

Saimaa University of Applied Sciences
Faculty of Technology, Lappeenranta
Mechanical Engineering and Production Technology

Spencer Raisanen

Semi-Automated Turning Simulation Production Using Catia V5

Thesis 2015

Abstract

Spencer Warren Raisanen

Semi-Automated Turning Simulation Production Using Catia V5, 105 pages, 2 appendices

Saimaa University of Applied Sciences

Bachelor of Engineering, Lappeenranta

Mechanical Engineering and Production Technology

Thesis 2015

Instructor: Lecturer Jouni Könönen, Saimaa University of Applied Sciences

The purpose of this thesis was to explore the possibility of using CAPP within a CAD/CAM program with an end goal of producing turning simulations for simple geometries. The system requires only simple geometrical input from the user and a user entered tool catalog. The end goal of this thesis was for CNC-code production to be possible with minimal human entry necessary through the CAM interