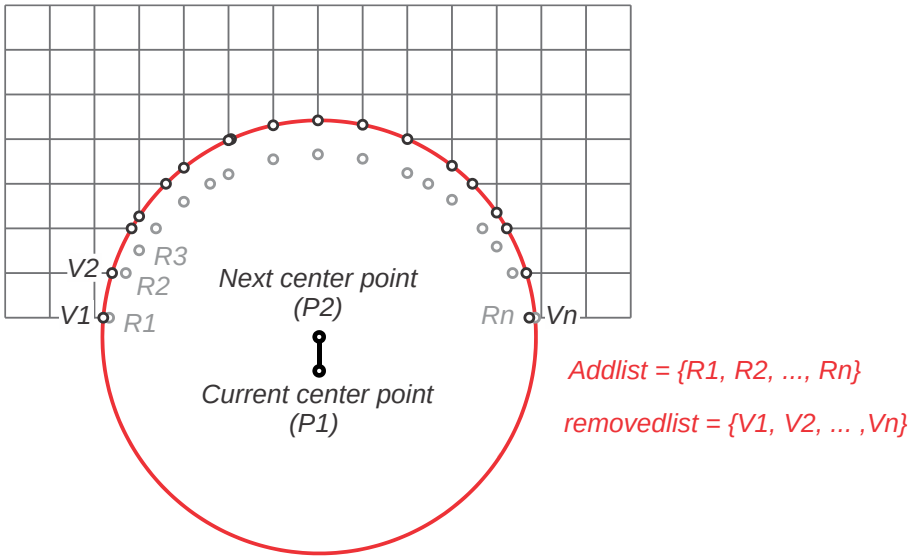


Figure 5-6 : Dexel trimming and creating arrays of removed points and added points

In the sorting method, first the corner point (the closest point from the previous cutter center point) is found, then points in *addlist* and *removedlist* are sorted based on distance from the corner point. Next, *polyline1* and *polyline2* are formed from sorted points in *addlist* and *removedlist* respectively. As shown in Figure 5-7 polyline *PS* was created by connecting *polyline1* and *polyline2*. Finally, total area (*PS* area) was defined by using equation (5.2).

$$A = \frac{1}{2}\sum_{i=0}^{N-1}(X_i Y_{i+1} - X_{i+1} Y_i) \qquad (5.2)$$

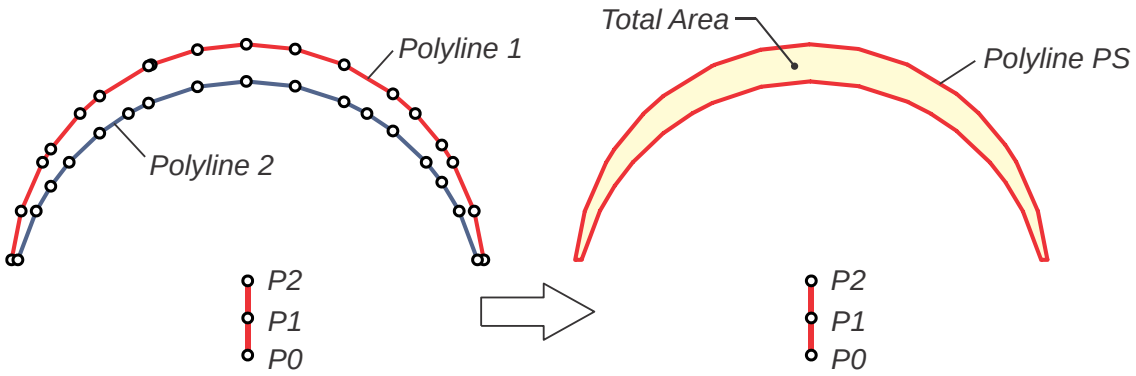


Figure 5-7 : Polygon creation from random points

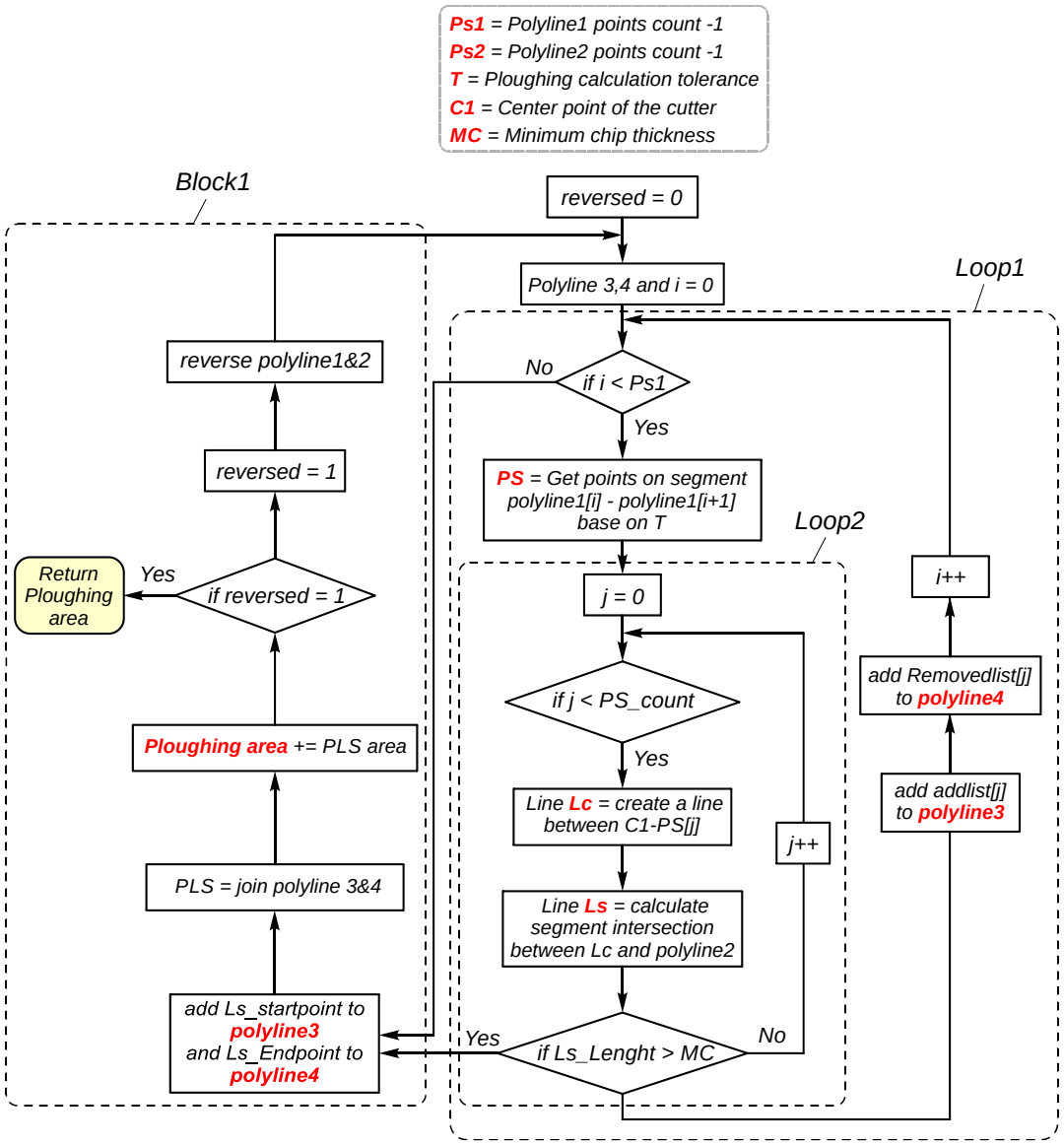


Figure 5-8 : Ploughing area calculation flowchart

In the ploughing area calculation algorithm, initially the default value of *reversed* was set to be zero. In *loop1*, when *i* (*polyline1* point's counter) was less than *PS1* (*polyline1* points count -1), the points on the segment corresponding to the *i* value in polyline1 were added to array *PS* which was based on the ploughing calculation tolerance value.

In *loop2*, when j (*PS* points counter) was less than the number of points in *PS*, line *LC* was created from the center of the *cutter*, to the point position corresponding to the j

value in *PS*. Then Line *L* is shaped resulting in *LC* and *polyline2* intersection. *Loop2* continued until the length of L was more than the minimum chip thickness value (*MC*). After finishing Loop 2, the points index corresponding to the j value in *addlist* and *removedlist* was added to *polyline 3* and *polyline 4*, respectively, and the program goes back to the start of *loop1*.

In *Loop1* and *Loop2* conditions, when *i* was not less than *PS1* or when *LS's* length was more than MC, the program jumped to *Block1*.

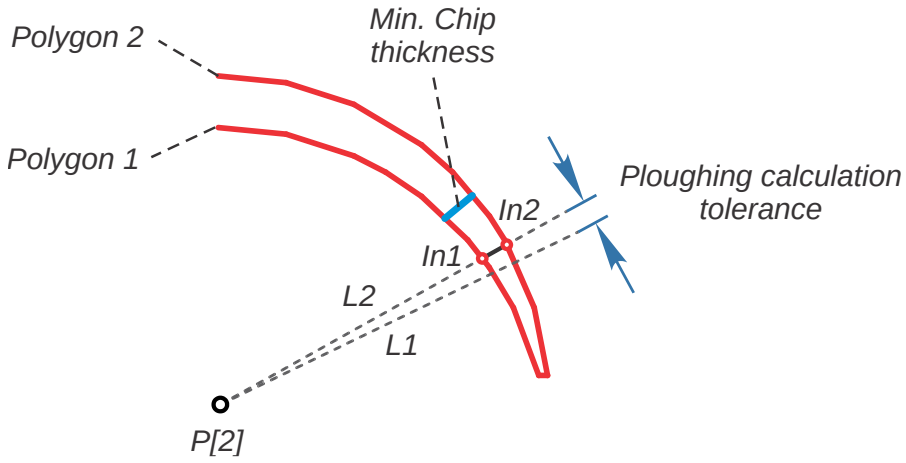


Figure 5-9 : Finding the intersection between a line and two polygons

Initially in *Block1*, the start and end point of *L* were added to *Polyline3* and *polyline4* respectively. Polyline *PLS* was created by joining *Polyline3* and *Polyline4*, and then the *PLS* area was added to the *Ploughing area* value. The *Reversed* value changed to 1 and the order of points was reversed in *Polyline 1* & *2*. Finally the program returned to the initial step before starting *Loop1*. In the case while R*everse* value was 1, the program returned the value of *Ploughing area*.

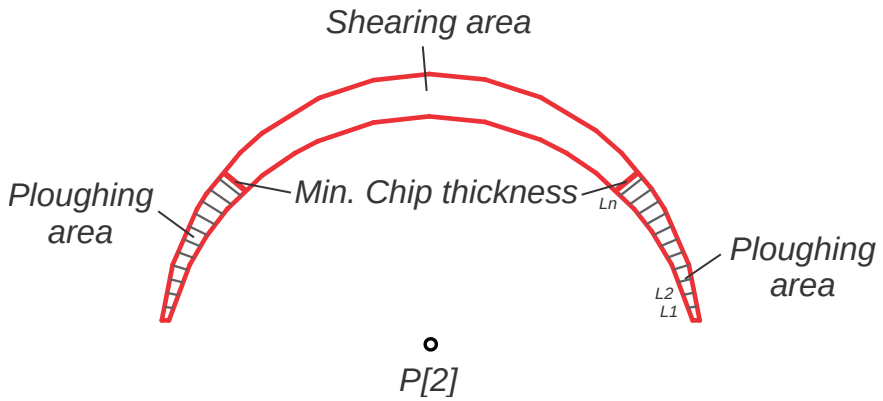


Figure 5-10 : Finding the minimum chip thickness border to calculate ploughing area

Figure 5-11 shows an example of a chip that is created when the mill is not fully immersed in the workpiece. A ploughing area always exists on both sides of the chip, and is calculated with the same method as in Figure 5-3 a chip can be many different shapes, determined by the shape of the workpiece and angle of immersion to its surface.

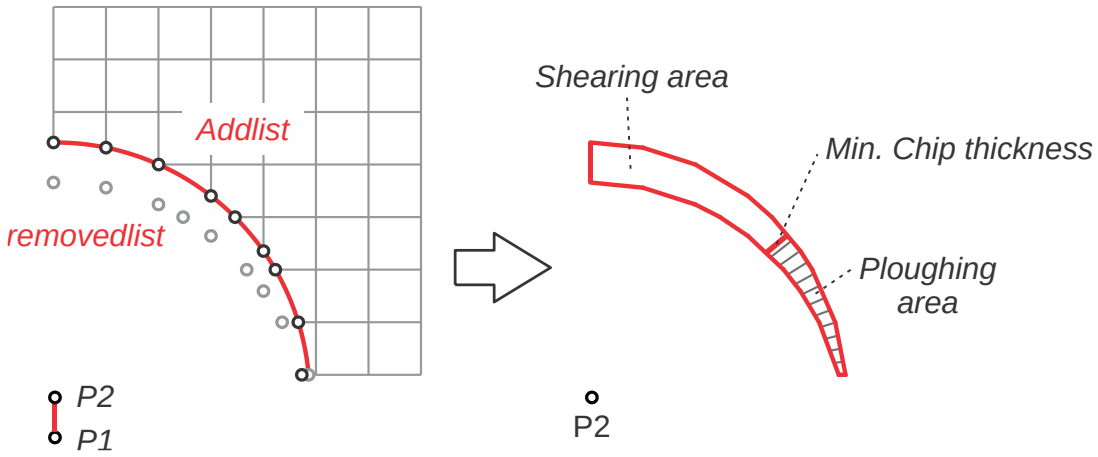


Figure 5-11 : Ploughing area calculation when the mill is not fully immersed

Figure 5-12 shows an example of a chip that is created when the mill is fully immersed and partially located outside of the workpiece. In this situation when D is more than R, intersection areas are divided in two separate shapes (*Sh1* and *Sh2*). Ploughing areas always exist on both sides of *Sh1* and *Sh2*, and are calculated with the same method as shown in Figure 5-12.

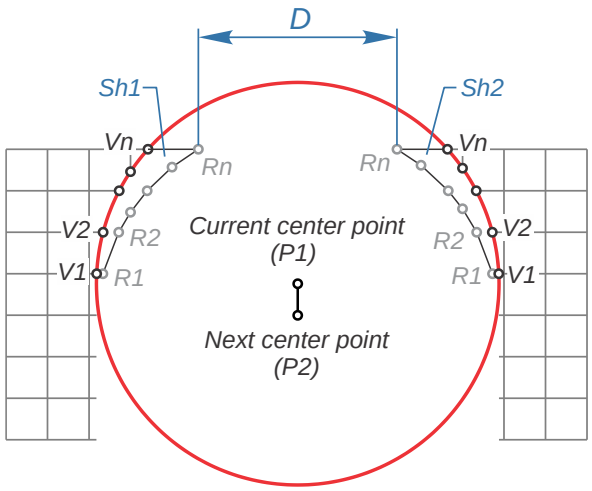


Figure 5-12 : Ploughing area calculation when the mill is fully immersed and partially outside the workpiece

In order to implement the method mentioned above, a micro milling simulator software was developed in C#. For graphical visualization; commercial CAD component *Eyeshot (www.devdept.com)* which is powered by OpenGL, was used. The simulation process and interface of the program are shown in Figure 5-13.

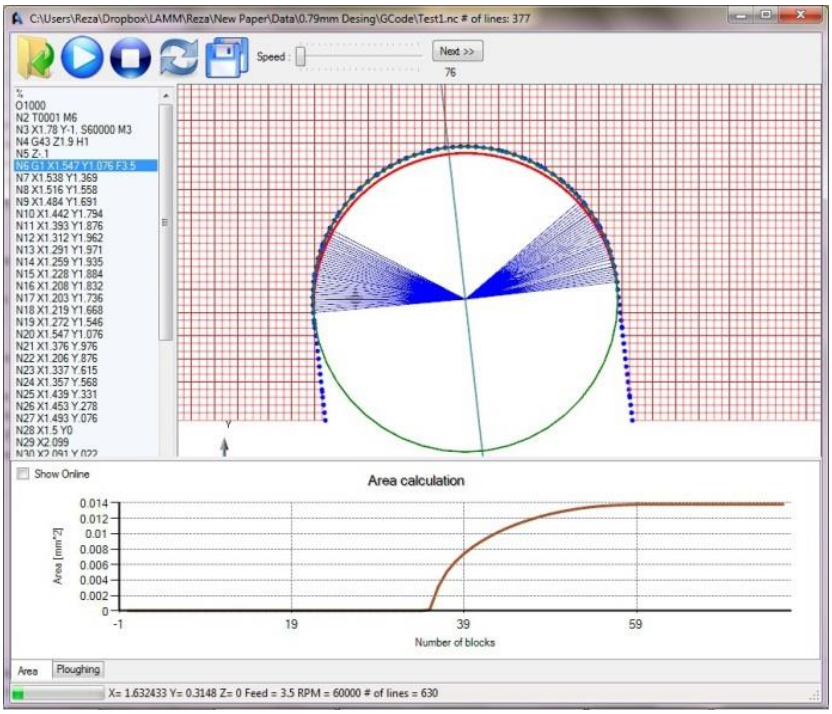


Figure 5-13 : Interface of the micro-milling simulator

5.3 Experimental Setup

To verify the results of the developed milling simulation software, a toolpath as shown in Figure 5-15 was generated in Mastercam. Five nominal feed rates of 0.25, 0.375, 0.5, 0.625, and 1 μm/tooth and a spindle speed of 60,000 rpm were selected as the cutting parameters that resulted in 5 G-code programs. Machining of 5 pockets as depicted in Figure 5-14 was carried out by using a two-fluted 0.794 mm micro end mill. The machined pocket is illustrated in Figure 5-16.

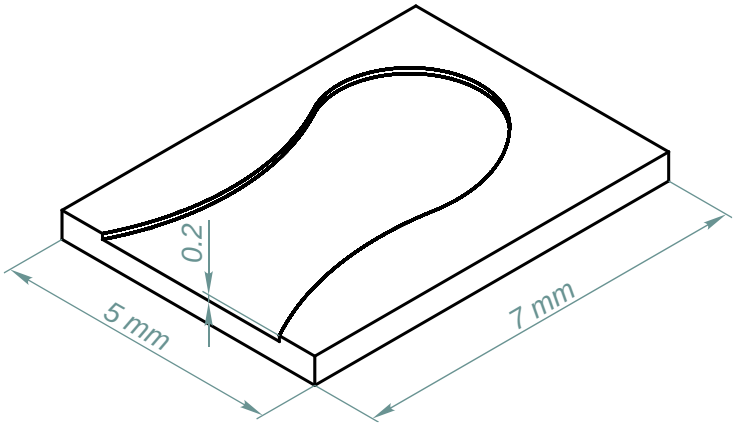


Figure 5-14 : Pocket drawings for 0.794 mm diameter tool

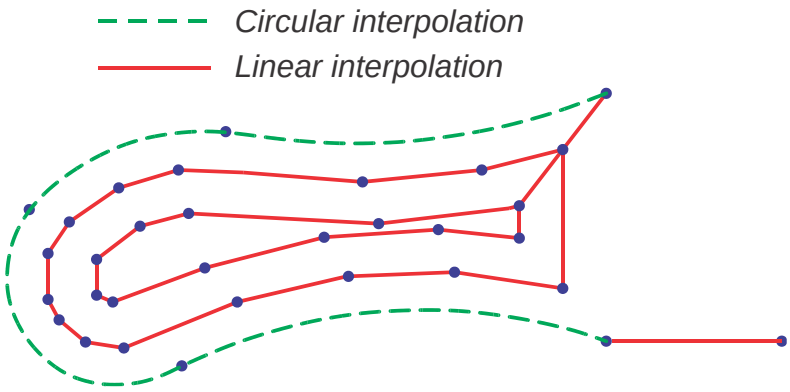


Figure 5-15 : Generated tool paths by Mastercam X6 for tool diameter of 0.794 mm

Figure 5-16 : Machined pocket with tool diameter 0.794 mm

The machine used in this project is an ALIO vertical micro-milling machine which has a spindle speed range from 10,000 to 80,000 rpm. The experimental setup is presented in Figure 5-17. The instantaneous cutting force was measured during the machining operation using a Kistler table dynamometer (MiniDyn 9256C1). The material of the workpiece is Aluminium 6061. The workpiece was clamped on the dynamometer on the feed table of the machining center. Machining was carried out under dry conditions. A data acquisition (DAQ) board from National Instruments (NI-P/N-USB-6251 BNC) was used to acquire the measured forces from the dynamometer. A sampling rate of 100 kHz was used to process the measured force data which is the maximum capacity of the DAQ.

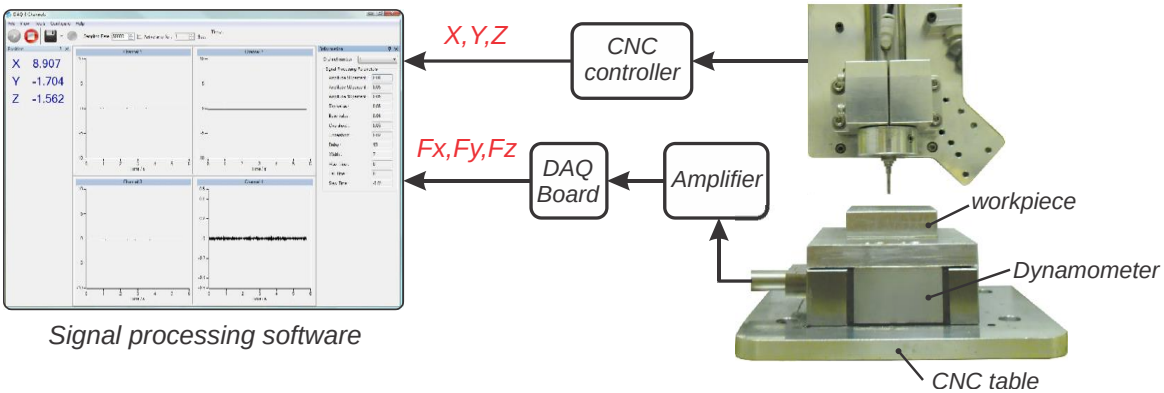


Figure 5-17 : Experimental setup of micro milling operations

5.4 Results

In the simulation software, and the generated toolpath with different feed rates were imported and the following parameters were defined; scale factor=10, resolution=10 µm, tool diameter=0.794 mm, number of teeth=2, minimum chip thickness=2.4 µm and ploughing calculation accuracy=1 µm.

The value of minimum chip thickness was defined by considering equation 1, the tool radius and the scale factor value.

The simulated results of pocket machining using 0.5, 0.75, 1, 1.25, and 2 mm/sec feed rates are presented in the Figure 5-18, Figure 5-20, Figure 5-22, Figure 5-24, and Figure 5-26 respectively. Additionally, the measured resultant cutting forces obtained during machining of the pockets using different feed rates of 0.5, 0.75, 1, 1.25 and 2 mm/sec are shown in Figure 5-19, Figure 5-21, Figure 5-23, Figure 5-25 and Figure 5-27 respectively.

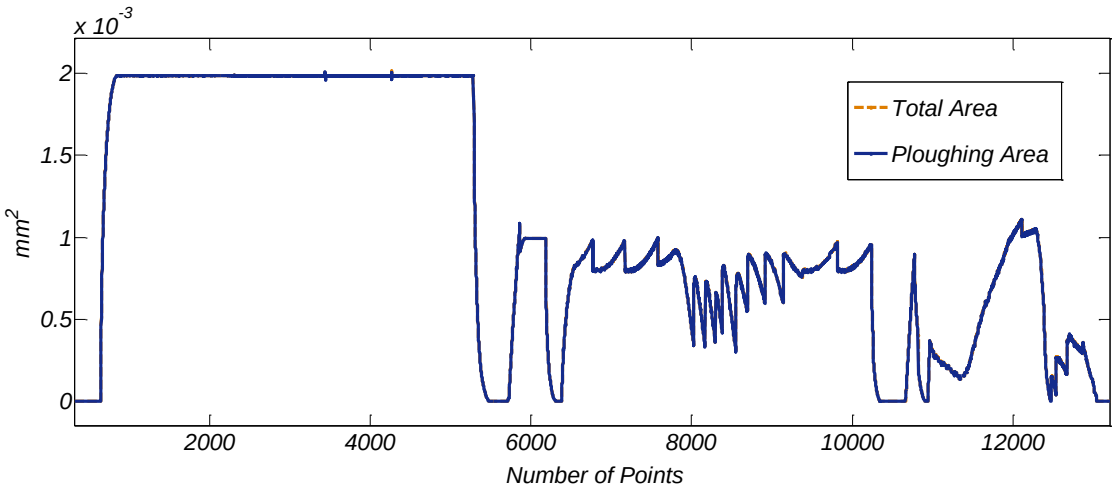


Figure 5-18 : Result of the simulation using 0.5 mm/sec feed

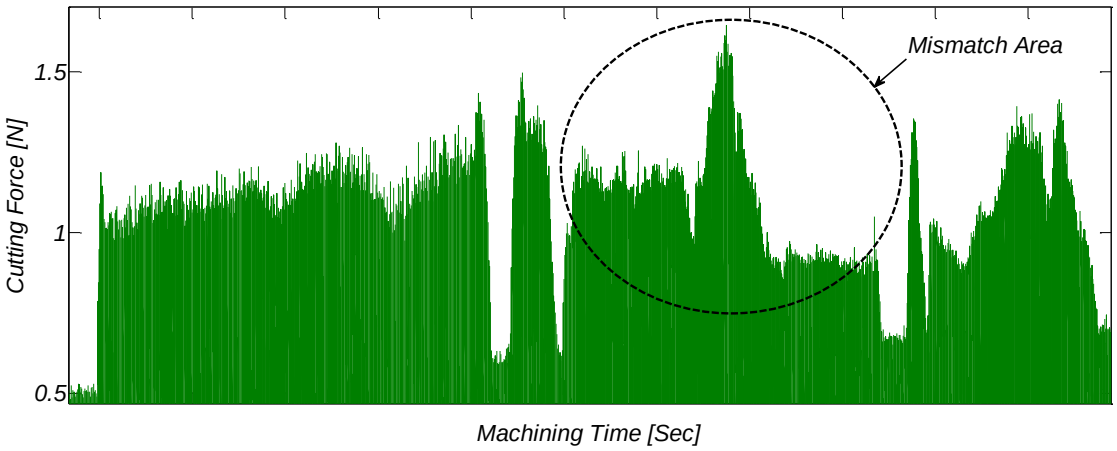


Figure 5-19 : Resultant forces with feed rate of 0.5 mm/sec

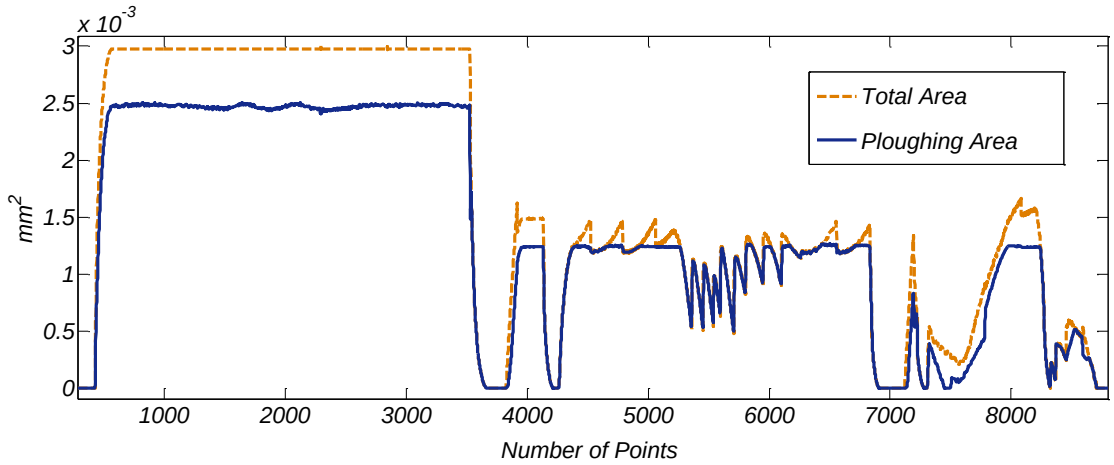


Figure 5-20 : Result of the simulation using 0.75 mm/sec feed

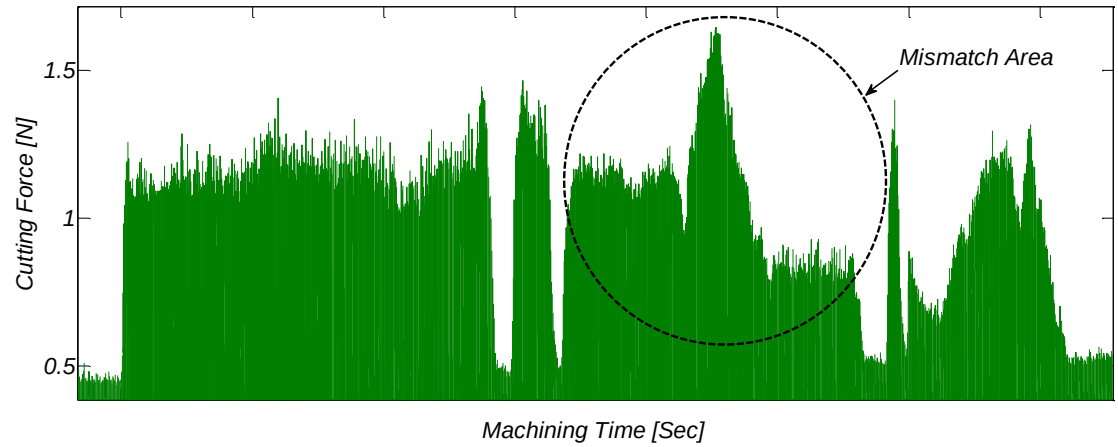


Figure 5-21 : Resultant forces with feed rate of 0.75 mm/sec

As shown in Figure 5-21 and Figure 5-19, the total area simulation model has a good match with the measured cutting forces except in the mismatch areas where the value of the ploughing area is very close to the total area. Additionally, it can be seen that average cutting forces in these conditions are approximately the same, while the feed rate was increased by 50% for the second experiment. Therefore, the effect of the ploughing process may cause force enhancement for the case study with lower feed rate.

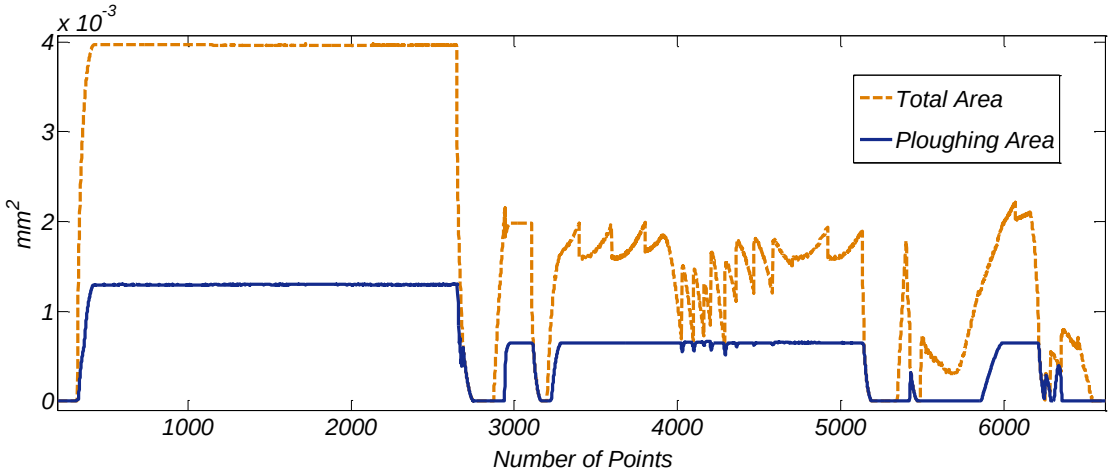


Figure 5-22 : Result of the simulation using 1 mm/sec feed

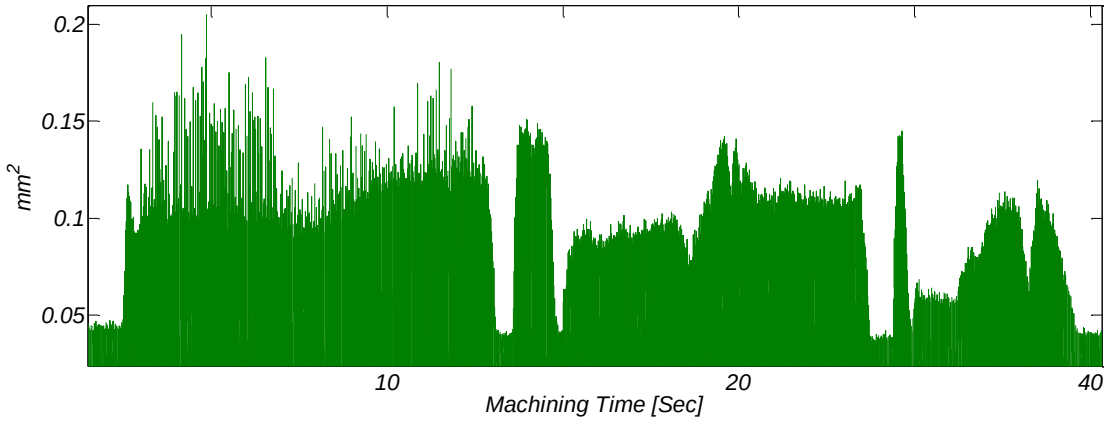


Figure 5-23 : Resultant forces with feed rate of 1 mm/sec

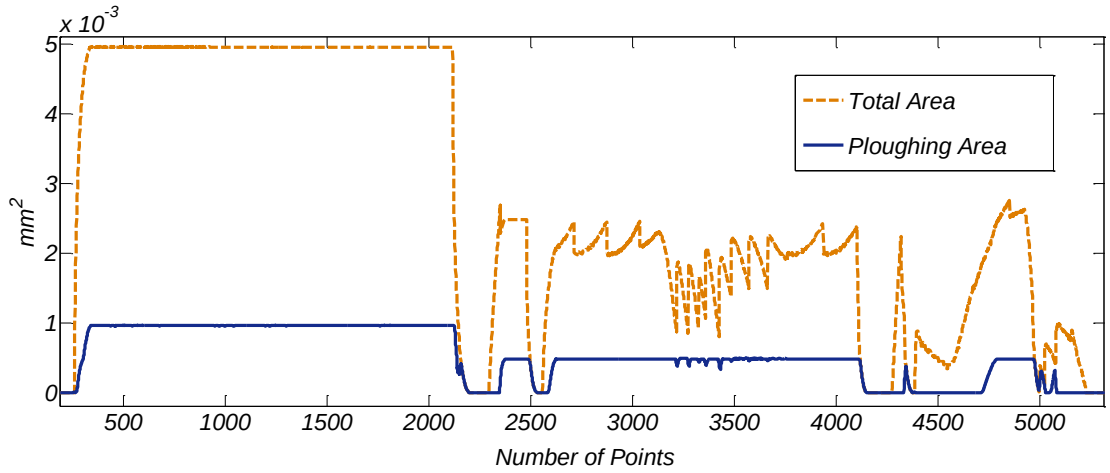


Figure 5-24 : Result of the simulation using 1.25 mm/sec feed

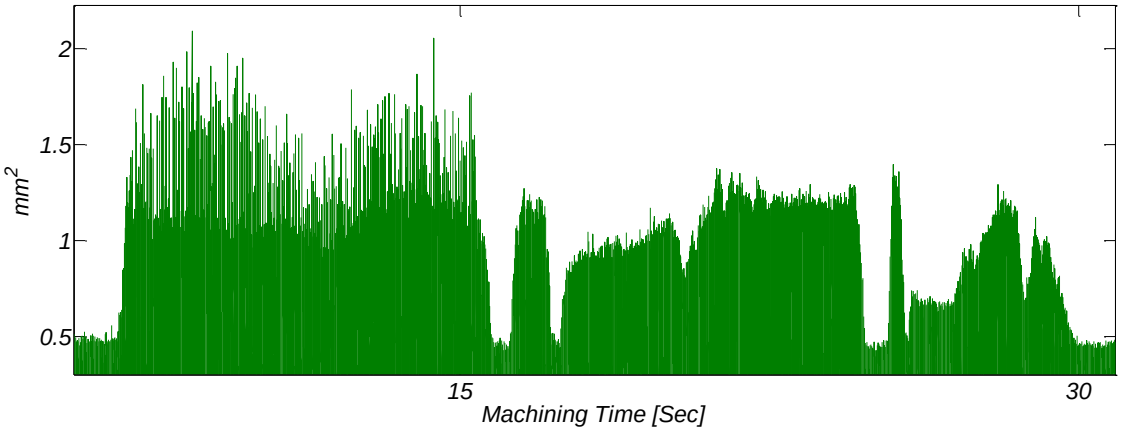


Figure 5-25 : Resultant forces with feed rate of 1.25 mm/sec

As shown in Figure 5-23 and Figure 5-25, the total area model has a better match in comparison with previous cases. However, resultant forces with a feed rate of 1.25 mm/sec have the best match with the total area simulation data. It means that the amount of ploughing during machining with a feed rate of 1.25 mm/sec was not large enough to cause vibration and consequently affect cutting forces during machining.

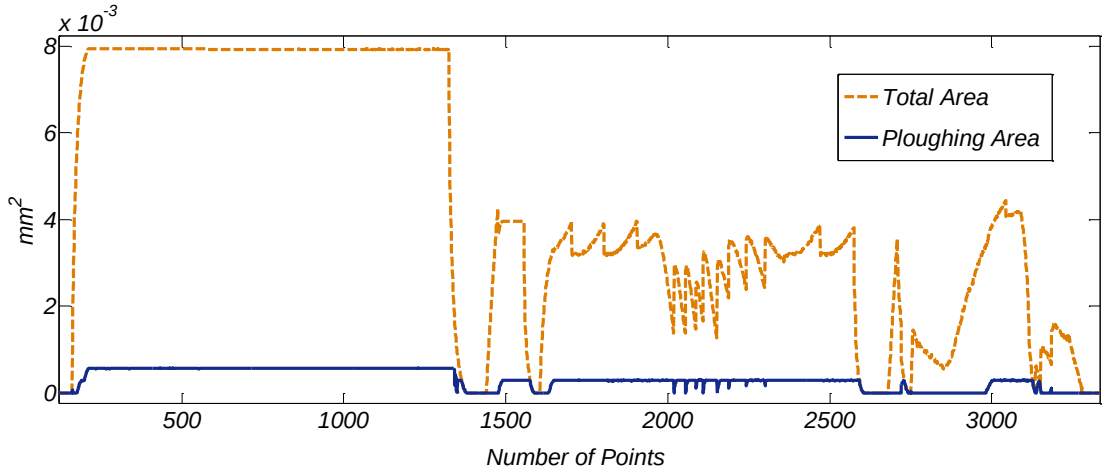


Figure 5-26 : Result of the simulation using 2mm/sec feed

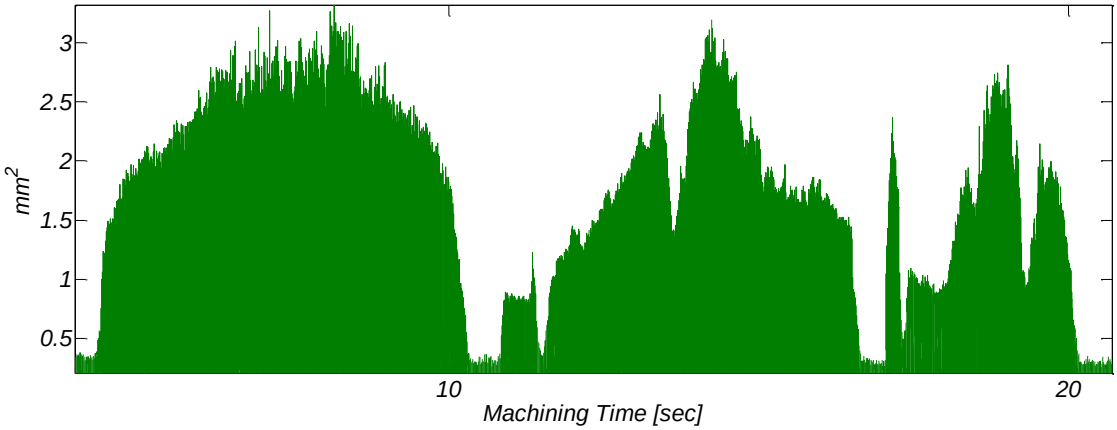


Figure 5-27 : Resultant forces with feed rate of 2 mm/sec

As shown in Figure 5-27, cutting forces in comparison with the previous case (resultant forces with feed rate of 1.25 mm/sec) were increased. The measured cutting forces in the model do not have a perfect match with the simulated total area model. However, increasing cutting forces with the same ratio as the increase of feed rate proves that the amount of ploughing is insignificant.

5.5 Conclusion

A 2-dimensional ploughing volume simulation model developed for micro flat end milling was presented. Real time simulation software based on the dual-Dexel method

was developed. Simulations were performed to calculate the amount of ploughing for a given toolpath. For each simulation, different cutting parameters were used to investigate the ploughing process in micro milling. The simulated results are compared with measured resultant forces for verification of the model. The ploughing simulation software can be used to better understand the micro milling process. By estimating the amount of ploughing, the best cutting conditions could be selected for a given toolpath, resulting in improving tool life and surface quality.

Chapter 6: CAD/CAM Development for Micro Laser Machining

In this section, effective and user friendly CAD/CAM software is presented that automatically generates any three dimensional complex toolpaths according to a CAD drawing. In advanced manufacturing, often it is essential to scan the sample following a complex trajectory which consists of short (few microns) and multidirectional moves. The reported CAM software offers constant velocity for all short trajectory elements and provides an efficient shift of tool path direction in sharp corners of a tool trajectory, which is vital for any laser based precision micromachining.

Figure 6-1 shows the laser materials processing system, which consists of a femtosecond laser unit, an optical system to guide the beam (mirrors, dichroic lens etc.) and a laser power control Density (neutral density filters, half-wave polarizers, focusing lens etc.), auxiliary equipment, and control system. The optical system delivers and focuses the laser beam generated by the femtosecond laser unit on the sample material for material ablation. A set of neutral density (ND) filters is used to tune the output laser power to a suitable value. A computer controlled electronic shutter is used to selectively turn the laser beam on/off. The sample is mounted on a computer controlled 3-axis stage and the CAM software is used to generate a scanning path for the laser head and control beam shutter (on or off). The sample stage consists of three tables (X, Y and Z) and they are driven by linear motors. The features fabricated during focused irradiation of femtosecond pulses can be monitored using a CCD camera mounted above the dichroic mirror.

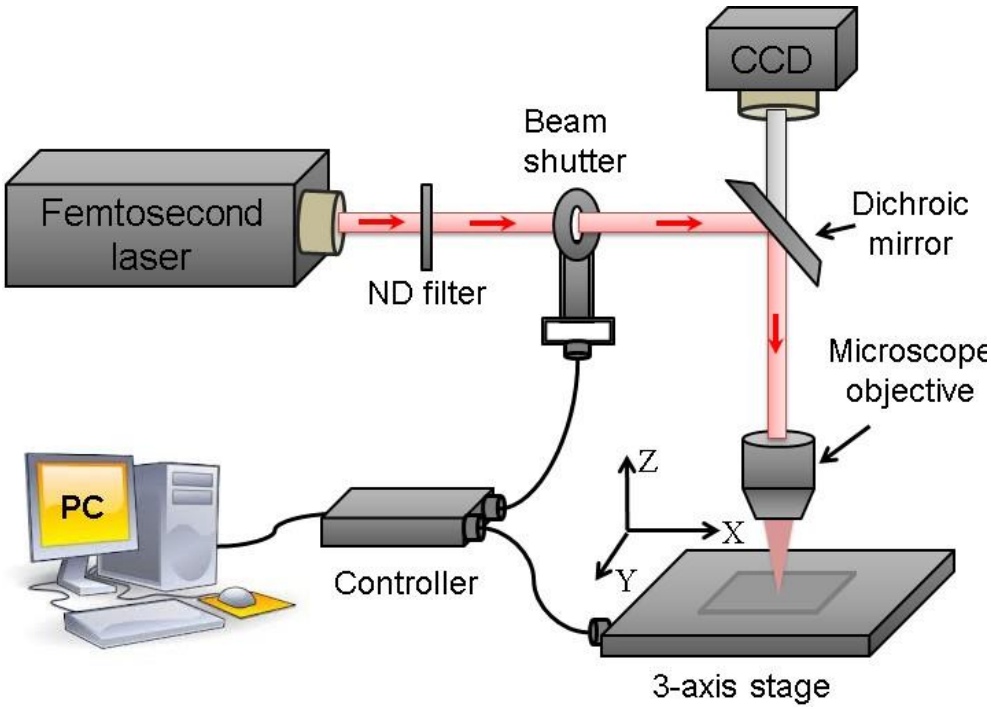


Figure 6-1 : Schematic of computer controlled laser machining system

6.1 CAM Environment

6.1.1 Import DXF and DWG

The ability to import standard 2D and 3D format files, i.e. DWG, DXF, IGES and STEP, is an extremely important feature for all CAM software packages. Laser machining mostly deals with 2D geometries; therefore, the software developed for this research can import DWG and DXF formats that satisfies the user's routine tasks.

Developing the ability to read and parse 2D formats from CAD software packages is a time-consuming process. Therefore, introduced eyeshot framework (www.devdept.com) was employed to add CAD capabilities to the software.

For importing DXF, DWG and STL format files, the ***ReadAutodesk*** class in the eyeshot library is used. For initializing this class, a string containing the file location is required and executing the "DoWork" function in this class returns an array of entities

(line, arc and etc.). The c# code below describes the implementation and initialization of the import procedure:

6.1.2 Automatic Chain Detection

In most CAM software packages, inputting entities to the toolpath generator function is done manually. This process can be complicated and laborious for parts with various features. An automatic chain detection algorithm was developed that can find closed and open chains, which accelerates the toolpath generation process. In order to detect and store chains, ***chain*** and ***EntityA*** classes were created. As shown in Figure 6-2, the chain class entities, associated information can be stored that facilitates further analysis with detected chains.

EntityA class provides easy access to Entity's properties without casting to its inherited classes (Line, Circle and etc.). Additionally as depicted in Figure 6-3 , the ***EntityA*** class stores some parameters (*Feed*, *Cutting* and *Fastmove*) for the motion program generator module.

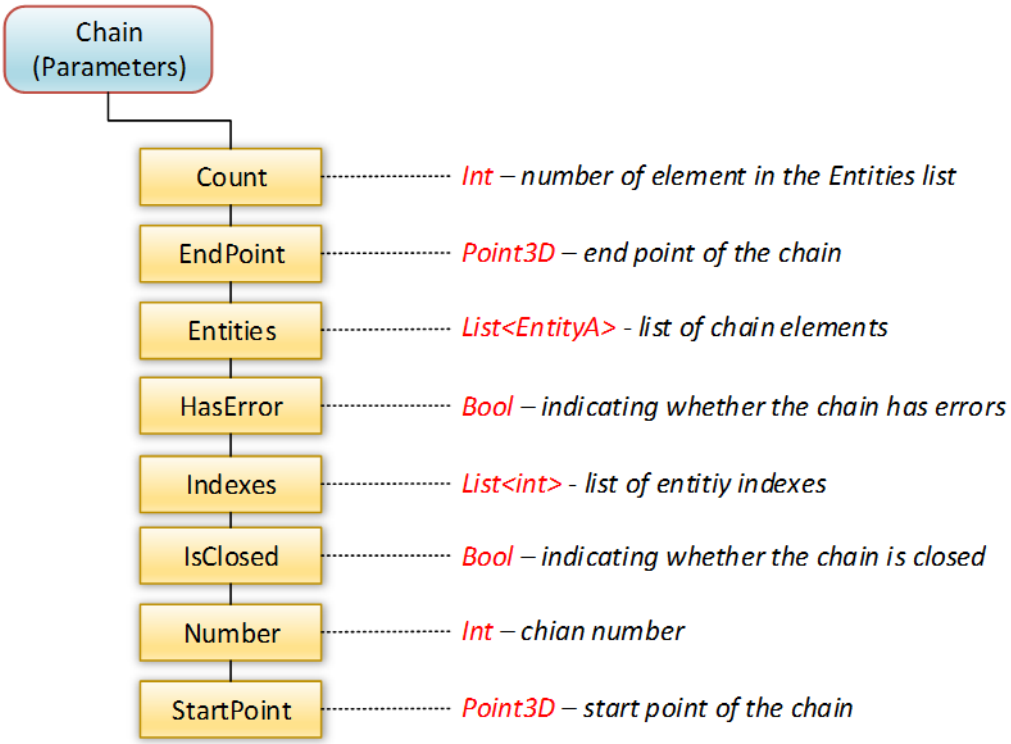


Figure 6-2 : Chain Class diagram

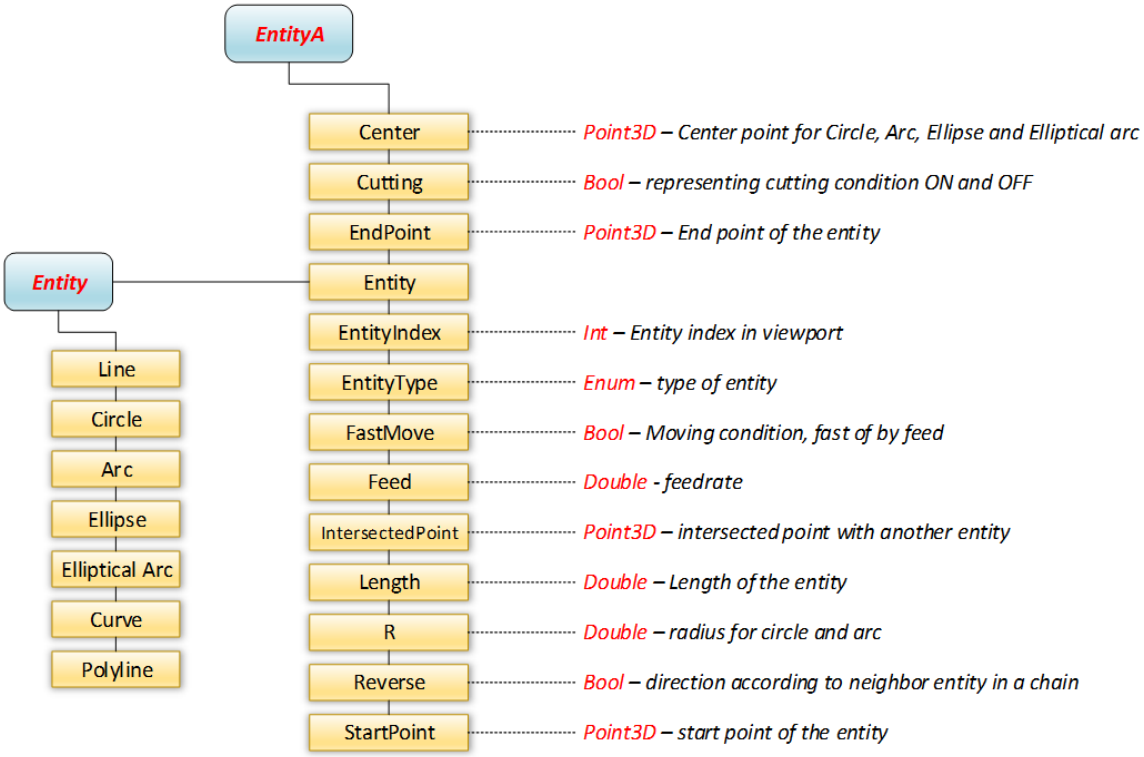


Figure 6-3 : EntityA class diagram

An automatic chain detection algorithm was developed that can find closed and open chains, which accelerates the toolpath generation process. The automatic chain detection function analyses all of the imported entities by comparing start and end points of each entity with created chains recursively. It means that the first chain is created and a new entity is added to the created chain. Thereafter, all start and end entities are compared with start and end chain and if one point equals the start or end point of the chain, the corresponded entity will be added to the chain and by calling the Sort function, the chain parameters will be updated. Continuing this procedure recursively, all chains can be found automatically. The chain detection flowchart in Figure 6-4 and the chain sort flowchart in Figure 6-5 describe the chain detection algorithm. For more details about the function please look at the *source code 2* and *source code 3*.

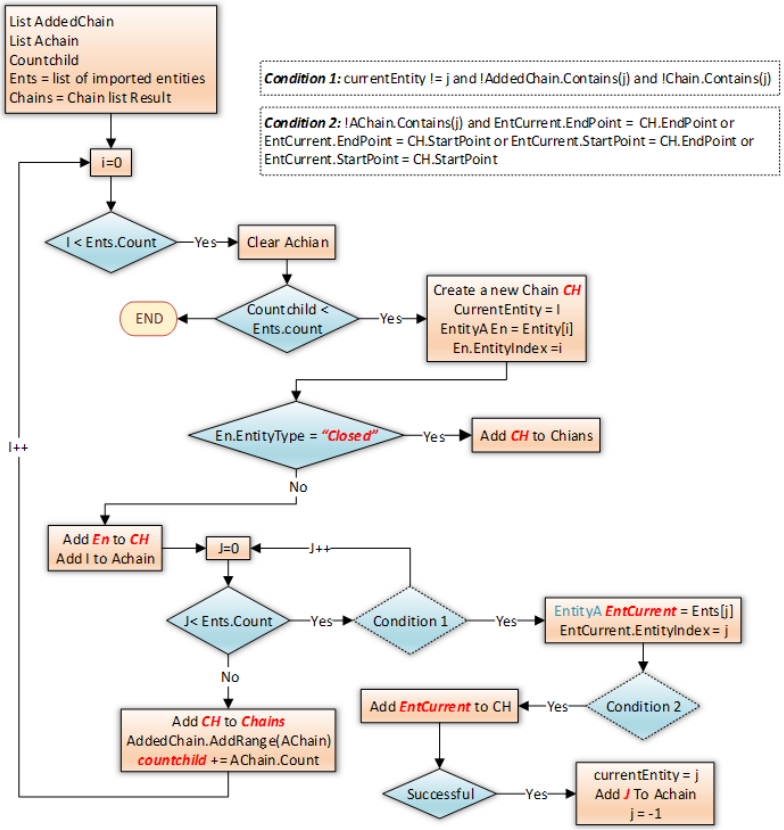


Figure 6-4 : Chain detection flowchart

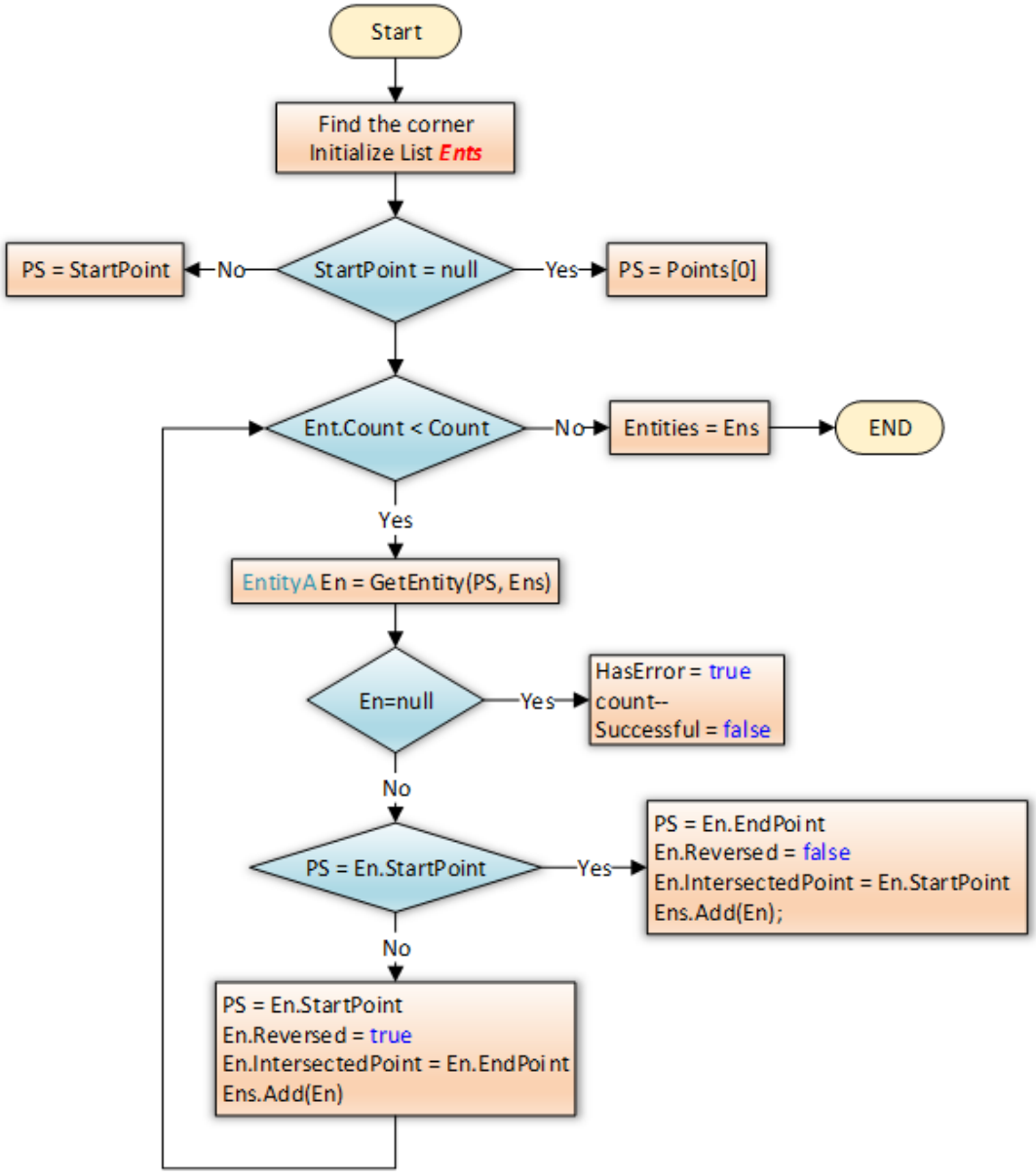


Figure 6-5 : Chain sort flowchart

6.1.3 Toolpath Generation

The developed toolpath generator core was equipped with the ability to create the following types of toolpaths for laser machining:

- 2D contour
- Spiral (Archimedean and square)

- Helix
- Zigzag (in different plane)
- Cone
- Planar section

ContourToolPath, *ToolPathHelix*, *ToolPathSpiral*, *ToolPathCone*, *ToolPathZigZag* and *PlanarSectionToolPath* classes were created and inherited from the ToolPathAll class as shown in Figure 6-6. ToolPathAll class provides an easy way for modification and calculation for other toolpath types. As shown in Figure 6-7 ToolPathAll class contains parameters and functions to generate different types of toolpaths.

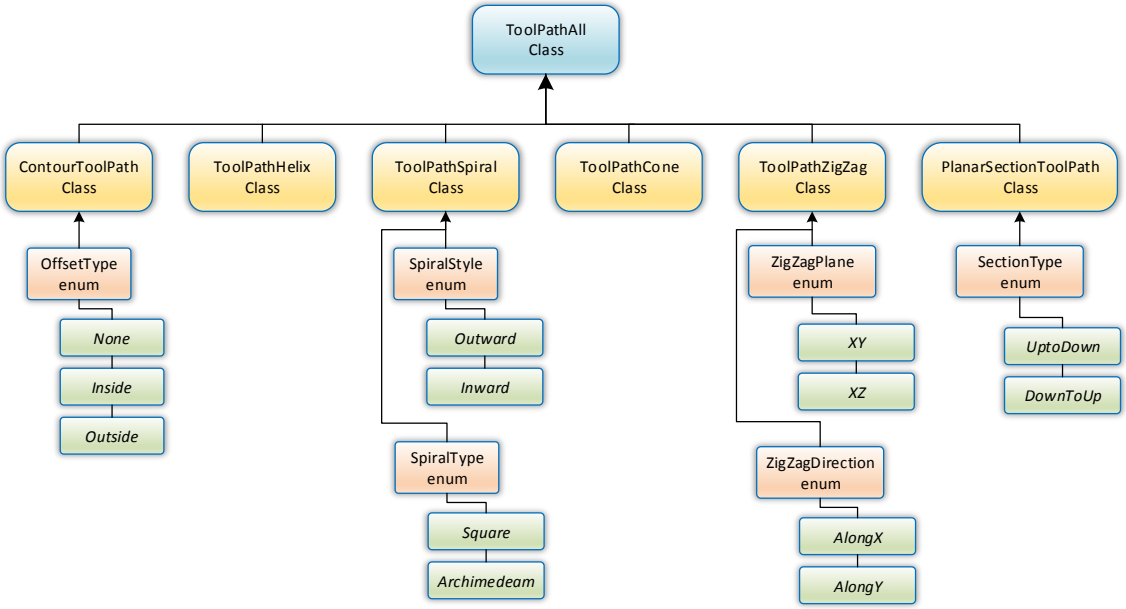


Figure 6-6 : ToolPathAll class diagram

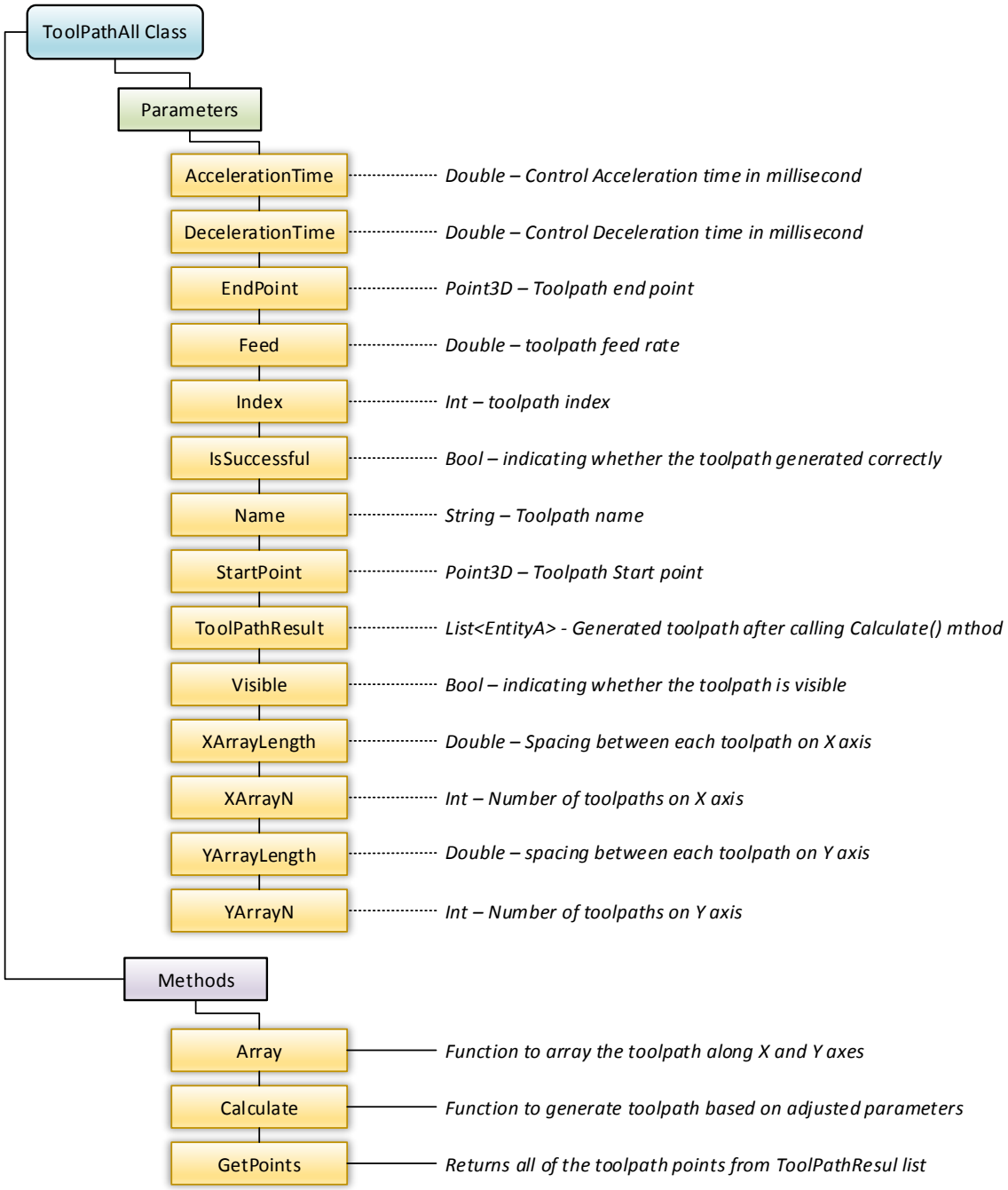


Figure 6-7 : ToolPathAll class diagram

6.1.3.1 Contour Toolpath

One of the requirements in laser micromachining is to keep the feed rate at a constant value along the entire toolpath. To reach the desired feed rate, controllers implement

different types of velocity profiles that have an acceleration time (t_A) and deceleration time (t_D). In order to start machining from a certain point with a feedrate of V, a 3-axis stage should start accelerating from a distance of at least ½ VT_a, and start decelerating at a distance of at least ½ VT_D to stop at a desired point. This will be a challenge when dealing with start points, end points, and corner points of non-tangent segments of intersection points. At these points the feed rate will not be constant as mentioned previously.

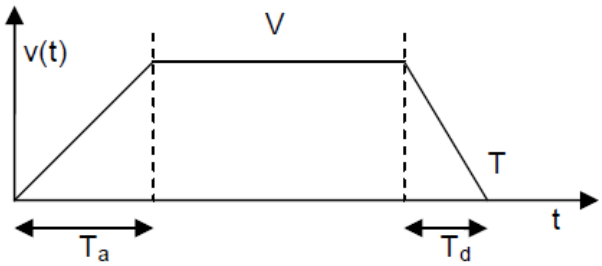


Figure 6-8 : Trapezoidal velocity profiles

To overcome this challenge, tangent circles and lines were introduced to create a continuous toolpath that will keep the feedrate constant at all times. Note that the laser will be turned off on tangent circles and lines and will be turned on again when toolpath reaches the toolpath segment.

For every non-tangent intersection point when the angle between the intersected segments is less than 45° as depicted in Figure 6-9, the toolpath is stretched from both incoming and ongoing toolpaths (L1 and L2) and an arc is defined tangent to these extensions (C1). It's worth noting that L1, L2 and C1 should be adjusted in a way that provides constant feedrate along the entire segments as well as minimizing the toolpath length. The minimum L1 length is found to minimize the total looping time.

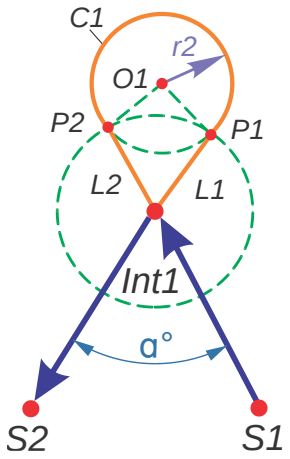


Figure 6-9 : Segment looping method 1

For every non-tangent intersection point where the angle between the intersected segments was more than 45° as illustrated in Figure 6-9, toolpath segments *S1* and *S2* were stretched from the intersected point *Int1*. Circle *C1* was created from point *Int1* and a radius of *r1*. The value of *r1* was found from equation 1:

$$r1 = (Max[TA, TD] \times Feed) / 1000 \quad (6.1)$$

Points *P1* and *P2* were found by intersection of *C1* and extended segments *S1* and *S2*. By connecting *Int1* to *P1*, and *P1* to *P2*, and *P2* to *Int1*, lines *L1*, *L2* and *L3* were formed, respectively. For creating a smooth trajectory during machining, sharp corners between *L1*-*L3* and *L3*-*L2* were filleted by a radius of L1/5.

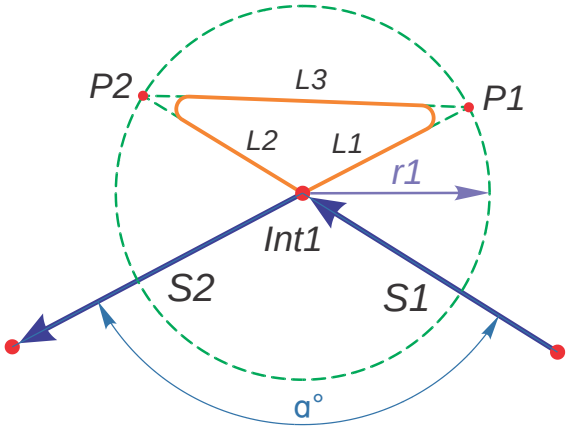


Figure 6-10 : Segment looping method 2

Adding loops at sharp corners kept the feedrate constant and resulted in better surface quality and machining uniformity. Other useful feature for 2D contouring was contour offsetting. Nowadays, most advanced controllers support radius toolpath compensation (G41 and G42) but using controller radius compensation (CRC) in laser machining would be very complicated because the laser shutter also must be controlled and for complicated geometries, using CRC may cause errors. Therefore, the toolpath offsetting feature was developed to do contour offsetting in the CAM environment. As shown in Figure 6-11 the contour offset can be outside of inside. A new chain offsetting method was developed that handles any kind of geometries.

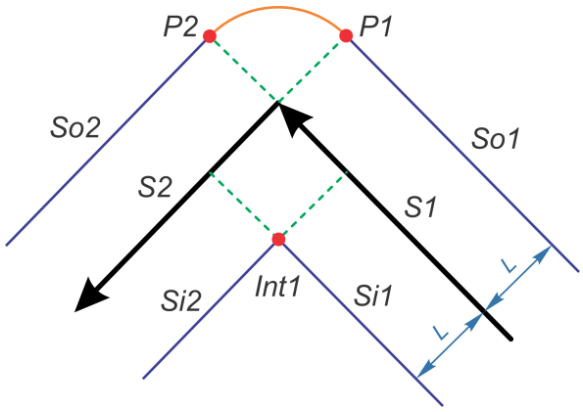


Figure 6-11 : Entity inside and outside offsetting

As depicted in Figure 6-11, after offsetting two curves, 3 cases could be encountered:

I. The intersection of two offsetted curves was the start point or end point;

II. The offsetted curves did not intersect an arc from *P1*, and *P2* had to be created to connect them.

III. The intersection point between two offsetted curves was neither the start nor the end point, *Si1* and *Si2* were trimmed and connected from point *Int1*.

For more details about the contour offsetting algorithm please refer to *source code 4*.

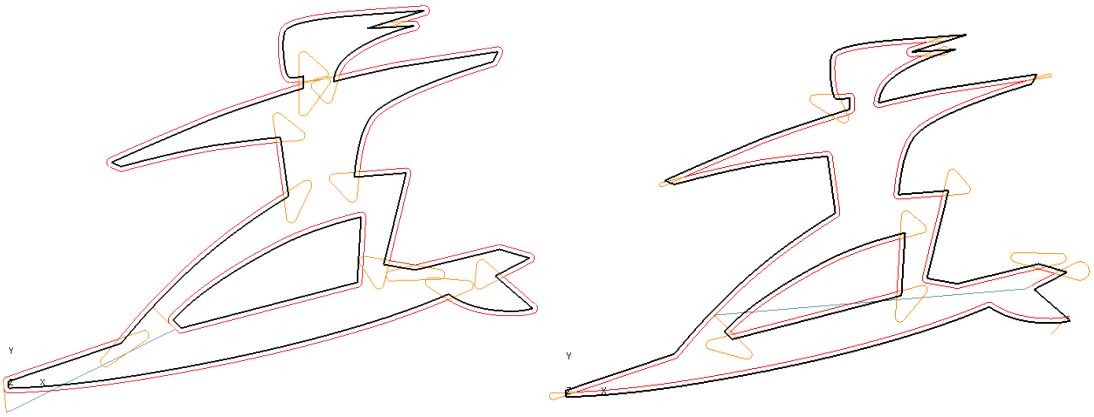


Figure 6-12 : Outside and inside 2D contour Toolpath

Additionally, the 2D contour strategy is capable of finding optimum feature orders to minimize machining time. Figure 6-13 shows an example of an optimum chain order based on minimum distance.

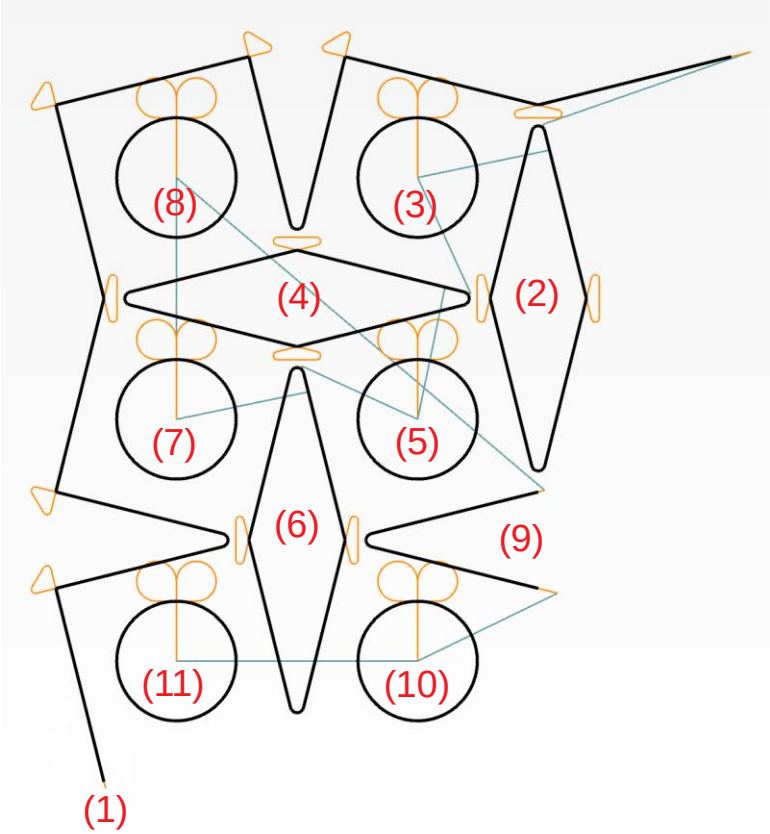


Figure 6-13 : Chain order based on minimum traveling distance

The *ContourToolPath* Class was created to store necessary parameters including a list of chains, offset length, offset type and multi layers configuration (Figure 6-14). For more detail about 2D contour toolpath generation, please look at the provided flowchart (Figure 6-15) and *source code 5*.

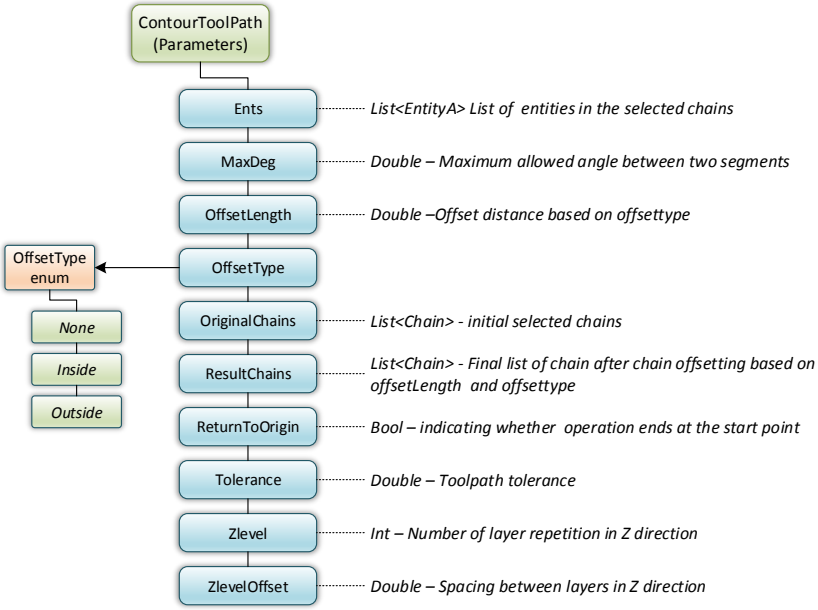


Figure 6-14 : Contour Toolpath class diagram

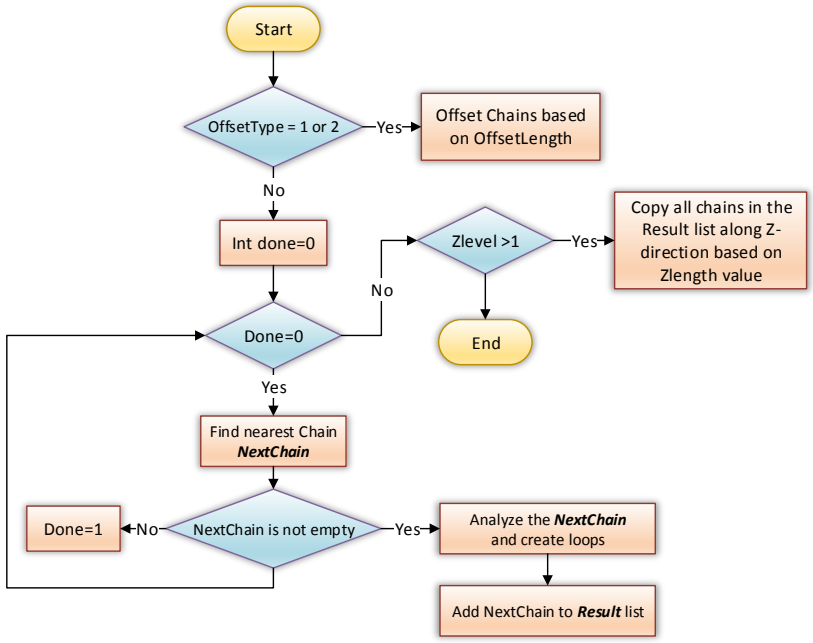


Figure 6-15 : Contour toolpath flowchart

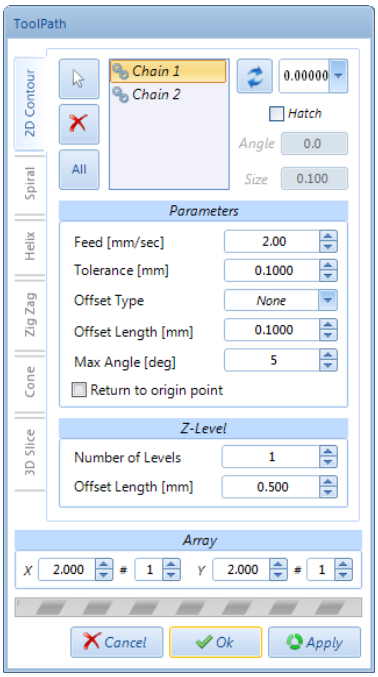


Figure 6-16 : 2D Contour toolpath GUI

6.1.3.2 Helical Toolpath

A helical toolpath in laser machining is an important feature that is used for making holes. Helical movement provides a smooth trajectory and keeps the feed rate constant during machining. Mathematically, a helix is a three-dimensional curve that can be described as:

$$r(t) = \big(x(t), y(t), z(t)\big) = (a\cos(t), a\sin(t), bt) \tag{6.2}$$

As shown in Figure 6-17, class *ToolPathHelix* contains parameters such as Pitch, R, startPoint, tolerance and turns that are necessary to draw a three dimensional helix (Figure 6-18).

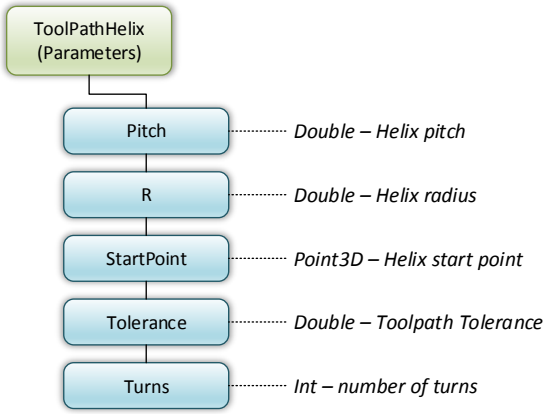


Figure 6-17 : ToolPathHelix class diagram

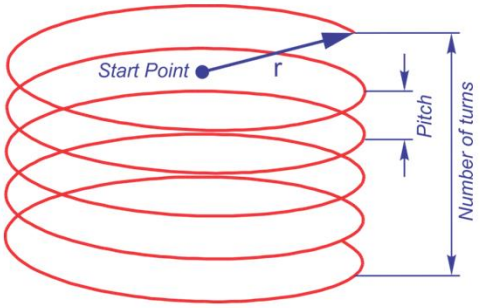


Figure 6-18 : Helical toolpath parameters

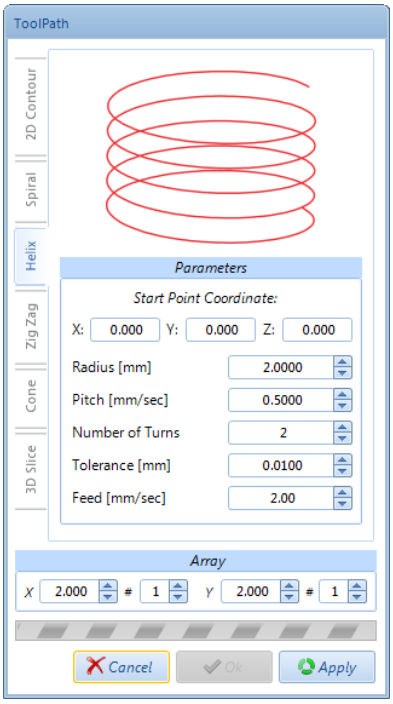


Figure 6-19 : Helix toolpath GUI

6.1.3.3 Spiral Toolpath

A spiral toolpath can be used to form circular shapes, square shapes, and surface scanning for laser machining. A *SpiralToolPath* class was developed that is capable of generating two types of spirals:

6.1.3.3.1 Square spiral

The developed algorithm automatically generates a square shape spiral based on step size and length values which provide a constant feed rate on the entire toolpath, A non-cutting loops are added at sharp edges as depicted in Figure 6-20. Please refer to *source code 7* for more details about the square toolpath generator algorithm.

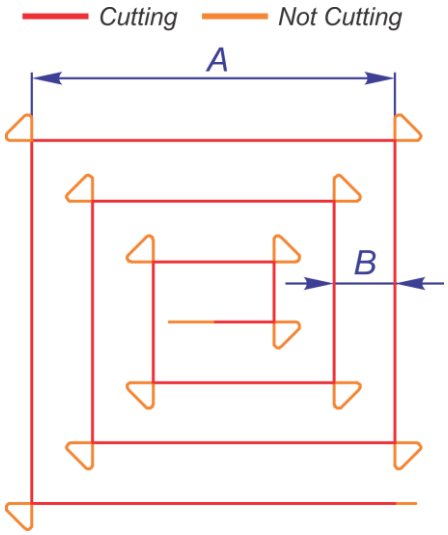


Figure 6-20 : Square spiral toolpath

6.1.3.3.2 Archimedean Spiral

In polar coordinates, the Archimedean spiral can be described by the equation:

$$r = a + b\theta \tag{6.3}$$

As illustrated in Figure 6-21, the Archimedean spiral toolpath is a combination of the Archimedean spiral and a full circle to form a closed, circular shape. *Source code 8* gives more details about the developed Archimedean spiral toolpath generator.

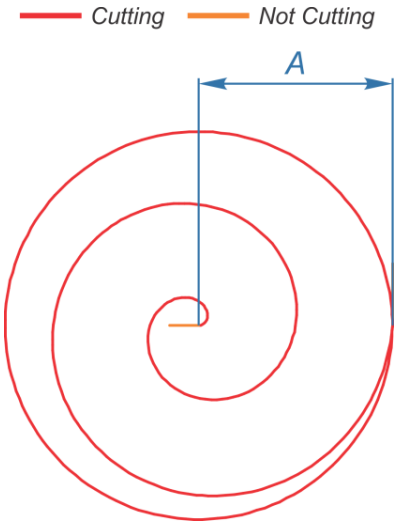


Figure 6-21 : Archimedean spiral toolpath

As shown in Figure 6-22 the *ToolPathSpiral* class stores parameters for both Square and Archimedean spiral toolpaths and, based on *SpiralType*, this class calculates the needed function for generating the toolpath.

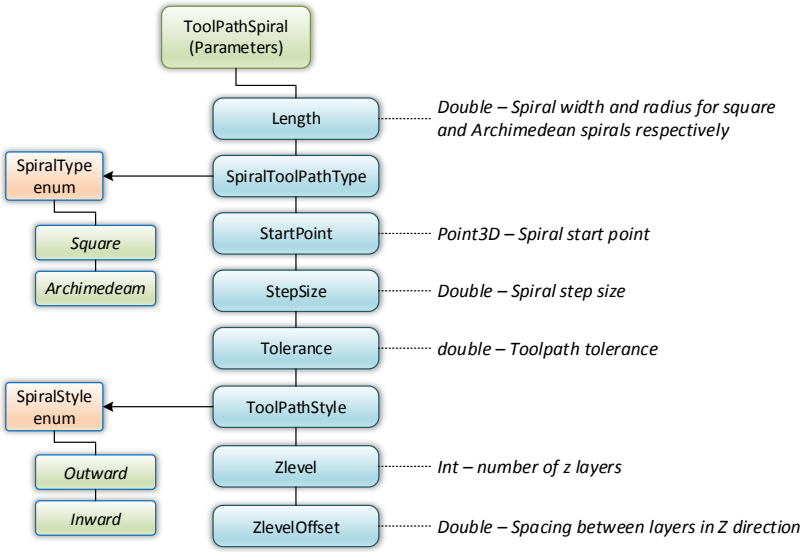


Figure 6-22 : SpiralToolPath class diagram

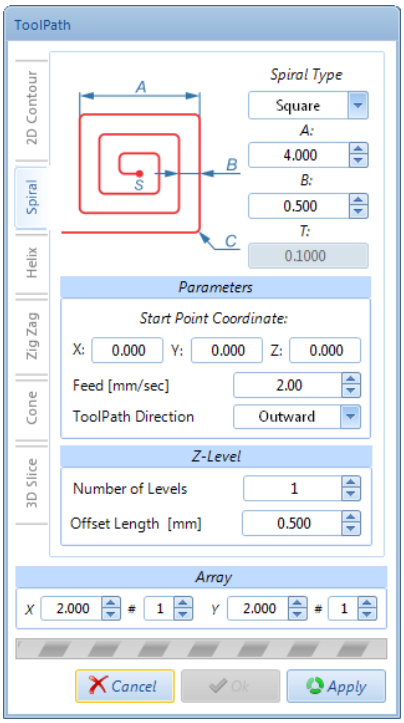


Figure 6-23 : Spiral toolpath GUI

6.1.3.4 Zig Zag Toolpath

The Zig zag toolpath is used for face machining. The class *ToolPathZigZag* that was developed is capable of generating different types of zigzag toolpaths. The Zig Zag toolpath can be along the X (Figure 6-24) and Y (Figure 6-25) axes and on different planes XY and XZ with variable parameters.

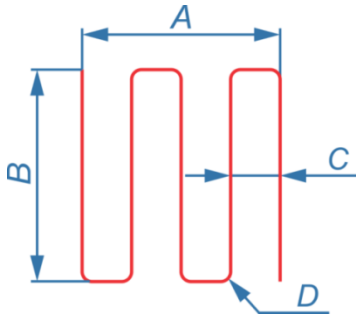


Figure 6-24 : Zig Zag toolpath along X

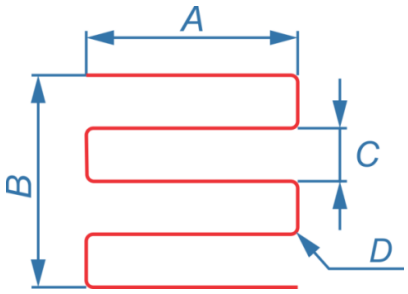


Figure 6-25 : Zig Zag Toolpath along Y

As shown in Figure 6-26, the *ToolPathZigZag* class stores parameters for different types of zigzag toolpaths. Based on *ZigZagDirection* and *ZigZagPlane* this class generates the required toolpath. *Source code 10* describes how to generate variable toolpaths from a function.

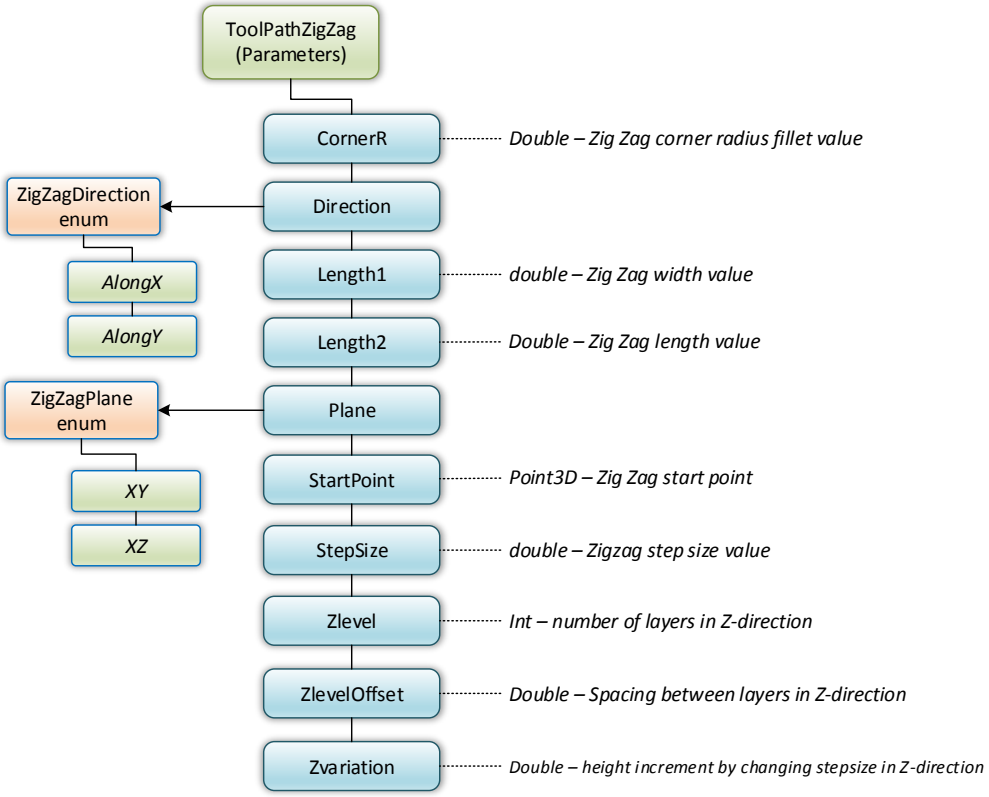


Figure 6-26 : ZigZagToolpath class diagram

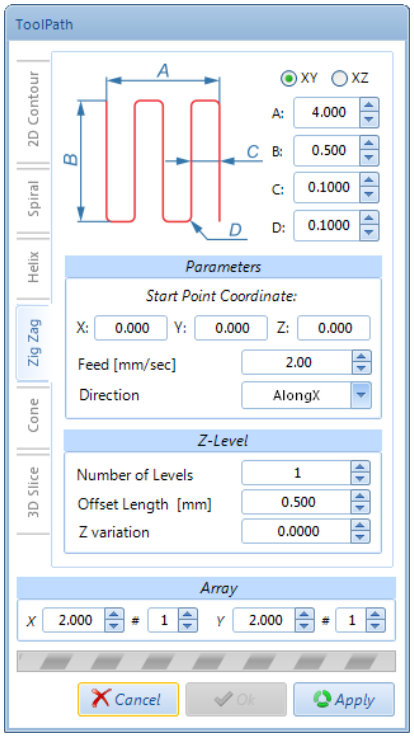


Figure 6-27 : Zigzag toolpath GUI

6.1.3.5 Cone Toolpath

The *ToolpathCone* provides a simple and automatic way to generate a toolpath for making cone shapes without drawing additional geometries. The cone toolpath is a combination of several spiral toolpaths with different diameters on different offset planes along the Z axis. The *ToolPathCone* class requires some parameters as shown in Figure 6-28. These parameters are: cone lower diameter (d1), cone upper diameter (d2), cone height (H), laser beam diameter (BeamD), number of passes (Zlevel), start point and toolpath tolerance. A cone can be created by d1, d2, and H values and thereafter, the cone angle (α) would be found based on class defined parameters by the equation below:

$$\alpha = \arctan\left(\frac{\left(\frac{d1-d2}{2}\right)}{H}\right) \tag{6.4}$$

A cone is a conic section based on pass number *Pn* and pass height value H_{Pn}:

$$H_{Pn} = H - \left(P_n \times \left(\frac{H}{Zlevel}\right)\right) \tag{6.5}$$

Depending on the Z-level value, the cone must be sectioned such that it forms circles with different diameters and height positions. For defining intersected circle diameters r_c the equation below is implemented:

$$r_C = d2 + 2 \times (H_{Pn} \times \tan(\alpha)) \tag{6.6}$$

Finally for each intersected circle a spiral toolpath based on the *rc* value is created.

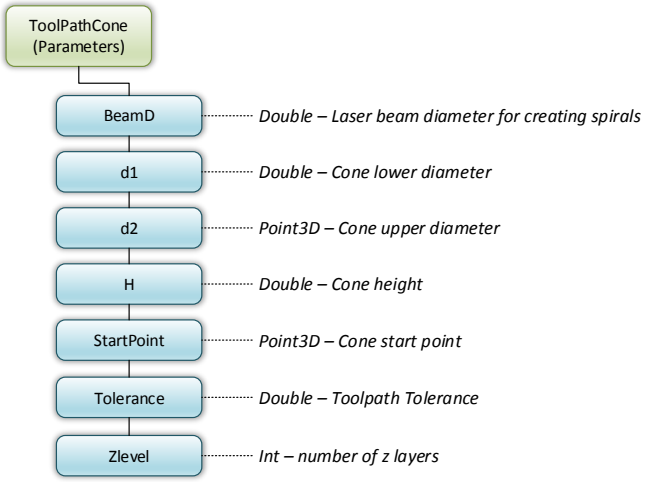


Figure 6-28 : ToolPathCone Class diagram

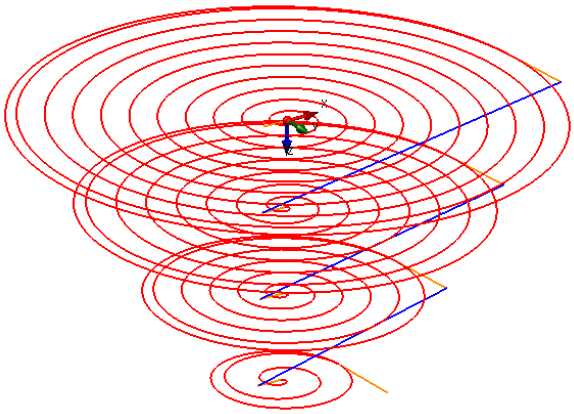


Figure 6-29 : Cone toolpath

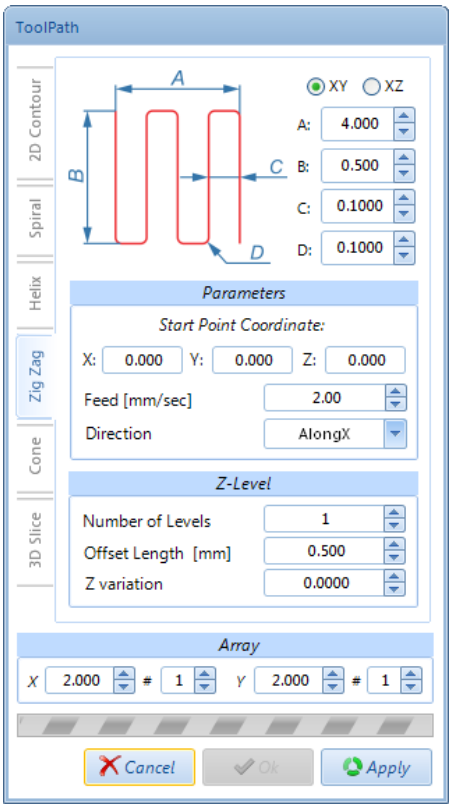


Figure 6-30 : ToolpathCone GUI

6.3.3.6 Planar Section Toolpath

The Planar section toolpath is used for 3D laser machining and 3D rapid prototyping. As shown in Figure 6-31 the *PlanarSectionToolPath* class that was developed requires a 3D model (Model), a step down size (ZOffset), a Laser beam radius, (R) and a maximum allowed angle between segments (MaxDeg). Initially, the 3D model is sectioned based on *SectionType* and *ZOffset* value. By using Pocket function of the *Region* class from the eyeshot library, each created section area machining toolpath is generated based on the laser beam radius (R). Finally, all of the separate toolpaths connect together to form a planar section toolpath. Figure 6-32 shows an example of a planar section toolpath for an imported STL model. For more information about creating planar sections and using region class functions refer to *source code 11*.

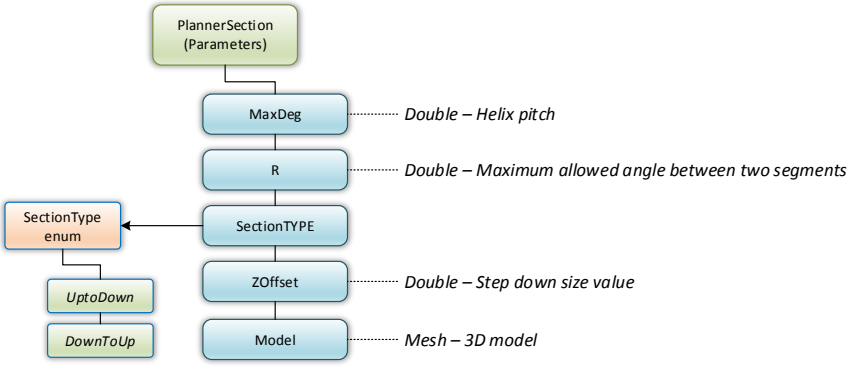


Figure 6-31 : Planarsectiontoolpath class diagram

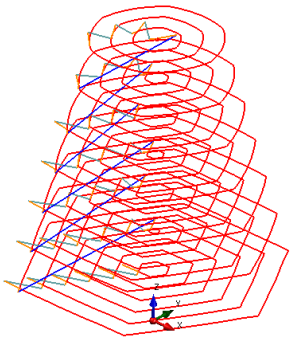


Figure 6-32 : Planar section toolpath

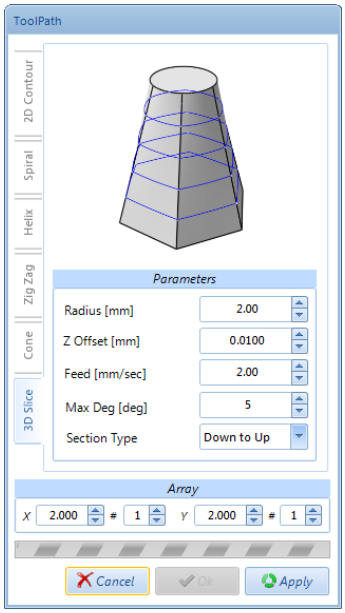


Figure 6-33 : Planar section toolpath GUI

6.1.4 Convert Curve to Arcs

In the traditional method for converting curves into a motion program (G-code program), curves are divided into sub-lines based on input tolerance, chosen so that the desired resolution can be retained. Depending on curvature values of the curve smaller sub lines have to be created. This approach results in more motion blocks and small feature segments, which leads to decreased feed rate and increased machining time. Therefore, another conversion method has to be implemented to decrease the number of G-code program lines and increase the motion segment lengths. A simple algorithm (please refer to Figure 6-34) was developed to convert curves into arcs by matching curve curvature radius values based on the input tolerance. For more details about the algorithm and programming implementation, please refer to *Source code 1*.

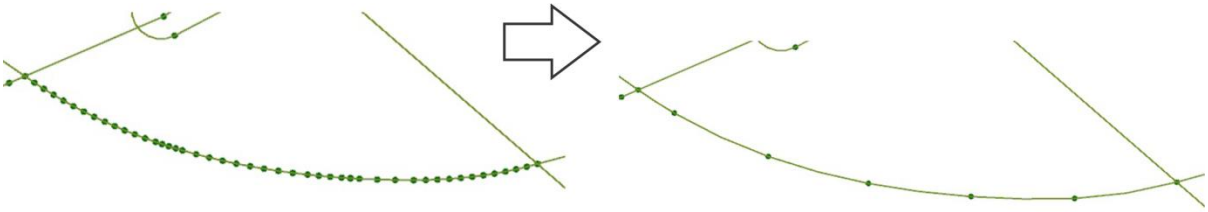


Figure 6-34 : Spline to arc Conversion example

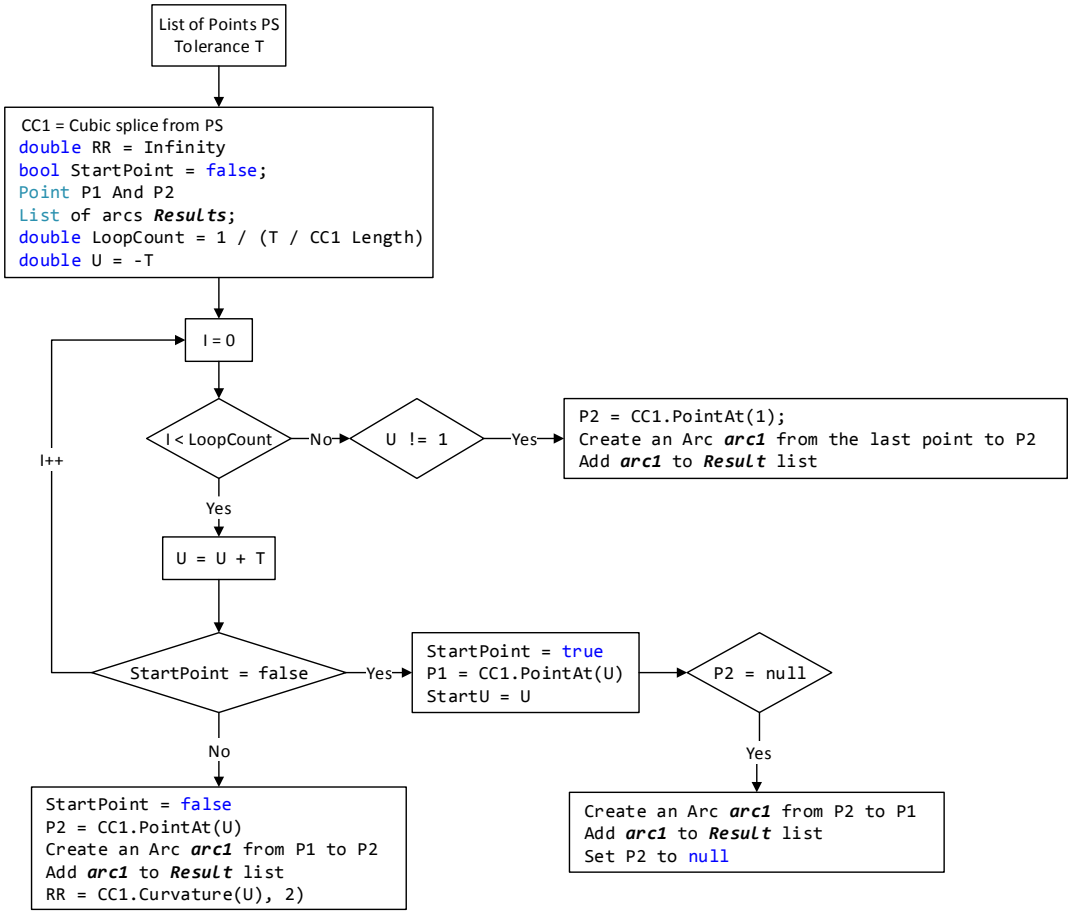


Figure 6-35 : Spline to arc algorithm

6.1.5 G-code Generator

The G-code generator function calculates the required tolerances and decimal digits based on feature size. The G-code generator library generates ISO format motion program from input values for linear, rapid and circular interpolation (please refer to *source code 13ок*).

6.2 Control Environment

The control environment described in Figure 6-36 was developed to achieve the following targets:

- CNC controller communication

- Laser shutter control
- Laser machining monitoring

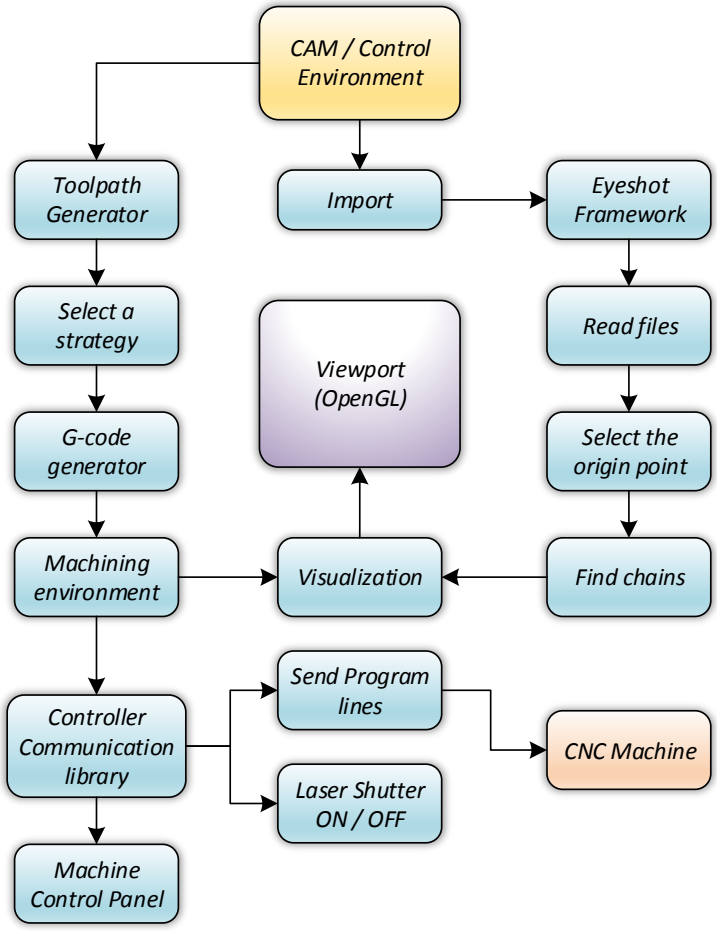


Figure 6-36 : Functional model of CAM and control environments

6.2.1 CNC Controller Communication

After generating the toolpath and creating a G-code program, the created program had to be sent to the controller. For this project, a Delta Tau controller model of the Geo Brick LV (Figure 6-37) was used. Therefore, the generated G-code must be compatible with the mentioned controller model. For this purpose, The CAM environment configuration provides a flexible G-code setup to set linear, rapid, and circular interpolation symbols. The next issue was to send and upload the created G-code file

without using the default software provided by the Delta Tau Company. Hence, the PMAC communication driver was implemented to use the controller's functionality in the .NET framework environment.

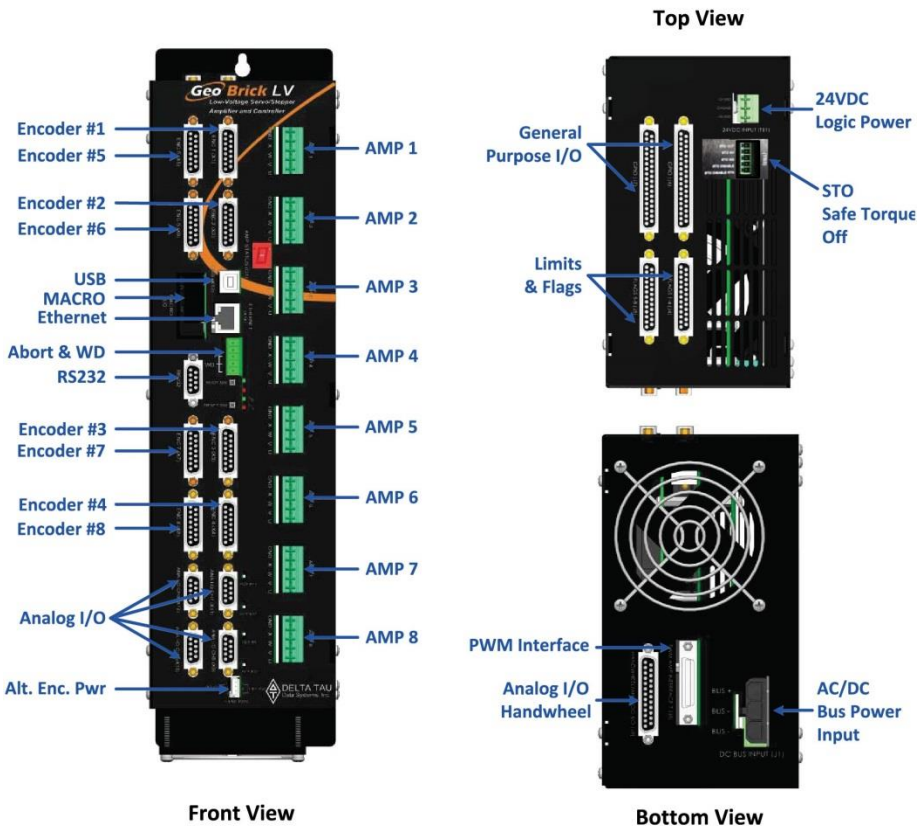


Figure 6-37 : Delta Tau Geo Brick LV controller scheme

The PMAC driver contained some useful functions and methods that provided a robust communication between PMAC type of controllers and a PC through USB, COM and Ethernet connections.

Monitoring Laser machining was done by moving a cone representing a laser beam as the axis moved. The real time G-code segment tracing function also showed the currently program line, the program segment, as well as the estimated machining time.

In cases when the generated G–code program was larger than the buffer size of the controller, the program was divided into subprograms. Each subprogram was uploaded in

turn after completing the previous subprogram. This method (G-code splitter) allowed machining of very complicated geometries with vast features for a relatively longer period of time. For more details about this method, please refer to *Source Code 14*.

6.2.2 Laser Shutter Control

Laser shutters can be controlled through either a serial port or an analog input signal. Controlling through an analog input signal was found to be the faster and more robust way to manipulate the shutter. Therefore, the G-code generator function added a necessary variable corresponding to the digital interface of the controller (the mentioned controller J6 interface was used), so that it could turn off and turn on the shutter. The G-code presented below shows an example in which M33 was turning off and on the laser shutter.

```
LINEAR X1.8183 Y5.455 Z0 F2 -> Linear interpolation
M33=0 P400=1   Shutter Turn OFF
Y6.3977
P400=2  Current executive Program Line
CIRCLE1 X1.861 Y6.355 I0.0427 J0 -> CC Circular interpolation
P400=3
LINEAR X1.8183
P400=4
CIRCLE2 I0 J-0.9 -> CCW Circular Interpolation
M33=1 P400=5   Shutter Turn ON
LINEAR X1.7756
M33=0 P400=6
CIRCLE1 X1.8183 Y6.3977 J0.0427
P400=7
LINEAR Y5.455
P400=8
M30
%
```

Figure 6-38 : Generated Motion program sample

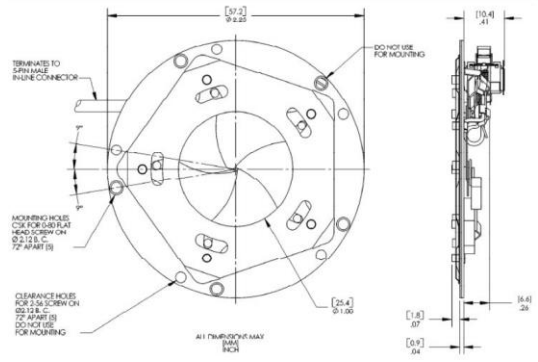
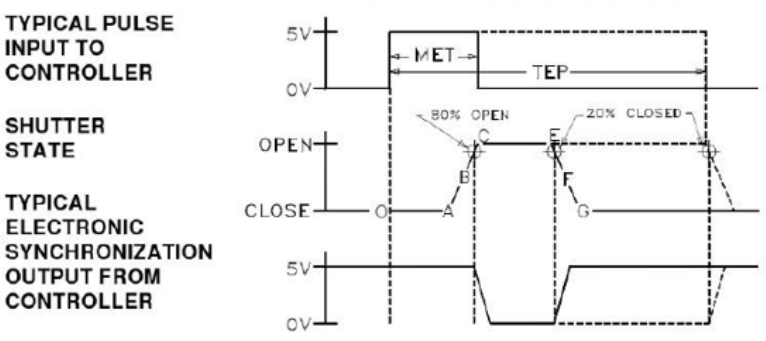


Figure 6-39 : Laser shutter layout



Timing in msec

O-A Delay time on opening after current is applied	3.0
A-C Transfer time on opening	9.0
O-C Total opening time	12.0
B-F Min equivalent exp. time	17.5
C-E Min dwell time with min input pulse	6.0
E-G transfer time on closing	14.0
A-G total window time	29.0
MET: Min exposure time	15.0
TEP: Typical exposure pulse	>15.0

Figure 6-40 : Timing of pulse input and output relative to shutter state

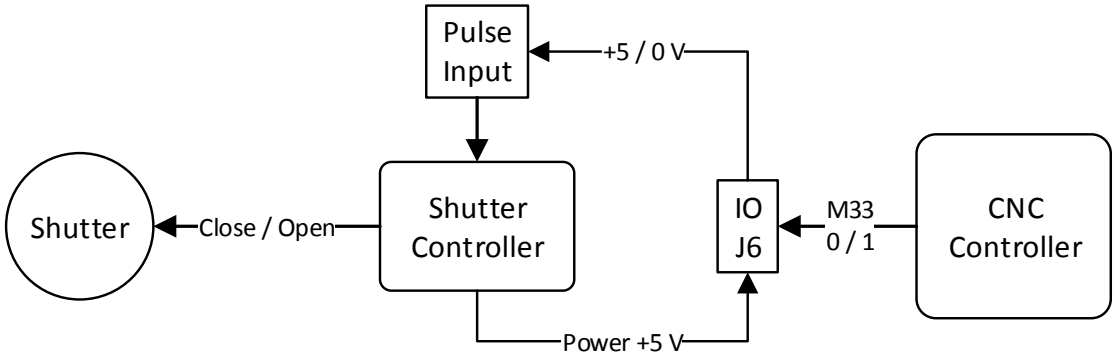


Figure 6-41 : Laser shutter connection scheme

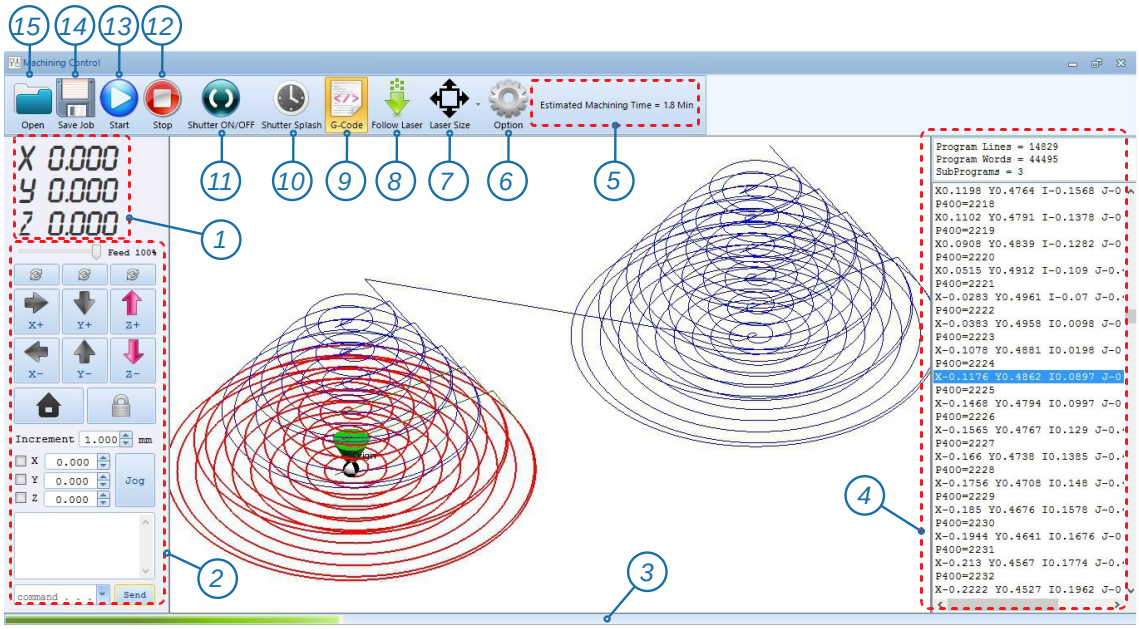


Figure 6-42 : Control environment interface of the software

1) Position panel
2) CNC control panel
3) G-code program progress bar
4) G-code Panel
5) Estimated machining time
6) Controller options
7) Laser beam size
8) Follow/unfollow laser beam
9) G-code panel show/hide
10) Set shutter timer
11) Laser shutter status (On / OFF)
12) Stop controller progress
13) Upload and run a G-code program
14) Save project
15) Load project

6.3 CAD Environment

The CAD environment of the software provides some useful tools to create and modify 2D geometries. The description below takes the form of a user manual.

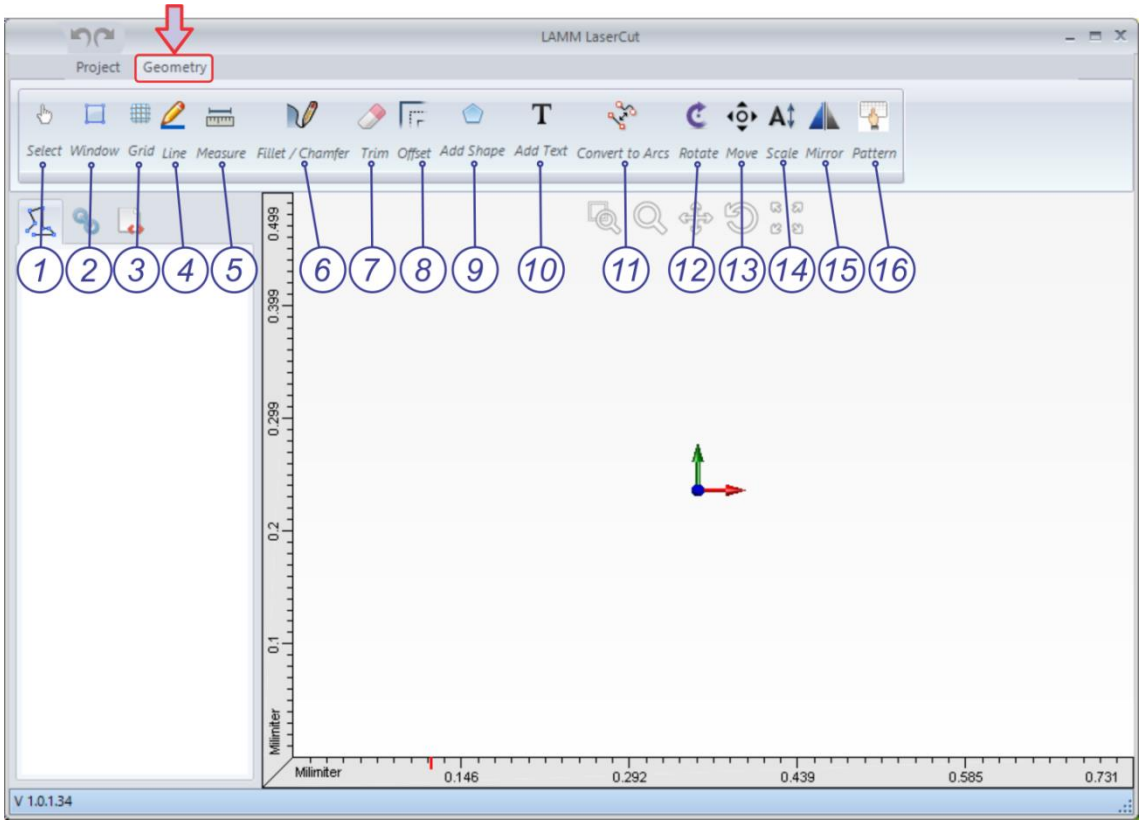


Figure 6-43 : CAD Environment user interface

1) ***Select*** - Default. Allows selecting drawing entities and clicking and dragging certain points. Double clicking on an entity opens its settings dialog box.

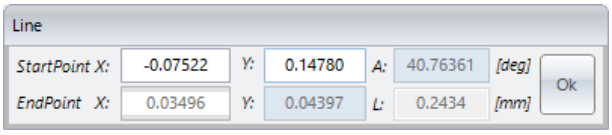


Figure 6-44 : Line setting dialog box

2) ***Window*** – allows selecting multiple drawing entities by dragging a window over them.
3) ***Grid*** - A grid can be added to the sketch view to allow for easy and precise drawing. To make the grid visible and reactive, the user checks the "visible" box. The user can set the size of the grid using the Max/Min X and Y inputs or use the

Auto Size function to have the grid expand or contract dynamically as the user draws. The step size can be set using the "Step" input box, or Auto Step can be turned on to have size adjust during drawing.

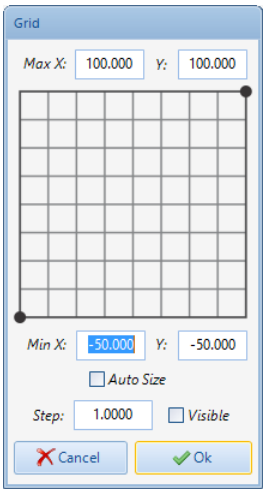


Figure 6-45 : Grid setup dialog box

4) ***Line*** - Draw a line. Click once to place the start point and click again to place the end point. Join lines by placing the start of one line on the end of a previously placed line. It is also possible to input the line parameters directly by using the line properties box.
5) ***Measure*** - Measures the distance between two points. Click once to select the first point and again to select the second.
6) ***Fillet/Chamfer*** - Allows selecting two connected drawing entities and replacing the intersection with a fillet or chamfer of a specified radius.

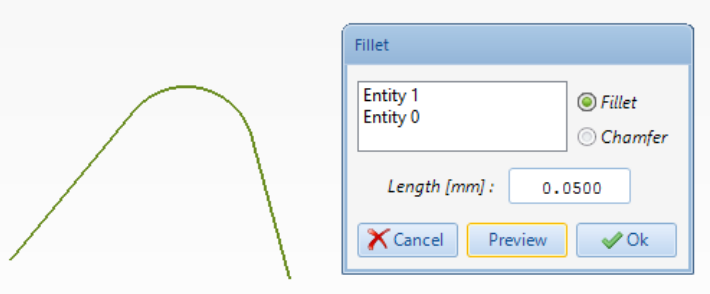


Figure 6-46 : Fillet / chamfer dialog box

7) ***Trim*** - Erases a segment of a line or shape between the two closest intersections
8) ***Offset*** - Allows user to create an offset version of selected drawing entities. Select sharp to have unrounded intersections.

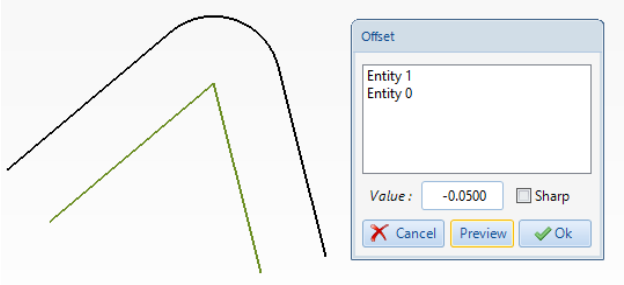


Figure 6-47 : Offset dialog box

9) ***Add shape*** - Add arcs, circles or ellipses to the sketch. A circle is defined by its center point and radius. An ellipse is defined by its center point and x and y radii.

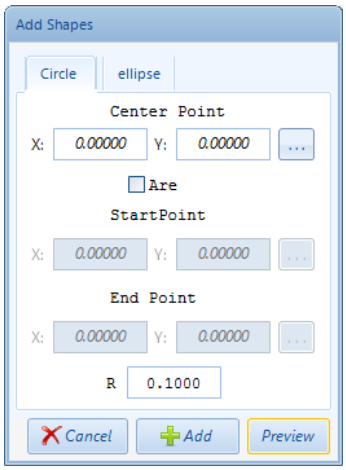


Figure 6-48 : Add shapes dialog box

10) ***Add Text*** - Create a drawing of any text input in any font. Choose the start point and height of the text. The tolerance corresponds to the smallest line segment that will be generated.

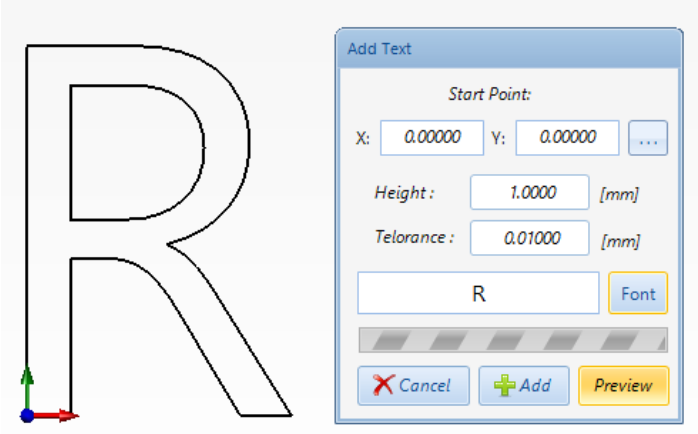


Figure 6-49 : Add text dialog box

11) ***Convert to Arcs*** - Convert the selected drawing entities into arcs with an input tolerance. Arcs are much more efficient than many small lines.

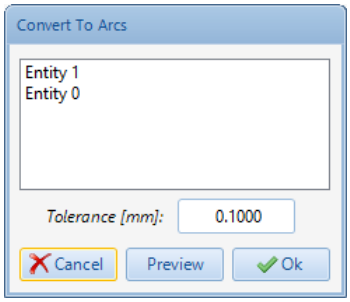


Figure 6-50 : Convert to arcs dialog box

12) ***Rotate*** - Rotates the chosen drawing entities around the origin by the specified amount.

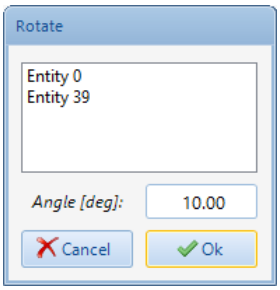


Figure 6-51 : Rotate dialog box

13) ***Move*** - Moves the selected drawing entities from their position relative to the 'from' point to the same position relative to the 'to' point.

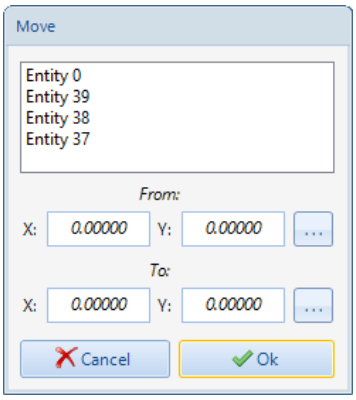


Figure 6-52 : Move dialog box

14) ***Scale*** - Scales the selected drawing entities by the specified factor.

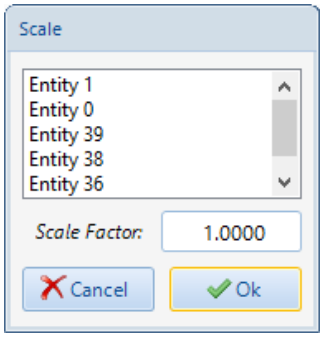


Figure 6-53 : Scale dialog box

15) ***Mirror*** - Mirrors the selected drawing entities about the X or Y axis.

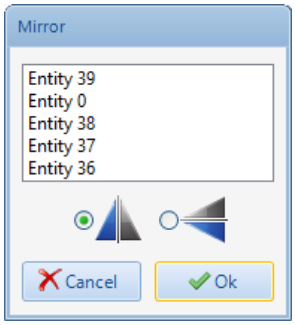


Figure 6-54 : Mirror dialog box

16) ***Pattern*** - Duplicates the selected entities in either a linear or circular pattern. For a linear pattern, choose the number of elements in each direction and their respective spacing. For a circular pattern, choose the point about which user would like to rotate and then the total angle of rotation and number of elements in the pattern. Check the 'Rotate entities' box to have the new entities turned with the rotation.

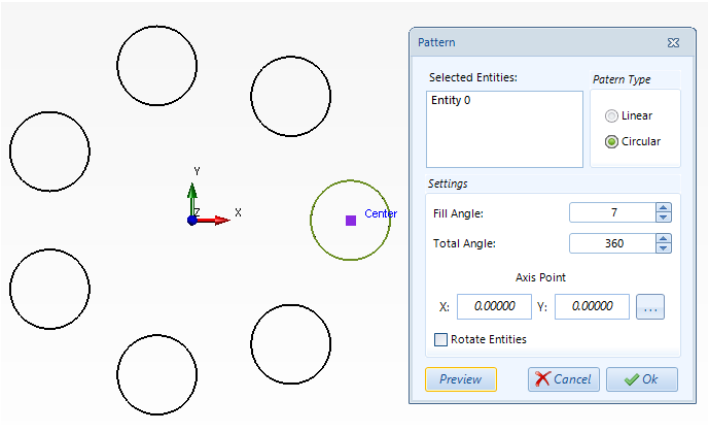


Figure 6-55 : Pattern Dialog box

6.4 Software Interface

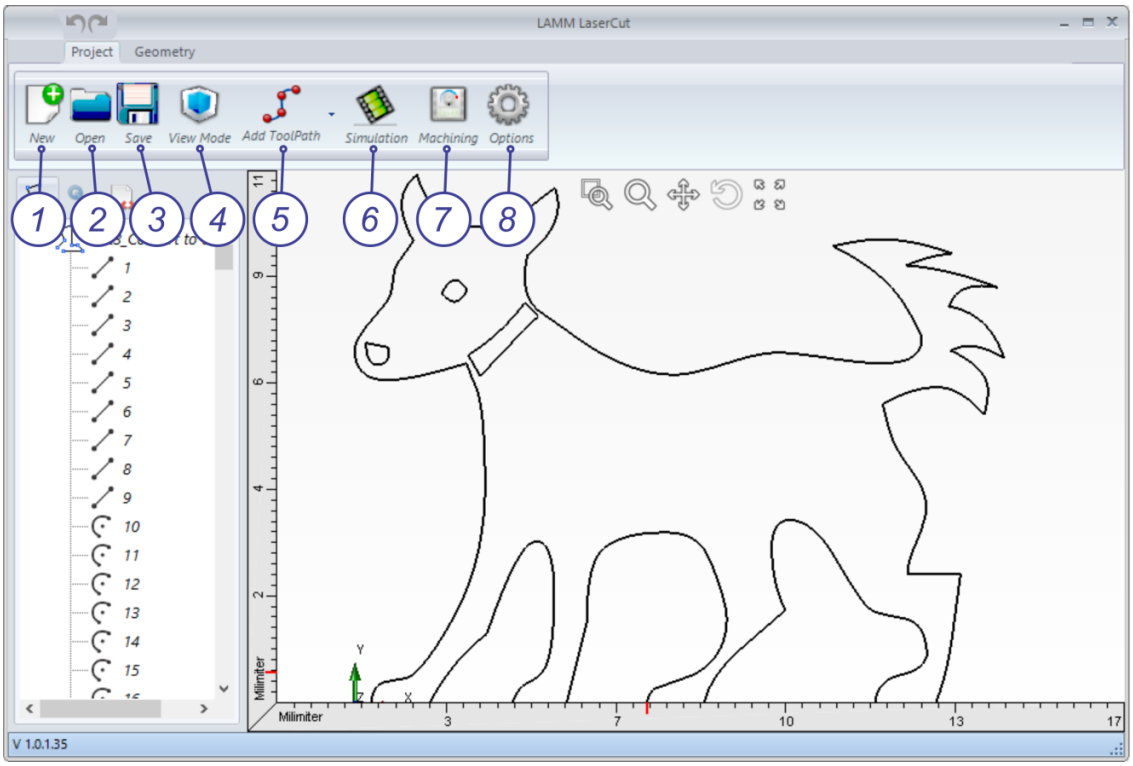


Figure 6-56 : laser machining user interface

1) ***New*** - Starts a new project
2) ***Open*** - Opens a .dxf or .dwg file into a 2D sketch
3) ***Save*** - Saves user's current sketch as a .dwg file
4) ***View Mode*** - Toggles between sketching view mode and 3D viewing mode
5) ***Add ToolPath*** - Creates a new Tool Path. Use 'Add Contour' to create a new tool path using selected chains from your current sketch. Select the chains by clicking the mouse pointer button and then choosing them from the left, or hit "all" to select all chains. Set the speed of the tool path with feed, in millimetres per second. The 'Tolerance' value determines the smallest line that will be converted into G-code and is used to remove unnecessary elements depending on project size. Offset type and length allow user to create an offset to account for the size of the laser cut. Set the number of levels and the level spacing in the Z-level section.

You can also create an array of the same tool path by selecting the spacing and number of elements in the x and y directions

6) ***Simulation*** **-** Opens the simulation screen where you can view a simulation of the current project
7) ***Machining*** **-** Opens the machining interface
8) ***Options*** **-** Adjust settings for G-code production and controller configuration

Chapter 7: Conclusion and Future Work

7.1 Future works

The software that was developed should be developed further with more features and improvements. The features below were planned to improve the functionality of the current software.

- ***Improve CAD environment to facilitate adding and modifying geometries***

  An importing DXF and DWG format files capability from other CAD software packages is very useful, but having CAD functionality together with an importing module would improve the efficiency of preparing a CAD model for generating toolpaths. Importing and editing back and forth from other CAD software can be very time consuming. Therefore, the software should have a simple CAD environment for adding and editing geometries. To improve the CAD module, dynamic Move, rotate, scale and mirror objects, dimensional constraint types for any geometrical objects should be added.

- ***Convert bitmap image into 2D and 3D point cloud***

  A bitmap image can be converted into a 2D or 3D point cloud for engraving.

- ***Automatic finding of cutting parameters based on workpiece material***

  Tuning laser machining parameters (such as depth and width of cut that are both function of laser power and feed for a specific sample) for a particular material is laborious and time consuming. For commonly used materials, a laser machining database could be incorporated in the software that will help to choose machining parameters for a given workpiece material. A graduate student could work to optimize laser machining parameters for wide varieties of materials and create the database for the software.

- ***Laser machining simulation for multi-axis machining***

Material removal simulation is one the useful features of CAM software packages that predicates the final result of machining before the operation. Adding laser simulation capability for the software will help users to look at the final result of machining after generating toolpaths and to check the generated toolpaths without doing actual machining.

- ***Ability to control galvanometer scan head for rapid 3D laser machining***

  3D printing is another useful feature that brings the ability to make 3D parts in micro scale. 3D model files (STL, IGES and STEP) can be imported and sectioned based on the layers height value. Then for each created section, an area machining toolpath is generated based on the laser beam radius. Hatching method also can be used for 3D rapid prototyping from a 2D profile. Generated motion program will be able to control galvanometer scan heads that increase the productivity of rapid prototyping operation.

- ***Ability to control multi axis laser machining***

  The evolution of the Hexapod 6 axis motion system is the Tripod integrated with nano precision stages and a rotary for 6 axes of True Nano precision. This platform due to basic physics can outperform Hexapod systems with Nano precision while increasing the work envelope and significantly increasing the stiffness of the structure, making this platform ideal for nano machining and thermal bonding applications where reactive forces are common. This hexapod stage will be provided by the ALIO Company to support motion program generation and control of these kinds of stages for the software developed.

- ***Development of postprocessor for different brands of controllers***

  The developed software communicates only with the Delta Tau controller and The CAM module can only generate motion programs for this type of controller. Communication and motion program generation for *AeroTech* and *XPS* controllers is in development plans.

7.2 Conclusions

In chapter 1: 2-Dimensional ploughing volume simulation software was developed for micro flat end milling. The presented real time simulation software is based on the dual-Dexel method. Simulations were performed to calculate the amount of ploughing for a given toolpath. For each simulation, different cutting parameters were used to investigate the ploughing process in micro milling. The simulated results were compared with measured resultant forces for verification of the model. The ploughing simulation software can be used to better understand the micro milling process. By estimating the amount of ploughing, the best cutting condition can be selected for a given toolpath, resulting in increased tool life and surface quality.

In chapter 2: Automated contact sensing software based on a rotating wire probe and an AE sensor was developed. The presented automated scanning algorithm works effectively on the X-Y plane. Automated scanning of calibration artifacts and machined features have been achieved. The system can be mounted on a conventional micro-milling machine to measure the component dimensions. Only a platform with embedded AE sensors, a spindle with the probe and a DAQ card are needed to set up a measurement system.

In Chapter 3: Feed rate optimization software has been implemented based on chip volume for micro milling. An algorithm has been developed to determine the optimum feed rate for each tool position using the chip volume at the corresponding position. After optimization, the resultant forces and the machining time decreased substantially. However, as the tool diameter and the size of the feature to be machined decrease, segmentation length along the tool path seems to play an important role in controlling the

feed rate during machining. The actual feed rate measured turned out to be lower than the specified feed rate after optimization, possibly due to increased segmentations. By keeping the segment length unchanged and calculating average chip volume values for each segment, the optimization resulted in decreasing the average cutting forces and matching time.

In Chapter 4: A novel computer-aided manufacturing (CAM) software system was proposed for laser ablation machining processes. The algorithms and prototype software system designed offers efficient optimization of the toolpath for controlled delivery of laser energy into a work-piece. The software simplifies part program creation and maintains constant velocity of the sample stage for each segment of a complex tool trajectory. These features enable efficient deposition of laser energy into the work piece and therefore, reduction in heat-affected zone is expected in laser ablation based micromachining. The developed software provides fast modification of the toolpath, automatic and efficient sequencing of path elements in a complicated tool trajectory, location of reference points, and automatic repair of geometrical errors in imported drawing exchange files (DXF) or DWG format files.

Bibliography

[1] J. R. Mayor and A. A. Sodemann, "Intelligent Tool-Path Segmentation for Improved Stability and Reduced Machining Time in Micromilling," *J. Manuf. Sci. Eng.,* vol. 130, no. 3, p. 13, 2008.

[2] M. Jun, S. Kapoor and R. DeVor, "Investigation of the Dynamics of Micro-End Milling, Part 2: Model Validation and Interpretation," *Journal of Manufacturing Science and Engineering,* vol. 128, pp. 901-912, 2006.

[3] C. Goo, M. Jun and A. Saito, "Probing System for Measurement of Micro-Scale Components," *SME Journal of Manufacturing Processes,* vol. 14, pp. 174-180, 2012.

[4] M. Atsushi and I. Soichi, "Monitoring and Control of Cutting Forces in Machining Processes: A Review," *Int. J. of Automation Technology,* vol. 3, pp. 445-456, 2009.

[5] Y. Altintas, "Prediction of cutting forces and tool breakage in milling from feed drive current measurement," *J. of Engineering for Industry,* vol. 114, pp. 386-392, 1992.

[6] J. M. Lee, D. K. Choi, J. Kim and C. N. Chu, "Real-time tool breakage monitoring for NC milling process," *CIRP Annals – Manufacturing,* vol. 44, no. 1, pp. 59-62, 1995.

[7] A. Matsubara, "Current Status and Trends of Monitoring and Control Technology in Machining Process," *J. of the Society of Instrument and Control Engineers,* vol. 41, pp. 781-786, 2002.

[8] K. Furutani, "Piezoelectric sensors," *J. of the Society of Instrument and Control Engineers,* vol. 45, pp. 296-301, 2006.

[9] "Kistler," [Online]. Available: http://www.kistler.com.

[10] K. Jemielniak, "Commercial Tool Condition Monitoring Systems," *J. of Advanced Manufacturing Technology,* vol. 15, pp. 711-721, 1999.

[11] J. Tlusty and G. C. Andrews, "A critical review of sensor for unmanned machining," *Annals of the CIRP,* vol. 32, no. 2, pp. 563-572, 1983.

[12] "ARTIS," [Online]. Available: http://www.artis.de.

[13] J. H. Kim, H. K. Chang, D. C. Han and D. Y. Jang, "Cutting Force Estimation by Measuring Spindle Displacement in Milling Process," *CIRP Annals – Manufacturing Technology,* vol. 54, no. 1, pp. 67-70, 2005.

[14] G. B. Jeong, D. H. Kim and D. Y. Jang, "Real time monitoring and diagnosis system development in turning through measuring a roundness error based on three-point method," *Int. J. of Machine Tools and Manufacture,* vol. 45, pp. 1494-1503, 2005.

[15] "Yamazaki Mazak Corporation," [Online]. Available: http://www.mazak.jp.

[16] A. A. D. Sarhan, A. Matsubara, M. Sugihara, H. Saraie, S. Ibaraki and Y. Kakino, "Monitoring Method of Cutting Force by Using AdditionalSpindle Sensors," *JSME Int. J.,* vol. 49, pp. 307-315, 2006.

[17] E. Bagci, "Monitoring and analysis of MRR-based feedrate optimization approach and effects of cutting conditions using acoustic sound pressure level in free-form

surface milling," *Scientific Research and Essays,* vol. 6(2), pp. 256-277, 2011.

[18] S. Doruk Merdol and Y. Altintas, "Virtual Simulation and Optimization of Milling Applications—Part II: Optimization and Feedrate Scheduling," *Journal of Manufacturing Science and Engineering,* vol. 130, pp. 051005-1, 2008.

[19] J. Donggo, K. Kwangsoo and J. Jungmin, "Voxel-Based Virtual Multi-Axis Machining," *The international journal of Advanced Manufacturing Technology,* vol. 16, p. 709–713, 2000.

[20] W. Wang, "Solid Modeling for Opfimi g Metal Removal of Three-dimensional N C End Milling," *Journal of Manufacturing Systems,* vol. 7, pp. 57-65, 1988.

[21] S. Doruk Merdol and A. Yusuf, "Virtual cutting and optimization of three-axis milling processes," *International Journal of Machine Tools & Manufacture,* vol. 48, p. 1063– 1071, 2008.

[22] T. Bailey, M. A. Elbestawi, T. I. El-Wardany and P. Fitzpatrick, "Generic Simulation Approach for Multi-Axis Machining, Part 2: Model Calibration and Feed Rate Scheduling," *JOURNAL OF MANUFACTURING SCIENCE AND ENGINEERING,* vol. 124, pp. 634-642, 2002.

[23] B. Guzel and I. Lazoglu, "Increasing productivity in sculpture surface machining via off-line piecewise variable feedrate scheduling based on the force system model," *International Journal of Machine Tools & Manufacture,* vol. 44, pp. 21-28, 2004.

[24] L. Zhang, J. Feng and Y. Wang, "Feedrate scheduling strategy for free-form surface machining through an integrated geometric and mechanistic model," *Int J Adv Manuf Technol,* vol. 40, p. 1191–1201, 2009.

[25] A. Anderson, "Detecting and eliminating collisions in NC machining," *Comput-Aided Des,* vol. 10, pp. 231-237, 1978.

[26] W. Wang and K. Wang, "Real time verification of multiaxis NC programs with raster graphics," in *Proceedings of IEEE,International Conference on Robotics and Automation*, San Francisco, 1986.

[27] I. Chappel, "The use of vectors to simulate material removed by numerically controlled milling," *Comput-Aided Des,* vol. 15, pp. 165-158, 1983.

[28] V. Sungertekin and H. Voelcker, "Graphical simulation and automatic verification of NC machining programmes," in *International Conference on Robotics and Automation*, 1986.

[29] Y. Aitintas and A. Spence, "End Milling Force Algorithms for CAD Systems," *CIRP Annals - Manufacturing Technology,* vol. 40, no. 1, p. 31–34, 1991.

[30] A. Sullivan, H. Erdim, N. P. Ronald and S. F. Frisken, "High accuracy NC milling simulation using composite adaptively sampled distance fields," *Computer-Aided Design,* vol. 44, pp. 522-536, 2012.

[31] D. Roth, F. Ismail and S. Bedi, "Mechanistic modeling of the milling process using an adaptive depth buffer," *Comput-Aided Des,* vol. 30, pp. 1-17, 2003.

[32] "BSpline BRep Objects," [Online]. Available: http://www.dynoinsight.com/help/Geom/Objects/BRep/BSplines/BSplineBRep.aspx .

[33] W. Wang and K. Wang, "Geometric modeling for swept volume of moving solids,"

IEEE Computer Graphics and Applications, vol. 6, pp. 8-17, 1986.

[34] S. K and W. KK., "Five-axis swept volumes for graphic NC simulation and verification," in *Proceedings of the ASME design automation conference*, 1989.

[35] H. Yunching and O. J. H., "NC milling error assessment and tool path correction," in *proceedings of the 21st annual conference oncomputer graphics and interactive techniques*, New York (NY, USA), 1994.

[36] H. KC., "Solid sweeping in image space-application in NCsimulation," *The Visual Computer ,* vol. 10, p. 306–316, 1994.

[37] M. Heinrich, S. Tobias, S. Marc, A. Frank and W. Klaus, "Online sculpting and visualization of multi-dexel volumes," in *Proceedings of the eighth ACM symposium on solid modeling and applications*, New York (NY, USA), 2003.

[38] M. Inui, M. Kaneda and R. Kakio, "Fast Simulation of Sculptured Surface Milling with 3-Axis NC Machine," *The International Federation for Information Processing,* vol. 18, pp. 97-108, 1999.

[39] K. Yasumasa, I. Kumiko, I. Tomotoshi, N. Shiro and E. Kazuhiko, "A flexible quantitative method for NC machining verification using a space-division based solid model," *The Visual Computer,* vol. 7, pp. 2-3, 1991.

[40] D. Jang, K. Kim and J. Jung, "Voxel-Based Virtual Multi-Axis Machining," *Int J Adv Manuf Technol,* vol. 16, p. 709–713, 2000.

[41] K. Karunakaran and R. Shringi, "A solid model-based off-line adaptive controller for feed rate scheduling for milling process," *journal of materials processing technology,* vol. 20, p. 384–396, 2008.

[42] "FORMTEC Engineering Services GmbH," [Online]. Available: www.formtec.de.

[43] S. Sreeram, A. S. Kumar, M. Rahman and M. T.Zaman, "Optimization of cutting parameters in micro end milling operations under dry cutting conditions using genetic algorithms," *Int J Adv Manuf Technol,* vol. 30, p. 1030–1039, 2006.

[44] D. S., P. D. T., I. A. and P. K., "Micromilling strategies: Optimization issues," *Journal of Engineering Manufacture,* pp. 731-736, 2004.

[45] Özel, L. X. and D. A., "Modelling and Simulation of Micro-Milling Process," in *4th International Conference and Exhibition on Design and Production of Machines and Dies/Molds*, CESME, TURKEY, 2007.

[46] T. Oiwal and H. Nishitani, "Three-dimensional touch probe using three fibre optic displacement sensors," *Journal of Meas. Sci. Technol.,* vol. 15, pp. 84-90, 2004.

[47] H. Ji, H. Y. Hsu, L. X. Kong and A. B. Wedding, "Development of a contact probe incorporating a Bragg grating strain sensor for nano coordinate measuring machine," *Journal of Meas. Sci. Technol.,* vol. 20, no. 095304, p. 7, 2009.

[48] H. Butefisch, U. Brand, S. Buttgenbach and G. Wilkening, "Tactile metrology for active microsystems," *Journal of Microsyst Technol.,* vol. 14, p. 1933–1939, 2008.

[49] T. Liebrich and W. Knapp, "New concept of a 3D-probing system for micro-components," *Journal of CIRP Annals - Manufacturing Technology,* vol. 59, p. 513–516, 2010.

[50] K. Hidaka and P. Schellekens, "Study of a Small-sized Ultrasonic Probe," *CIRP Annals-Manufacturing Technology,* vol. 55, no. 1, pp. 567-570, 2006.

[51] M. Malekiana, S. S. Parka and M. Jun, "Modeling of dynamic micro-milling cutting forces," *International Journal of Machine Tools and Manufacture,* vol. 49, no. 7-8, p. 586–598, 2009.

[52] X. Liu, R. E. DeVor and S. G. Kapoor, "An Analytical Model for the Prediction of Minimum Chip Thickness in Micromachining," *J. Manuf. Sci. Eng.*, vol. 128, no. 2, pp. 474-481, 2005.

[53] P. M. Vogler, E. R. DeVor and G. S. Kapoor, "On the Modeling and Analysis of Machining Performance in Micro-Endmilling, Part II: Cutting Force Prediction," *ASME J. Manuf. Sci. Eng.*, vol. 126, no. 4, p. 695–705, 2004.

[54] K. Lee and A. D. Dornfeld, "An Experimental Study on Burr Formation in Micro Milling Aluminum and Copper," *Trans. NAMRI/SME,* vol. 30, p. 255–262, 2002.

[55] P. M. Vogler, E. R. DeVor and G. S. Kapoor, "On the Modeling and Analysis of Machining Performance in Micro-Endmilling, Part I: Surface Generation," *ASME J. Manuf. Sci. Eng.*, vol. 126, no. 4, p. 685–694, 2004.

[56] H. Weule, V. Huntrup and H. Tritschle, "Micro-Cutting of Steel to Meet New Requirements in Miniaturization," *CIRP Ann.*, vol. 50, p. 61–64., 2001.

[57] G. Bissacco, H. Hansen and J. Slunsky, "Modelling the cutting edge radius size effect for force prediction in micro milling," *CIRP Annals - Manufacturing Technology,* vol. 57, p. 113–116, 2008.

[58] R. R. Gattass and E. Mazur, "Femtosecond laser micromachining in transparent materials," *Nature Photonics,* vol. 2, pp. 219 - 225, 2008.

[59] T. Chong, M. Hong and L. Shi, "Laser precision engineering: from microfabrication to nanoprocessing," *Laser & Photonics Reviews,* vol. 4, no. 1, p. 123–143, 2010.

[60] A. Gillner, J. Holtkamp, C. Hartmann, A. Olowinsky, J. Gedicke, K. Klages, L. Bosse and A. Bayer, "Laser application in microtechnology," *J. of Materials Processing Technology,* vol. 167, pp. 494-498, 2005.

[61] S. I. Böhlen, J. Fieret, A. Holmes and W. K. Lee, "CAD/CAM software for an industrial laser manufacturing tool," *Proc. of SPI,* vol. 4977, pp. 198-206, 2003.

[62] E. Mutapcic, E. lovenitti and E. Harvey, "A 3D Visualization Tool for Excimer Laser Micromachining," *Proc. of SPIE,* vol. 4236, pp. 230-241, 2001.

[63] F. M. Chen, P. Y. Chen and T. W. Hsiao, "Tool path generation for 2D profiles in CO2 laser machine system," *International Technology and Innovation Conference,* pp. 1764-1770, 2006.

[64] C. S. Park, "Tool-path generation for Z- constant contour machining," *Computer-Aided Design,* vol. 35, pp. 27-36, 2003.

[65] Z. Li, Z. Zhang, L. Zheng and A. Dhanorker, "Feedrate optimization for variant milling process based on cutting force prediction," *Int J Adv Manuf Technol,* vol. 24, p. 541–552, 2004.

Appendix

Source code 1

```
//create and initialize a new ReadAutodesk class from a selected path in the file dialogbox
ReadAutodesk Read = new ReadAutodesk(openFileDialog1.FileName);
//call Dowork to start reading
Viewport1.DoWork(Read);
//Make sure that array in not empty
if (Read.Entities.Length > 0)
{
        //create and initialize a new Importcore class to detect chains
        DWG = new ImportCore(Read.Entities);
}
```

Source code 2

```
  private void GetChains()
        {
            // delete all the elements in the Chains List
            Chains.Clear();
            // initialize new lists
            List<int> AddedChain = new List<int>();
            List<int> AChain = new List<int>();
            // defind chain member counter
            int countchild = 0;
            bool Successful;
            // loop for reading imported entities from Ents list
            for (int i = 0; i < Ents.Count; i++)
            {
                // reset Achian list
                AChain.Clear();
                // check if all entities are defined by a chain
                if (countchild < Ents.Count)
                {
                    // check exsitence of current entity in AddedChain list
                    if (!AddedChain.Contains(i))
                    {
                        // create a new chain
                        Chain Chain = new LaserCAM.Chain(Chains.Count + 1);
                        int currentEntity = i;
                        // defind a new EntityA from current entity
                        EntityA Ent = new EntityA(Ents[i]);
                        Ent.EntityIndex = i;
                        // if entity if circle or is closed
                        if (Ent.EntityType == EntityType.Circle || Ent.IsClosed)
                        {
                            // create a chain from current entity and add it to chians
list
                            Chain.Add(Ent, out Successful);
                            Chains.Add(Chain);
                        }
                        // if current entity is not closed
                        else if (Ent.EntityType != EntityType.NotAllowed)
                        {
                            //add current entity to the created chian
                            Chain.Add(Ent, out Successful);
                            AChain.Add(i);
                            // another loop to anlyze imported entities
                            for (int j = 0; j < Ents.Count; j++)
                            {
                                // if the current entity is undefined
                                if (currentEntity != j && !AddedChain.Contains(j) &&
!Chain.Contains(j))
                                {
                                    // create a new EntityA from current entity
                                    EntityA EntCurrent = new EntityA(Ents[j]);
                                    EntCurrent.EntityIndex = j;
```

```
                                    // check if the current entity is not a circle
                                    if (!AChain.Contains(j) && EntCurrent.EntityType !=
EntityType.Circle && EntCurrent.EntityType != EntityType.NotAllowed)
                                    {
                                         // check for intrsected point between created
chain and current entity
                                        if (PointCheck(EntCurrent.EndPoint,
Chain.EndPoint) ||         PointCheck(EntCurrent.EndPoint, Chain.StartPoint) ||
                                            PointCheck(EntCurrent.StartPoint,
Chain.EndPoint) || PointCheck(EntCurrent.StartPoint, Chain.StartPoint))
                                        {
                                            // add current entity to the chain
                                            Chain.Add(EntCurrent, out Successful);
                                            // if it was successful
                                            if (Successful)
                                            {
                                                currentEntity = j;
                                                AChain.Add(j);
                                                j = -1;
                                            }
                                        }
                                    }
                                }
                            }
                            // add chain to the main Chain list
                            Chains.Add(Chain);
                            // add added entity indexes
                            AddedChain.AddRange(AChain);
                            // update chain element counter
                            countchild += AChain.Count;
                        }
                    }
                }
                else
                    break;
            }
        }
    }
```

Source code 3

```
    private void Sort(bool findcornners, out bool Successful)
        {
            Successful = true;
            // find the start point
            if (findcornners)
                FindCorners();

            Point3D PS;
            if (StartPoint != null)
                PS = StartPoint;
            else
                PS = Points[0];
            List<EntityA> Ens = new List<EntityA>();
            // get number of entities in the chain
            int count = Entities_.Count;

            while (Ens.Count < count)
            {
                EntityA En = GetEntity(PS, Ens);

                if (En == null)
                {
                    HasError = true;
                    count--;
                    Successful = false;
                }
                // define entity direction
                else if (ImportCore.PointCheck(PS, En.StartPoint))
                {
```

```
                    PS = En.EndPoint;
                    En.Reversed = false;
                    En.IntersectedPoint = En.StartPoint;
                    Ens.Add(En);
                }
                else
                {
                    PS = En.StartPoint;
                    En.Reversed = true;
                    En.IntersectedPoint = En.EndPoint;
                    Ens.Add(En);
                }

            }

            //find the chian end point
            if (Ens[Ens.Count - 1].Reversed)
                EndPoint_ = Ens[Ens.Count - 1].StartPoint;
            else
                EndPoint_ = Ens[Ens.Count - 1].EndPoint;

            Entities_ = Ens;
            // update indexes
            Indexes_.Clear();
            foreach (EntityA en in Entities_)
                Indexes_.Add(en.EntityIndex);
        }
```

Source code 4

```
private Chain OffsetChain(Chain Input, double r)
        {
            try
            {
                List<ICurve> ens = new List<ICurve>();
                Point3D CPoint;
                Point3D PPoint;
                Point3D SPoint;
                EntityA CEnt;
                EntityA PEnt = null;
                List<ICurve[]> R_C;
                ICurve C_Offset;
                List<int> index;
                int INDEX;
                bool IsOrientedClockwise = Input.IsOrientedClockwise();

                if (!Input.IsCircle)
                {
                    for (int i = 0; i < Input.Count; i++)
                    {
                        CEnt = Input.Entities[i];

                        C_Offset = OffsetACurve(CEnt, r, OffsetType, IsOrientedClockwise);

                        if (i > 0 && ens.Count > 0)
                        {
                            PEnt = Input.Entities[i - 1];

                            if (CEnt.Reversed)
                            {
                                CPoint = C_Offset.EndPoint;
                                SPoint = CEnt.EndPoint;
                            }
                            else
                            {
                                CPoint = C_Offset.StartPoint;
                                SPoint = CEnt.StartPoint;
                            }

                            if (PEnt.Reversed)
                                PPoint = ens[ens.Count - 1].StartPoint;
                            else
                                PPoint = ens[ens.Count - 1].EndPoint;
```

```
                            if (ImportCore.PointCheck(PPoint, CPoint))
                            {
                                ens.Add(C_Offset);
                            }
                            else
                            {
                                if (CheckCurveIntersection(ens, C_Offset, PPoint, CPoint, out R_C, out
index))
                                {
                                    for (int k = 0; k < R_C.Count; k++)
                                    {
                                        if (index[k] == ens.Count - 1)
                                        {
                                            ens[index[k]] = R_C[k][0];
                                            ens.Add(R_C[k][1]);
                                        }
                                        else
                                        {
                                            if (index[k] > 0)
                                            {
                                                ens.RemoveRange(index[k] + 1, ens.Count - 1 - index[k]);
                                                i--;
                                                break;
                                            }
                                        }
                                    }
                                }
                                else
                                {
                                    Arc arc1 = new Arc(Plane.XY, SPoint, r, PPoint, CPoint, false);
                                    Arc arc2 = new Arc(Plane.XY, SPoint, r, PPoint, CPoint, true);

                                    Arc Rarc;
                                    if (arc1.Length() > arc2.Length())
                                        Rarc = arc2;
                                    else
                                        Rarc = arc1;

                                    if (!CheckCurveIntersectionWithAll(Rarc, Input.Entities, out INDEX))
                                    {
                                        ens.Add(Rarc);
                                        ens.Add(C_Offset);
                                    }
                                }
                            }
                        }
                        else
                            ens.Add(C_Offset);
                    }
                }
                else
                {
                    CEnt = Input.Entities[0];
                    ICurve C = (ICurve)CEnt.Entity;
                    if (OffsetType == LaserCAM.OffsetType.Outside)
                    {
                        if (CEnt.Reversed)
                            C_Offset = C.Offset(-r, new Vector3D(0, 0, 1), 0.0001, false);
                        else
                            C_Offset = C.Offset(r, new Vector3D(0, 0, 1), 0.0001, false);
                    }
                    else
                    {
                        if (CEnt.Reversed)
                            C_Offset = C.Offset(r, new Vector3D(0, 0, 1), 0.0001, false);
                        else
                            C_Offset = C.Offset(-r, new Vector3D(0, 0, 1), 0.0001, false);
                    }
                    ens.Add(C_Offset);
                }

                //check the last entity
                if (!Input.IsCircle && Input.Count > 1)
                {
                    CEnt = Input.Entities[Input.Count - 1];

                    PEnt = Input.Entities[0];
```

```
                    C_Offset = ens[ens.Count - 1];

                    if (CEnt.Reversed)
                        CPoint = C_Offset.StartPoint;
                    else
                        CPoint = C_Offset.EndPoint;

                    if (PEnt.Reversed)
                    {
                        PPoint = ens[0].EndPoint;
                        SPoint = PEnt.EndPoint;
                    }
                    else
                    {
                        PPoint = ens[0].StartPoint;
                        SPoint = PEnt.StartPoint;
                    }

                    if (CheckCurveIntersection(ens, C_Offset, PPoint, CPoint, out R_C, out index))
                    {
                        ens[ens.Count - 1] = R_C[0][0];
                        ens[0] = R_C[0][1];
                    }
                    else if (!ImportCore.PointCheck(PPoint, CPoint) && Input.IsClosed)
                    {
                        Arc arc1 = new Arc(Plane.XY, SPoint, r, CPoint, PPoint, false);
                        Arc arc2 = new Arc(Plane.XY, SPoint, r, CPoint, PPoint, true);
                        if (arc1.Length() > arc2.Length())
                            ens.Add(arc2);
                        else
                            ens.Add(arc1);
                    }
                }

                Chain NewChain = new Chain(ens.Count);
                Entity[] ents = ens.Cast<Entity>().ToArray();
                EntityA[] entAs = new EntityA[ents.Length];
                bool done;
                for (int i = 0; i < ents.Length; i++)
                {
                    EntityA Ena = new EntityA(ents[i]);
                    Ena.EntityIndex = i;
                    entAs[i] = Ena;
                }

                NewChain.AddRange(entAs, out done);
                NewChain.Number = Input.Number;
                //if (done)
                return NewChain;
            }
            catch
            {
                throw new Exception("Invalid Offset value");
            }
        }
```

Source code 5

```
    public override void Calculate()
        {
            ToolPath_.Clear();
            List<Chain> OrderedChains = new List<Chain>();
            Chain FirstChain;
            CreatChainLoop(GetFirstChain(), out FirstChain);
            OrderedChains.Add(FirstChain);
            // order chains
            if (OriginalChains.Count > 1)
            {
                bool done = false;
                while (!done)
                {
                    Chain NextChain = FindNextChain(FirstChain, OrderedChains);
                    if (NextChain != null)
                    {
                        CreatChainLoop(NextChain, out FirstChain);
                        OrderedChains.Add(FirstChain);
```

```
                    }
                    else
                        done = true;
                }
            }

            ResultChains = OrderedChains;

            //translation between chains
            List<EntityA> CurrentChainToolpath;

            for (int i = 0; i < ResultChains.Count; i++)
            {
                if (i != 0)
                {
                    if (ZLevel > 1)
                    {
                        Line LIN = new Line(ToolPath_[ToolPath_.Count - 1].EndPoint, ResultChains[i -
1].EndLoop);
                        LIN.Color = Color.Blue;
                        LIN.ColorMethod = colorMethodType.byEntity;
                        ToolPath_.Add(new EntityA(LIN) { Cutting = false });
                    }
                    Line L = new Line(ResultChains[i - 1].EndLoop, ResultChains[i].StartLoop);
                    L.Color = Color.CadetBlue;
                    L.ColorMethod = colorMethodType.byEntity;
                    ToolPath_.Add(new EntityA(L) { Cutting = false, FastMove = true });
                }

                CurrentChainToolpath = ResultChains[i].ToolPath;
                ToolPath_.AddRange(CurrentChainToolpath);

                for (int j = 0; j < ZLevel - 1; j++)
                {
                    // Go to the start Point
                    Point3D PP1 = new Point3D(CurrentChainToolpath[CurrentChainToolpath.Count -
1].EndPoint.X, CurrentChainToolpath[CurrentChainToolpath.Count - 1].EndPoint.Y, j * ZlevelOffset);
                    Line Zline2 = new Line(PP1, new Point3D(CurrentChainToolpath[0].StartPoint.X,
CurrentChainToolpath[0].StartPoint.Y, PP1.Z));
                    Zline2.Color = Color.Blue;
                    Zline2.ColorMethod = colorMethodType.byEntity;

                    // check if the chian is not a circle
                    if (!ResultChains[i].IsCircle)
                        ToolPath_.Add(new EntityA(Zline2) { Cutting = false, FastMove = true });

                    //Move down in z Direction
                    Point3D PP2 = new Point3D(Zline2.EndPoint.X, Zline2.EndPoint.Y, (j + 1) * ZlevelOffset);
                    Line Zline1 = new Line(Zline2.EndPoint, PP2);
                    Zline1.Color = Color.YellowGreen;
                    Zline1.ColorMethod = colorMethodType.byEntity;
                    ToolPath_.Add(new EntityA(Zline1) { Cutting = false });

                    for (int k = 0; k < CurrentChainToolpath.Count; k++)
                    {
                        Entity en1 = (Entity)CurrentChainToolpath[k].Entity.Clone();
                        Vector3D Vt = new Vector3D(0, 0, ZlevelOffset * (j + 1));
                        en1.Translate(Vt);
                        EntityA ena1 = new EntityA(en1);
                        if (CurrentChainToolpath[k].Cutting)
                            ena1.Cutting = true;
                        else
                            ena1.Cutting = false;
                        if (CurrentChainToolpath[k].Reversed)
                            ena1.Reversed = true;
                        ToolPath_.Add(ena1);
                    }
                }

            }

            if (ReturnToOrigin)
            {
                if (ZLevel > 1)
                {
                    Line Lreturn1 = new Line(ToolPath_[ToolPath_.Count - 1].EndPoint, new
Point3D(ToolPath_[ToolPath_.Count - 1].EndPoint.X, ToolPath_[ToolPath_.Count - 1].EndPoint.Y, 0));
                    Lreturn1.Color = Color.Blue;
```

```
                    Lreturn1.ColorMethod = colorMethodType.byEntity;
                    ToolPath_.Add(new EntityA(Lreturn1) { Cutting = false });
                }
                Line Lreturn2 = new Line(new Point3D(ToolPath_[ToolPath_.Count - 1].EndPoint.X,
ToolPath_[ToolPath_.Count - 1].EndPoint.Y), new Point3D(0, 0));
                Lreturn2.Color = Color.Blue;
                Lreturn2.ColorMethod = colorMethodType.byEntity;
                ToolPath_.Add(new EntityA(Lreturn2) { Cutting = false });
            }

            // convert curves to line segments
            List<EntityA> ConverToolpath = new List<EntityA>();
            ConverToolpath = ConvertFunction();

            // return results
            ToolPath_ = ConverToolpath;
        }
```

Source code 6

```
private EntityA[] DrawHelix()
        {
            double StartLoopLength = (AccelerationTime * Feed) / 1000;
            double EndLoopLength = (DecelerationTime * Feed) / 1000;
            // create a helix
            LinearPath Helix = LinearPath.CreateHelix(R, Pitch, Turns, true, Tolerance / 2);
            // move to the start point
            Helix.Translate(StartPoint.X, StartPoint.Y, 0);
            // convert helix into sub curves
            List<Line> Lines = Helix.GetIndividualCurves().Cast<Line>().ToList();
            List<EntityA> RS = new List<EntityA>();

            for (int i = 0; i < Lines.Count; i++)
            {
                // create a start move
                if (i == 0)
                {
                    double X1 = Lines[0].StartPoint.X + (Lines[0].StartPoint.X - Lines[0].EndPoint.X) /
Lines[0].Length() * StartLoopLength;
                    double Y1 = Lines[0].StartPoint.Y + (Lines[0].StartPoint.Y - Lines[0].EndPoint.Y) /
Lines[0].Length() * StartLoopLength;
                    double Z1 = Lines[0].StartPoint.Z + (Lines[0].StartPoint.Z - Lines[0].EndPoint.Z) /
Lines[0].Length() * StartLoopLength;
                    RS.Add(Loop.CreateLoopFromEntity(new Line(new Point3D(X1, Y1, Z1), Lines[0].StartPoint),
Feed, false, false));
                }
                RS.Add(Loop.CreateLoopFromEntity(Lines[i], Feed, false, true));
            }

            // create an end move
            double X2 = Lines[Lines.Count - 1].EndPoint.X + (Lines[Lines.Count - 1].EndPoint.X -
Lines[Lines.Count - 1].StartPoint.X) / Lines[Lines.Count - 1].Length() * EndLoopLength;
            double Y2 = Lines[Lines.Count - 1].EndPoint.Y + (Lines[Lines.Count - 1].EndPoint.Y -
Lines[Lines.Count - 1].StartPoint.Y) / Lines[Lines.Count - 1].Length() * EndLoopLength;
            double Z2 = Lines[Lines.Count - 1].EndPoint.Z + (Lines[Lines.Count - 1].EndPoint.Z -
Lines[Lines.Count - 1].StartPoint.Z) / Lines[Lines.Count - 1].Length() * EndLoopLength;
            RS.Add(Loop.CreateLoopFromEntity(new Line(Lines[Lines.Count - 1].EndPoint, new Point3D(X2, Y2,
Z2)), Feed, false, false));
            // return results
            return RS.ToArray();
        }
```

Source code 7

```
   private EntityA[] DrawSquareSpiral()
        {
            if (Length <= StepSize)
                throw new Exception("Invalid Parameter (Step Size B)");

            double INC = StepSize;
            double Increasment = StepSize;
            byte Direction = 1;
            byte Order = 1;
            double X = 0;
            double Y = StartPoint.Y;
```

```
            int LoopCount = 1;
            int PointCount = 0;
            int i = 0;
            List<Point3D> Points = new List<Point3D>();

            // **** add points ****
            while ((Math.Abs(Y - StartPoint.Y) <= (Length) / 2))
            {
                if (i == 0)
                {
                    Points.Add(new Point3D(StartPoint.X, StartPoint.Y, StartPoint.Z));
                    X = StartPoint.X;
                    Y = StartPoint.Y;
                }
                else
                {
                    if (Order == 1)
                    {
                        if (Direction == 1)
                            X = Points[Points.Count - 1].X + Increasment;
                        else
                            X = Points[Points.Count - 1].X - Increasment;

                        Order = 2;
                    }
                    else
                    {
                        if (Direction == 1)
                        {
                            Y = Points[Points.Count - 1].Y + Increasment;
                            Direction = 2;
                        }
                        else
                        {
                            Y = Points[Points.Count - 1].Y - Increasment;
                            Direction = 1;
                        }
                        Order = 1;
                    }

                    Points.Add(new Point3D(X, Y, StartPoint.Z));

                    if (PointCount == 1)
                    {
                        LoopCount++;
                        PointCount = 0;
                        Increasment += INC;
                    }
                    else
                        PointCount++;
                }
                i++;
            }
            // **** add points ****

            Points.RemoveAt(Points.Count - 1);
            Points[Points.Count - 1] = new Point3D(-(-StartPoint.X + ((Points[Points.Count - 2].X -
StartPoint.X))), Points[Points.Count - 2].Y);

            //Check the toolpath style
            if (ToolpathStyle == SpiralStyle.Outward)
                Points.Reverse();

            LinearPath LP = new LinearPath(Points);
            List<Line> Lines = LP.GetIndividualCurves().Cast<Line>().ToList();
            List<EntityA> Rs = new List<EntityA>();

            for (i = 0; i < Lines.Count; i++)
            {
                if (i > 0)
                {
                    EntityA en1 = new EntityA(Lines[i - 1]);
                    EntityA en2 = new EntityA(Lines[i]);
                    var ENRS = Loop.CreateLoop(en1, en2, Feed, AccelerationTime, 5);
                    foreach (var a in ENRS)
                        Rs.Add(Loop.CreateLoopFromEntity(a, Feed, false, false));
                    Rs.Add(Loop.CreateLoopFromEntity(Lines[i], Feed, false, true));
                }
```

```
                else if (i == 0)
                    Rs.Add(Loop.CreateLoopFromEntity(Lines[i], Feed, false, true));
            }

            List<EntityA> Rs_WithLoops = new List<EntityA>();
            List<EntityA> Rs_Final = new List<EntityA>();

            // *****  Create start and end moves *****
            double StartLoopLength = (AccelerationTime * Feed) / 1000;
            Line StartLoop;
            if (ToolpathStyle == SpiralStyle.Outward)
                StartLoop = new Line(new Point3D(Rs[0].StartPoint.X + StartLoopLength, Rs[0].StartPoint.Y),
Rs[0].StartPoint);
            else
                StartLoop = new Line(new Point3D(Rs[0].StartPoint.X - StartLoopLength, Rs[0].StartPoint.Y),
Rs[0].StartPoint);

            Rs_WithLoops.Add(Loop.CreateLoopFromEntity(StartLoop, Feed, false, false));

            Rs_WithLoops.AddRange(Rs);

            double EndLoopLength = (DecelerationTime * Feed) / 1000;
            Line EndLoop;
            if (ToolpathStyle == SpiralStyle.Outward)
                EndLoop = new Line(Rs[Rs.Count - 1].EndPoint, new Point3D(Rs[Rs.Count - 1].EndPoint.X -
EndLoopLength, Rs[Rs.Count - 1].EndPoint.Y));
            else
                EndLoop = new Line(Rs[Rs.Count - 1].EndPoint, new Point3D(Rs[Rs.Count - 1].EndPoint.X +
EndLoopLength, Rs[Rs.Count - 1].EndPoint.Y));

            Rs_WithLoops.Add(Loop.CreateLoopFromEntity(EndLoop, Feed, false, false));
            // *****  Create start and end moves *****

            // **************ZLEVEL***************
            for (i = 0; i < ZLevel; i++)
            {
                foreach (EntityA ena in Rs_WithLoops)
                    Rs_Final.Add(ena.Clone(i * ZlevelOffset));
                if (i != ZLevel - 1)
                {
                    //fast move translation
                    Point3D StartP1 = new Point3D(Rs_WithLoops[Rs_WithLoops.Count - 1].EndPoint.X,
Rs_WithLoops[Rs_WithLoops.Count - 1].EndPoint.Y, i * ZlevelOffset);
                    Point3D EndP1 = new Point3D(Rs_WithLoops[0].StartPoint.X, Rs_WithLoops[0].StartPoint.Y, i
* ZlevelOffset);
                    Line FastM1 = new Line(StartP1, EndP1);
                    Rs_Final.Add(Loop.CreateLoopFromEntity(FastM1, Feed, true, false));
                    //next
                    Line FastM2 = new Line(EndP1, new Point3D(EndP1.X, EndP1.Y, (i + 1) * ZlevelOffset));
                    Rs_Final.Add(Loop.CreateLoopFromEntity(FastM2, Feed, true, false));
                }
            }
            // **************ZLEVEL***************

            return Rs_Final.ToArray();
        }
```

Source code 8

```
private EntityA[] DrawSpiral2()
        {
            double X = StartPoint.X;
            double Y = StartPoint.Y;
            double theta = 0;
            double radius = 0;
            double scale = Length / StepSize;
            double delta = 10;
            double revolutions = StepSize;
            double centreX = StartPoint.X;
            double centreY = StartPoint.Y;
            List<Point3D> Points = new List<Point3D>();

            while (theta <= (revolutions * 360))
            {
                radius = (theta / 360) * scale;
                X = (radius * Math.Cos(theta / 180 * Math.PI)) + centreX;
                Y = (radius * Math.Sin(theta / 180 * Math.PI)) + centreY;
```

```
                Points.Add(new Point3D(X, Y));
                theta += delta;
            }

            //Draw a circle
            theta = 0;
            while (theta <= 360)
            {
                X = (radius * Math.Cos(theta / 180 * Math.PI)) + centreX;
                Y = (radius * Math.Sin(theta / 180 * Math.PI)) + centreY;
                Points.Add(new Point3D(X, Y));

                theta += delta;
            }
            //Check the toolpath style
            if (ToolpathStyle == SpiralStyle.Outward)
                Points.Reverse();

            //Convert the points into arcs
            List<Entity> Result = Core.SplineToArcs.CurveToArcs(Points, Tolerance);

            List<EntityA> Rs = new List<EntityA>();
            List<EntityA> Rs_WithLoops = new List<EntityA>();
            List<EntityA> Rs_Final = new List<EntityA>();

            foreach (var a in Result)
            {
                a.Color = Color.Red;
                a.ColorMethod = colorMethodType.byEntity;
                EntityA ena = new EntityA(a);
                ena.Feed = Feed;
                ena.FastMove = false;
                ena.Cutting = true;
                Rs.Add(ena);
            }

            //start and end loops
            Rs[0].IntersectedPoint = Rs[0].StartPoint;
            Rs_WithLoops.Add(Loop.CreateLoopFromEntity(Loop.CreateStartLoop(Rs[0], Feed, AccelerationTime),
Feed, false, false));

            Rs_WithLoops.AddRange(Rs);

            //End loop
            Rs[Rs.Count - 1].IntersectedPoint = Rs[Rs.Count - 1].EndPoint;
            Rs_WithLoops.Add(Loop.CreateLoopFromEntity(Loop.CreateEndLoop(Rs[Rs.Count - 1], Feed,
DecelerationTime), Feed, false, false));

            // **************ZLEVEL***************
            for (int i = 0; i < ZLevel; i++)
            {
                foreach (EntityA ena in Rs_WithLoops)
                    Rs_Final.Add(ena.Clone(i * ZlevelOffset));
                if (i != ZLevel - 1)
                {
                    //fast move translation
                    Point3D StartP1 = new Point3D(Rs_WithLoops[Rs_WithLoops.Count - 1].EndPoint.X,
Rs_WithLoops[Rs_WithLoops.Count - 1].EndPoint.Y, i * ZlevelOffset);
                    Point3D EndP1 = new Point3D(Rs_WithLoops[0].StartPoint.X, Rs_WithLoops[0].StartPoint.Y, i
* ZlevelOffset);
                    Line FastM1 = new Line(StartP1, EndP1);
                    Rs_Final.Add(Loop.CreateLoopFromEntity(FastM1, Feed, true, false));
                    //next
                    Line FastM2 = new Line(EndP1, new Point3D(EndP1.X, EndP1.Y, (i + 1) * ZlevelOffset));
                    Rs_Final.Add(Loop.CreateLoopFromEntity(FastM2, Feed, true, false));
                }
            }
            // **************ZLEVEL***************

            return Rs_Final.ToArray();
        }
```

Source code 9

```
private EntityA[] DrawZigzag()
        {
            double X = 0;
            double Y = 0;
            int Order = 2;
            int Dir = 1;
            int i = 0;
            double StartLoopLength = (AccelerationTime * Feed) / 1000;
            double EndLoopLength = (DecelerationTime * Feed) / 1000;

            List<Point3D> Points = new List<Point3D>();
            List<EntityA> Rs_Final = new List<EntityA>();

            #region XY Plane

            if (Plane == ZigZagPlane.XY)
            {
                if (CornerR > this.StepSize/2)
                    throw new Exception("Invalide Parameter (Corner radius)");

                if (Direction == ZigZagDirection.AlongY)
                {
                    X = StartPoint.X;
                    double Z = 0;
                    while (X < Length1 + StartPoint.X)
                    {
                        if (i == 0)
                        {
                            Points.Add(new Point3D(StartPoint.X, StartPoint.Y));
                            X = StartPoint.X;
                        }
                        else
                        {
                            if (Order == 1)
                            {
                                X = Points[Points.Count - 1].X + StepSize;
                                Order = 2;
                                Z += Zvariation;
                            }
                            else
                            {
                                if (Dir == 1)
                                {
                                    Y = StartPoint.Y + Length2;
                                    Dir = 2;
                                }
                                else
                                {
                                    Y = StartPoint.Y;
                                    Dir = 1;
                                }
                                Order = 1;
                            }

                            Points.Add(new Point3D(X, Y, Z));
                        }
                        i++;
                    }
                    //add the last point
                    Points.Add(new Point3D(Points[Points.Count - 1].X,
                        Points[Points.Count - 3].Y, Z));
                }
                else
                {
                    Y = StartPoint.Y;
                    double Z = 0;
                    while (Y < (Length2 + StartPoint.Y))
                    {
                        if (i == 0)
                        {
                            Points.Add(new Point3D(StartPoint.X, StartPoint.Y));
                            Y = StartPoint.Y;
                        }
                        else
                        {
                            if (Order == 1)
                            {
                                Y = Points[Points.Count - 1].Y + StepSize;
```

```
                    Order = 2;
                    Z += Zvariation;
                }
                else
                {
                    if (Dir == 1)
                    {
                        X = StartPoint.X + Length1;
                        Dir = 2;
                    }
                    else
                    {
                        X = StartPoint.X;
                        Dir = 1;
                    }
                    Order = 1;
                }

                Points.Add(new Point3D(X, Y, Z));
            }
            i++;
        }
        //add the last point
        Points.Add(new Point3D(Points[Points.Count - 3].X,
            Points[Points.Count - 1].Y, Z));
    }

    LinearPath LP = new LinearPath(Points);
    List<Line> Lines = LP.GetIndividualCurves().Cast<Line>().ToList();
    List<Entity> Result = new List<Entity>();
    List<Entity> DeleteList = new List<Entity>();

    Arc arcR;

    for (i = 0; i < Lines.Count; i++)
    {
        if (i > 0)
        {
            Curve.Fillet(Lines[i - 1], Lines[i], CornerR,
                false, false, true, true, out arcR);

            if (CornerR == this.StepSize / 2)
            {
                if (Math.Round(Lines[i - 1].Length(), 5) == CornerR)
                    DeleteList.Add(Lines[i - 1]);
                if (Math.Round(Lines[i].Length(), 5) == CornerR)
                    DeleteList.Add(Lines[i]);
            }

            Result.Add(arcR);
            Result.Add(Lines[i]);
        }
        else if (i == 0)
            Result.Add(Lines[i]);
    }

    foreach (var en in DeleteList)
        Result.Remove(en);

    List<EntityA> Rs = new List<EntityA>();
    List<EntityA> Rs_WithLoops = new List<EntityA>();

    foreach (var a in Result)
    {
        a.Color = Color.Red;
        a.ColorMethod = colorMethodType.byEntity;
        EntityA ena = new EntityA(a);
        ena.Feed = Feed;
        ena.FastMove = false;
        ena.Cutting = true;
        Rs.Add(ena);
    }

    //start and end loops
    Line StartLoop;
    if (Direction == ZigZagDirection.AlongX)
        StartLoop = new Line(new Point3D(Rs[0].StartPoint.X - StartLoopLength,
            Rs[0].StartPoint.Y), Rs[0].StartPoint);
```

```
                else
                    StartLoop = new Line(new Point3D(Rs[0].StartPoint.X,
                        Rs[0].StartPoint.Y - StartLoopLength), Rs[0].StartPoint);

                StartLoop.Color = Loop.LoopColor;
                StartLoop.ColorMethod = colorMethodType.byEntity;
                EntityA EnAStartLoop = new EntityA(StartLoop);
                EnAStartLoop.Feed = Feed;
                EnAStartLoop.FastMove = false;
                EnAStartLoop.Cutting = false;
                Rs_WithLoops.Add(EnAStartLoop);

                Rs_WithLoops.AddRange(Rs);

                //End loop
                Line EndLoop;
                if (Direction == ZigZagDirection.AlongY)
                {
                    if (Dir == 1)
                        EndLoop = new Line(Rs[Rs.Count - 1].EndPoint,
                            new Point3D(Rs[Rs.Count - 1].EndPoint.X,
                            Rs[Rs.Count - 1].EndPoint.Y + EndLoopLength,
                            Rs[Rs.Count - 1].EndPoint.Z));
                    else
                        EndLoop = new Line(Rs[Rs.Count - 1].EndPoint,
                            new Point3D(Rs[Rs.Count - 1].EndPoint.X,
                            Rs[Rs.Count - 1].EndPoint.Y - EndLoopLength,
                            Rs[Rs.Count - 1].EndPoint.Z));
                }
                else
                {
                    if (Dir == 1)
                        EndLoop = new Line(Rs[Rs.Count - 1].EndPoint,
                            new Point3D(Rs[Rs.Count - 1].EndPoint.X + EndLoopLength,
                            Rs[Rs.Count - 1].EndPoint.Y, Rs[Rs.Count - 1].EndPoint.Z));
                    else
                        EndLoop = new Line(Rs[Rs.Count - 1].EndPoint,
                            new Point3D(Rs[Rs.Count - 1].EndPoint.X - EndLoopLength,
                            Rs[Rs.Count - 1].EndPoint.Y,
                            Rs[Rs.Count - 1].EndPoint.Z));
                }

                EndLoop.Color = Loop.LoopColor;
                EndLoop.ColorMethod = colorMethodType.byEntity;
                EntityA EnAEndLoop = new EntityA(EndLoop);
                EnAEndLoop.Feed = Feed;
                EnAEndLoop.FastMove = false;
                EnAEndLoop.Cutting = false;
                Rs_WithLoops.Add(EnAEndLoop);

                // **************ZLEVEL***************
                for (i = 0; i < ZLevel; i++)
                {
                    foreach (EntityA ena in Rs_WithLoops)
                        Rs_Final.Add(ena.Clone(i * ZlevelOffset));

                    if (i != ZLevel - 1)
                    {
                        //fast move translation
                        Point3D StartP1 = new Point3D(Rs_WithLoops[Rs_WithLoops.Count - 1].EndPoint.X,
                            Rs_WithLoops[Rs_WithLoops.Count - 1].EndPoint.Y,
                            Rs_WithLoops[Rs_WithLoops.Count - 1].EndPoint.Z + (i * ZlevelOffset));
                        Point3D EndP1 = new Point3D(Rs_WithLoops[0].StartPoint.X,
                            Rs_WithLoops[0].StartPoint.Y,
                            Rs_WithLoops[0].StartPoint.Z + (i * ZlevelOffset));
                        Line FastM1 = new Line(StartP1, EndP1);
                        FastM1.Color = Color.Blue;
                        FastM1.ColorMethod = colorMethodType.byEntity;
                        EntityA FastN1 = new EntityA(FastM1);
                        FastN1.Cutting = false;
                        FastN1.FastMove = true;
                        Rs_Final.Add(FastN1);
                        //next
                        Line FastM2 = new Line(EndP1, new Point3D(EndP1.X,
                            EndP1.Y, (i + 1) * ZlevelOffset));
                        FastM2.Color = Color.Blue;
                        FastM2.ColorMethod = colorMethodType.byEntity;
                        EntityA FastN2 = new EntityA(FastM2);
```

```
                        FastN2.Cutting = false;
                        FastN2.FastMove = true;
                        Rs_Final.Add(FastN2);
                    }
                }

                return Rs_Final.ToArray();
            }

            #endregion

            #region XZ Plane

            else
            {
                List<Line> Lines = new List<Line>();

                #region Along X

                if (Direction == ZigZagDirection.AlongX)
                {
                    Dir = 1;
                    for (i = 0; i <= Length2 / StepSize; i++)
                    {
                        if (i == 0)
                            Lines.Add(new Line(new Point3D(StartPoint.X, StartPoint.Y),
                                new Point3D(StartPoint.X + Length1, StartPoint.Y)));
                        else
                        {
                            if (Dir == 1)
                            {
                                Point3D PT1 = new Point3D(Lines[i - 1].EndPoint.X,
                                    Lines[i - 1].EndPoint.Y, i * StepSize);
                                Lines.Add(new Line(PT1, new Point3D(StartPoint.X,
                                    PT1.Y, PT1.Z)));
                                Dir = 2;
                            }
                            else
                            {
                                Point3D PT1 = new Point3D(Lines[i - 1].EndPoint.X,
                                    Lines[i - 1].EndPoint.Y, i * StepSize);
                                Lines.Add(new Line(PT1, new Point3D(StartPoint.X + Length1,
                                    PT1.Y, PT1.Z)));
                                Dir = 1;
                            }
                        }
                    }

                    List<EntityA> Rs = new List<EntityA>();
                    List<EntityA> Rs_WithLoops = new List<EntityA>();
                    Dir = 1;

                    for (i = 0; i < Lines.Count; i++)
                    {
                        if (i == 0)
                        {
                            Line L1 = new Line(new Point3D(StartPoint.X - StartLoopLength,
                                Lines[0].StartPoint.Y),
                                new Point3D(Lines[0].StartPoint.X, Lines[0].StartPoint.Y));
                            Rs.Add(Loop.CreateLoopFromEntity(L1, Feed, false, false));
                        }
                        else
                        {

                            if (Dir == 1)
                            {
                                // Loop1
                                Line L1 = new Line(Lines[i - 1].EndPoint,
                                    new Point3D(Lines[i - 1].EndPoint.X + EndLoopLength, Lines[i -
1].EndPoint.Y,
                                        Lines[i - 1].EndPoint.Z));
                                Rs.Add(Loop.CreateLoopFromEntity(L1, Feed, false, false));

                                //Loop2
                                Line L2 = new Line(L1.EndPoint, new Point3D(Lines[i].StartPoint.X +
StartLoopLength,
                                    Lines[i].StartPoint.Y, Lines[i].StartPoint.Z));
                                Rs.Add(Loop.CreateLoopFromEntity(L2, Feed, true, false));
```

```
                                //Loop3
                                Line L3 = new Line(L2.EndPoint, Lines[i].StartPoint);
                                Rs.Add(Loop.CreateLoopFromEntity(L3, Feed, false, false));

                                Dir = 2;
                            }
                            else
                            {
                                // Loop1
                                Line L1 = new Line(Lines[i - 1].EndPoint,
                                    new Point3D(Lines[i - 1].EndPoint.X - EndLoopLength, Lines[i -
1].EndPoint.Y,
                                        Lines[i - 1].EndPoint.Z));
                                Rs.Add(Loop.CreateLoopFromEntity(L1, Feed, false, false));

                                //Loop2
                                Line L2 = new Line(L1.EndPoint, new Point3D(Lines[i].StartPoint.X -
StartLoopLength,
                                    Lines[i].StartPoint.Y, Lines[i].StartPoint.Z));
                                Rs.Add(Loop.CreateLoopFromEntity(L2, Feed, true, false));

                                //Loop3
                                Line L3 = new Line(L2.EndPoint, Lines[i].StartPoint);
                                Rs.Add(Loop.CreateLoopFromEntity(L3, Feed, false, false));

                                Dir = 1;
                            }
                        }

                        Lines[i].Color = Color.Red;
                        Lines[i].ColorMethod = colorMethodType.byEntity;
                        EntityA ena1 = new EntityA(Lines[i]);
                        ena1.Feed = Feed;
                        ena1.FastMove = false;
                        ena1.Cutting = true;
                        Rs.Add(ena1);
                    }

                    //End Loop
                    if (Dir == 2)
                    {
                        Line LEND = new Line(Lines[Lines.Count - 1].EndPoint,
                            new Point3D(Lines[Lines.Count - 1].EndPoint.X - EndLoopLength,
                                Lines[Lines.Count - 1].EndPoint.Y, Lines[Lines.Count - 1].EndPoint.Z));
                        Rs.Add(Loop.CreateLoopFromEntity(LEND, Feed, false, false));
                    }
                    else
                    {
                        Line LEND = new Line(Lines[Lines.Count - 1].EndPoint,
                            new Point3D(Lines[Lines.Count - 1].EndPoint.X + EndLoopLength,
                                Lines[Lines.Count - 1].EndPoint.Y, Lines[Lines.Count - 1].EndPoint.Z));
                        Rs.Add(Loop.CreateLoopFromEntity(LEND, Feed, false, false));
                    }


                    return Rs.ToArray();
                }

                #endregion

                #region Along Y

                else
                {
                    Dir = 1;
                    for (i = 0; i <= Length2 / StepSize; i++)
                    {
                        if (i == 0)
                            Lines.Add(new Line(new Point3D(StartPoint.X, StartPoint.Y),
                                new Point3D(StartPoint.X, StartPoint.Y + Length1)));
                        else
                        {
                            if (Dir == 1)
                            {
                                Point3D PT1 = new Point3D(Lines[i - 1].EndPoint.X,
                                    Lines[i - 1].EndPoint.Y, i * StepSize);
                                Lines.Add(new Line(PT1, new Point3D(PT1.X,
```

```
                        StartPoint.Y, PT1.Z)));
                    Dir = 2;
                }
                else
                {
                    Point3D PT1 = new Point3D(Lines[i - 1].EndPoint.X,
                        Lines[i - 1].EndPoint.Y, i * StepSize);
                    Lines.Add(new Line(PT1, new Point3D(StartPoint.X,
                        PT1.Y + Length1, PT1.Z)));
                    Dir = 1;
                }
            }
        }

        #region Looping

        List<EntityA> Rs = new List<EntityA>();
        List<EntityA> Rs_WithLoops = new List<EntityA>();
        Dir = 1;

        for (i = 0; i < Lines.Count; i++)
        {
            if (i == 0)
            {
                Line L1 = new Line(new Point3D(StartPoint.X,
                    Lines[0].StartPoint.Y - StartLoopLength),
                    new Point3D(Lines[0].StartPoint.X,
                        Lines[0].StartPoint.Y));
                Rs.Add(Loop.CreateLoopFromEntity(L1, Feed,
                    false, false));
            }
            else
            {

                if (Dir == 1)
                {
                    // Loop1
                    Line L1 = new Line(Lines[i - 1].EndPoint,
                        new Point3D(Lines[i - 1].EndPoint.X,
                            Lines[i - 1].EndPoint.Y + EndLoopLength,
                            Lines[i - 1].EndPoint.Z));
                    Rs.Add(Loop.CreateLoopFromEntity(L1, Feed, false, false));

                    //Loop2
                    Line L2 = new Line(L1.EndPoint,
                        new Point3D(Lines[i].StartPoint.X,
                            Lines[i].StartPoint.Y + StartLoopLength,
                            Lines[i].StartPoint.Z));
                    Rs.Add(Loop.CreateLoopFromEntity(L2, Feed,
                        true, false));

                    //Loop3
                    Line L3 = new Line(L2.EndPoint,
                        Lines[i].StartPoint);
                    Rs.Add(Loop.CreateLoopFromEntity(L3,
                        Feed, false, false));

                    Dir = 2;
                }
                else
                {
                    // Loop1
                    Line L1 = new Line(Lines[i - 1].EndPoint,
                        new Point3D(Lines[i - 1].EndPoint.X,
                            Lines[i - 1].EndPoint.Y - EndLoopLength,
                            Lines[i - 1].EndPoint.Z));
                    Rs.Add(Loop.CreateLoopFromEntity(L1, Feed,
                        false, false));

                    //Loop2
                    Line L2 = new Line(L1.EndPoint,
                        new Point3D(Lines[i].StartPoint.X,
                        Lines[i].StartPoint.Y - StartLoopLength,
                        Lines[i].StartPoint.Z));
                    Rs.Add(Loop.CreateLoopFromEntity(L2, Feed,
                        true, false));

                    //Loop3
```

```
                                Line L3 = new Line(L2.EndPoint,
                                    Lines[i].StartPoint);
                                Rs.Add(Loop.CreateLoopFromEntity(L3, Feed,
                                    false, false));

                                Dir = 1;
                            }
                        }

                        Lines[i].Color = Color.Red;
                        Lines[i].ColorMethod = colorMethodType.byEntity;
                        EntityA ena1 = new EntityA(Lines[i]);
                        ena1.Feed = Feed;
                        ena1.FastMove = false;
                        ena1.Cutting = true;
                        Rs.Add(ena1);
                    }

                    //End Loop
                    if (Dir == 2)
                    {
                        Line LEND = new Line(Lines[Lines.Count - 1].EndPoint,
                            new Point3D(Lines[Lines.Count - 1].EndPoint.X,
                                Lines[Lines.Count - 1].EndPoint.Y - EndLoopLength,
                                Lines[Lines.Count - 1].EndPoint.Z));
                        Rs.Add(Loop.CreateLoopFromEntity(LEND, Feed, false, false));
                    }
                    else
                    {
                        Line LEND = new Line(Lines[Lines.Count - 1].EndPoint,
                          new Point3D(Lines[Lines.Count - 1].EndPoint.X,
                              Lines[Lines.Count - 1].EndPoint.Y + EndLoopLength,
                              Lines[Lines.Count - 1].EndPoint.Z));
                        Rs.Add(Loop.CreateLoopFromEntity(LEND, Feed, false, false));
                    }

                    #endregion

                    return Rs.ToArray();
                }

                #endregion
            }

            #endregion
        }
```

Source code 10

```
    private EntityA[] DrawCone()
        {
            if (BeamD > d2)
                throw new Exception("Beam Diameter must be less than D2");

            if (d2 > d1)
                throw new Exception("D1 must be bigger than D2");

            List<EntityA> RSS = new List<EntityA>();
            RSS.AddRange(DrawSpiral(d1, 0));
            EntityA[] SpiralEntities;

            if (Zlevel > 1)
            {
                double hh;
                double Angle = Math.Atan(((d1 - d2) / 2) / H);
                for (int i = 1; i < Zlevel; i++)
                {
                    hh = H - (i * (H / Zlevel));
                    double rr = d2 + 2 * (hh * Math.Tan(Angle));
                    SpiralEntities = DrawSpiral(rr, H - hh);
                    Line FastTRMove = new Line(RSS[RSS.Count - 1].EndPoint,
                        SpiralEntities[0].StartPoint);
                    RSS.Add(Loop.CreateLoopFromEntity(FastTRMove,
                        Feed, true, false));
                    RSS.AddRange(SpiralEntities);
                }
            }
```

```
            SpiralEntities = DrawSpiral(d2, H);
            Line FastTRMoveEnd = new Line(RSS[RSS.Count - 1].EndPoint,
                SpiralEntities[0].StartPoint);
            RSS.Add(Loop.CreateLoopFromEntity(FastTRMoveEnd,
                Feed, true, false));
            RSS.AddRange(SpiralEntities);

            return RSS.ToArray();
        }
```

Source code 11

```
private void DoSlice()
        {
            Plane Pl1 = Plane.XY;
            double t = 0.01;
            int ZlevelCount = (int)((Model_.BoxMax.Z - Model_.BoxMin.Z) / ZOffset);

            if(SectionTYPE == Core.SectionTYPE.DownToUp)
                Pl1.Translate(0, 0, Model_.BoxMin.Z);
            else
                Pl1.Translate(0, 0, Model_.BoxMax.Z);

            Point3D LASTENDPOINT = null;

            for (int i = 0; i < ZlevelCount; i++)
            {
                List<Entity> Entities_Pocket = new List<Entity>();
                ICurve[] RC = Model_.Section(Pl1, t);

                devDept.Eyeshot.Entities.Region r1 = new devDept.Eyeshot.Entities.Region(RC,
                    Plane.XY, true);

                RC = r1.Pocket(R, cornerType.Round, t);
                foreach (Entity en in RC)
                {
                    if (en is LinearPath)
                    {
                        LinearPath LP = en as LinearPath;
                        LP.Translate(0, 0, Pl1.Origin.Z);
                        Entities_Pocket.AddRange(LP.Explode().Cast<Entity>());
                    }
                }

                if (Entities_Pocket.Count > 0)
                {
                    ImportCore GChains = new ImportCore(Entities_Pocket.ToArray(),
                        false);
                    ContourToolpath CT = new ContourToolpath(GChains.Chains,
                        Entities_Pocket, 0, OffsetType.None, 0);
                    CT.Feed = this.Feed;
                    CT.AccelerationTime = this.AccelerationTime;
                    CT.DecelerationTime = this.DecelerationTime;
                    CT.MaxDeg = MaxDeg;
                    CT.Calculate();

                    if (i != 0 && LASTENDPOINT !=null)
                    {
                        Line FastTRMoveEnd = new Line(LASTENDPOINT,
                            CT.StartPointLoop);
                        ToolPath_.Add(Loop.CreateLoopFromEntity(FastTRMoveEnd,
                            Feed, true, false));
                    }
                    ToolPath_.AddRange(CT.ToolPathResult);
                    LASTENDPOINT = CT.EndPointLoop;
                }
                if(SectionTYPE == Core.SectionTYPE.DownToUp)
                Pl1.Translate(0, 0, ZOffset);
                else
                    Pl1.Translate(0, 0, -ZOffset);
            }
        }
```

Source code 12

```
public static List<Entity> CurveToArcs(List<Point3D> Points, double T)
        {
            try
            {
                if (Points.Count > 2)
                {
                    Curve CC1 = Curve.CubicSplineInterpolation(Points);
                    double StartU = 0;
                    double EndU = 0;
                    double RR = double.PositiveInfinity;
                    bool StartPoint = false;
                    Point3D P1 = new Point3D();
                    Point3D P2 = null;
                    Point3D MidPoint;
                    double L = CC1.Length();
                    List<Entity> Results = new List<Entity>();
                    T = T / CC1.Length();
                    double LoopCount = 1 / T;
                    double U = -T;

                    for (int i = 0; i < LoopCount; i++)
                    {
                        U += T;
                        if (RR != Math.Round(CC1.Curvature(U), 2))
                        {
                            if (!StartPoint)
                            {
                                StartPoint = true;
                                P1 = CC1.PointAt(U);
                                StartU = U;
                                //next arc
                                if (P2 != null)
                                {
                                    MidPoint = CC1.PointAt(EndU + ((U - EndU) / 2));
                                    Arc arc = new Arc(P2, MidPoint, P1, false);
                                    Results.Add(arc);
                                    P2 = null;
                                }

                            }
                            else
                            {
                                StartPoint = false;
                                P2 = CC1.PointAt(U);
                                MidPoint = CC1.PointAt(StartU + ((U - StartU) / 2));
                                Arc arc = new Arc(P1, MidPoint, P2, false);
                                Results.Add(arc);
                                RR = Math.Round(CC1.Curvature(U), 2);
                                EndU = U;
                            }
                        }
                    }

                    if (Math.Round(U, 3) != 1)
                    {
                        Arc Parc = Results[Results.Count - 1] as Arc;
                        MidPoint = CC1.PointAt(U + ((1 - U) / 2));
                        P2 = CC1.PointAt(1);
                        Arc arc = new Arc(Parc.EndPoint, MidPoint, P2, false);
                        Results.Add(arc);
                    }

                    return Results;
                }
                else
                    return new List<Entity>();
            }
            catch
            {
                throw new Exception("Invalide Tolerance Value");
            }

        }
```

Source code 13

```
public static string GenerateGCode(List<ToolPathAll> TPS, int Tolerance, string CuttingVaribale_,
            string VariableProgramLine_, string G0, string G1, string G2, string G3, out int PLine)
        {
            VaribaleCuttingON = CuttingVaribale_ + "=1";
            VaribaleCuttingOFF = CuttingVaribale_ + "=0";
            VaribaleProgramLine = VariableProgramLine_;
            G0_ = G0;
            G1_ = G1;
            G2_ = G2;
            G3_ = G3;
            CurrentVaribaleCutting = "";
            CurrentG = "";
            CurrentX = double.NegativeInfinity;
            CurrentY = double.NegativeInfinity;
            CurrentZ = double.NegativeInfinity;
            CurrentI = double.NegativeInfinity;
            CurrentJ = double.NegativeInfinity;
            CurrentF = double.NegativeInfinity;
            ProgramLine = 1;
            Points.Clear();
            StringBuilder SB = new StringBuilder();
            //G-code header
            SB.AppendLine("OPEN PROG 001 CLEAR");
            SB.AppendLine("G54 G90 " + VaribaleCuttingOFF);
            //

            double tempCurrentI = double.NegativeInfinity;
            double tempCurrentJ = double.NegativeInfinity;
            string tempGcode = "";
            Point3D CurrentPoint;
            Point3D Arcstart;

            foreach (ToolPathAll T in TPS)
            {
                for (int i = 0; i < T.ToolPathResult.Count; i++)
                {
                    EntityA en = T.ToolPathResult[i];

                    if (en.EntityType == EntityType.Line || en.EntityType == EntityType.LinearPath)
                    {
                        Line line = en.Entity as Line;

                        if (line.Length() < Math.Pow(10, -Tolerance))
                            continue;

                        if (Points.Count == 0)
                        {
                            SB.AppendLine(GenerateG1(en, Tolerance, true, true, en.Cutting));
                            SB.AppendLine(GenerateG1(en, Tolerance, false, en.FastMove, en.Cutting));
                        }
                        else
                        {
                            if (!en.Reversed)
                                CurrentPoint = en.EndPoint;
                            else
                                CurrentPoint = en.StartPoint;
                            SB.AppendLine(GenerateG1(en, Tolerance, false, en.FastMove, en.Cutting));
                        }
                    }
                    else if (en.EntityType == EntityType.Arc)
                    {
                        Arc arc = en.Entity as Arc;
                        //check the length
                        if (arc.Length() < Math.Pow(10, -Tolerance))
                            continue;

                        if (!en.Reversed)
                        {
                            CurrentPoint = en.EndPoint;
                            Arcstart = en.StartPoint;
                            tempCurrentI = Math.Round(arc.Center.X - arc.StartPoint.X, Tolerance);
                            tempCurrentJ = Math.Round(arc.Center.Y - arc.StartPoint.Y, Tolerance);
                        }
```

```
                        else
                        {
                            CurrentPoint = en.StartPoint;
                            Arcstart = en.EndPoint;
                            tempCurrentI = Math.Round(arc.Center.X - arc.EndPoint.X, Tolerance);
                            tempCurrentJ = Math.Round(arc.Center.Y - arc.EndPoint.Y, Tolerance);
                        }

                        if (IsArcclockwise(arc))
                        {
                            if (en.Reversed)
                                tempGcode = G3_;
                            else
                                tempGcode = G2_;
                        }
                        else
                        {
                            if (en.Reversed)
                                tempGcode = G2_;
                            else
                                tempGcode = G3_;
                        }

                        if (CurrentG != tempGcode)
                            SB.Append(tempGcode + " ");

                        CurrentG = tempGcode;

                        SB.Append("X" + Math.Round(CurrentPoint.X, Tolerance).ToString("0.######") + " ");

                        SB.Append("Y" + Math.Round(CurrentPoint.Y, Tolerance).ToString("0.######") + " ");

                        if (CurrentZ != Math.Round(CurrentPoint.Z, Tolerance))
                            SB.Append("Z" + Math.Round(CurrentPoint.Z, Tolerance).ToString("0.######") + "
");


                        SB.Append("I" + tempCurrentI.ToString("0.######") + " ");

                        SB.Append("J" + tempCurrentJ.ToString("0.######") + " ");

                        SB.AppendLine();

                        // **************Cutting ON/OFF********************
                        if (en.Cutting)
                        {
                            if (CurrentVaribaleCutting != VaribaleCuttingON)
                            {
                                SB.Append(VaribaleCuttingON + " ");
                                CurrentVaribaleCutting = VaribaleCuttingON;
                            }
                        }
                        else
                        {
                            if (CurrentVaribaleCutting != VaribaleCuttingOFF)
                            {
                                SB.Append(VaribaleCuttingOFF + " ");
                                CurrentVaribaleCutting = VaribaleCuttingOFF;
                            }
                        }
                        // **************END Cutting ON/OFF********************

                        //**************PROGRAM LINE*************************
                        SB.Append(VaribaleProgramLine + "=" + ProgramLine);
                        ProgramLine++;
                        //**************END PROGRAM LINE*************************

                        SB.AppendLine();
                        Points.Add(CurrentPoint);
                        CurrentZ = Math.Round(CurrentPoint.Z, Tolerance);
                        CurrentY = Math.Round(CurrentPoint.Y, Tolerance);
                        CurrentX = Math.Round(CurrentPoint.X, Tolerance);
                        CurrentI = tempCurrentI;
                        CurrentJ = tempCurrentJ;
                    }
                    else if (en.EntityType == EntityType.Circle)
                    {
                        Circle circle = en.Entity as Circle;
                        CurrentPoint = new Point3D(circle.Center.X, circle.Center.Y + circle.Radius,
```

```
circle.Center.Z);
                        en.StartPoint = CurrentPoint;

                        if (CurrentG != G3_)
                            SB.Append(G3_ + " ");

                        CurrentG = G3_;

                        SB.Append("X" + Math.Round(CurrentPoint.X, Tolerance).ToString("0.######") + " ");

                        SB.Append("Y" + Math.Round(CurrentPoint.Y, Tolerance).ToString("0.######") + " ");

                        SB.Append("Z" + Math.Round(CurrentPoint.Z, Tolerance).ToString("0.######") + " ");

                        tempCurrentI = 0;
                        tempCurrentJ = -circle.Radius;

                        SB.Append("I" + tempCurrentI.ToString("0.######") + " ");

                        SB.Append("J" + tempCurrentJ.ToString("0.######") + " ");

                        Points.Add(CurrentPoint);

                        SB.AppendLine();

                        // **************Cutting ON/OFF********************
                        if (en.Cutting)
                        {
                            if (CurrentVaribaleCutting != VaribaleCuttingON)
                            {
                                SB.Append(VaribaleCuttingON + " ");
                                CurrentVaribaleCutting = VaribaleCuttingON;
                            }
                        }
                        else
                        {
                            if (CurrentVaribaleCutting != VaribaleCuttingOFF)
                            {
                                SB.Append(VaribaleCuttingOFF + " ");
                                CurrentVaribaleCutting = VaribaleCuttingOFF;
                            }
                        }
                        // **************END Cutting ON/OFF********************

                        //**************PROGRAM LINE**************************
                        SB.Append(VaribaleProgramLine + "=" + ProgramLine);
                        ProgramLine++;
                        //**************END PROGRAM LINE**************************

                        SB.AppendLine();

                        CurrentZ = Math.Round(CurrentPoint.Z, Tolerance);
                        CurrentY = Math.Round(CurrentPoint.Y, Tolerance);
                        CurrentX = Math.Round(CurrentPoint.X, Tolerance);
                        CurrentI = tempCurrentI;
                        CurrentJ = tempCurrentJ;
                    }
                }
            }

            //End of G-code Program
            SB.AppendLine("M30");
            SB.AppendLine("%");
            SB.AppendLine("CLOSE");

            PLine = ProgramLine;
            return SB.ToString();
        }
```

Source code 14

```
private string[] ProgramSplit()
        {
            string line;
            List<GcodeLine> Gcodelines = new List<GcodeLine>();
            string[] Stringtrim = new string[1] { " " };
            string G0 = Settings.Default.G0;
```

```
            string G1 = Settings.Default.G1;
            string G2 = Settings.Default.G2;
            string G3 = Settings.Default.G3;
            string CurrentG = "";
            string PCurrentG = "";
            bool PN = false;
            double feed = 0;

            using (StringReader reader = new StringReader(GCODE_Program))
            {
                while ((line = reader.ReadLine()) != null)
                {
                    if (line.Contains(ProgramLineVarible_))
                        PN = true;
                    else
                        PN = false;

                    if (line.Contains("F"))
                    {
                        string[] NS = line.Split(Stringtrim, StringSplitOptions.None);
                        foreach (string S in NS)
                        {
                            if (S.StartsWith("F"))
                            {
                                feed = double.Parse(S.Substring(1));
                                break;
                            }
                        }
                    }

                    if (line.Contains(CuttingON_))
                        Cutting = true;
                    if (line.Contains(CuttingOFF_))
                        Cutting = false;
                    if (line.StartsWith(G0))
                    {
                        CurrentG = G0;
                        feed = 0;
                    }
                    else if (line.StartsWith(G1))
                        CurrentG = G1;
                    else if (line.StartsWith(G2))
                        CurrentG = G2;
                    else if (line.StartsWith(G3))
                        CurrentG = G3;
                    else
                        CurrentG = PCurrentG;
                    Gcodelines.Add(new GcodeLine(line.Split(Stringtrim, StringSplitOptions.None).Length,
                        line, Cutting, CurrentG, PN, feed));
                    PCurrentG = CurrentG;
                }
            }

            int WordCount = 0;
            SplitIndexes = new List<int>();
            int Splitcount = 0;
            int PPIndex = 0;

            for (int i = 0; i < Gcodelines.Count; i++)
            {
                WordCount += Gcodelines[i].Words;
                if (WordCount + 10 > Maxword && i < GCODE.Count - 3)
                {
                    if (!Gcodelines[i].Cutting)
                    {
                        if (Gcodelines[i].LineNumber)
                        {
                            SplitIndexes.Add(i);
                            Splitcount = i;
                            PPIndex = i;
                        }
                        else
                        {
                            SplitIndexes.Add(i + 1);
                            Splitcount = i + 1;
                            PPIndex = i + 1;
                        }
```

```
                    WordCount = 0;
                }
                else
                {
                    for (int j = i; j > PPIndex + 4; j--)
                    {
                        if (!Gcodelines[j].Cutting)
                        {
                            if (Gcodelines[j].LineNumber)
                            {
                                SplitIndexes.Add(j);
                                Splitcount = j;
                                PPIndex = j;
                            }
                            else
                            {
                                SplitIndexes.Add(j - 1);
                                Splitcount = j - 1;
                                PPIndex = j - 1;
                            }
                            WordCount = 0;
                            break;
                        }
                        if (j == 5)
                        {
                            //if segment we cutting off is not found
                            if (Gcodelines[i].LineNumber)
                            {
                                SplitIndexes.Add(i);
                                Splitcount = i;
                            }
                            else
                            {
                                SplitIndexes.Add(i + 1);
                                Splitcount = i + 1;
                            }

                            WordCount = 0;
                        }
                    }

                }
            }
        }

        int startedLine = 0;
        string[] Programs = new string[SplitIndexes.Count + 1];
        for (int i = 0; i < SplitIndexes.Count; i++)
        {
            StringBuilder SB = new StringBuilder();
            if (i != 0)
            {
                SB.AppendLine("OPEN PROG 001 CLEAR");
                SB.AppendLine("G54 G90");
            }
            for (int j = startedLine; j <= SplitIndexes[i]; j++)
            {
                if (j != startedLine)
                    SB.AppendLine(Gcodelines[j].Line);
                else if (j == SplitIndexes[i] - 1)
                {
                    if (Gcodelines[j].Cutting)
                        SB.AppendLine(CuttingVarible_ + "=0");
                }
                else
                {
                    if (!Gcodelines[j].Line.StartsWith(G0) && !Gcodelines[j].Line.StartsWith(G1)
                        && !Gcodelines[j].Line.StartsWith(G2) && !Gcodelines[j].Line.StartsWith(G3)
                        && Gcodelines[j].Gcode.Length > 0)
                        SB.Append(Gcodelines[j].Gcode + " " + Gcodelines[j].Line);
                    else
                        SB.Append(Gcodelines[j].Line);

                    if (!Gcodelines[j].Line.Contains("F") && Gcodelines[j].Feed != 0)
                        SB.Append(" F" + Gcodelines[j].Feed);

                    SB.AppendLine();
                }
```

```
                }
                SB.AppendLine("M30");
                SB.AppendLine("%");
                SB.AppendLine("CLOSE");
                Programs[i] = SB.ToString();
                startedLine = SplitIndexes[i] + 1;
            }

            StringBuilder SB1 = new StringBuilder();
            SB1.AppendLine("OPEN PROG 001 CLEAR");
            SB1.AppendLine("G54 G90");
            for (int j = SplitIndexes[SplitIndexes.Count - 1] + 1; j < Gcodelines.Count; j++)
            {
                if (j != startedLine)
                    SB1.AppendLine(Gcodelines[j].Line);
                else
                {
                    if (!Gcodelines[j].Line.StartsWith(G0) && !Gcodelines[j].Line.StartsWith(G1)
                        && !Gcodelines[j].Line.StartsWith(G2) && !Gcodelines[j].Line.StartsWith(G3))
                        SB1.Append(Gcodelines[j].Gcode + " " + Gcodelines[j].Line);
                    else
                        SB1.Append(Gcodelines[j].Line);

                    if (!Gcodelines[j].Line.Contains("F") && Gcodelines[j].Feed != 0)
                        SB1.Append(" F" + Gcodelines[j].Feed);

                    SB1.AppendLine();
                }
            }
            SB1.AppendLine("M30");
            SB1.AppendLine("%");
            SB1.AppendLine("CLOSE");

            Programs[Programs.Length - 1] = SB1.ToString();

            return Programs;
        }
```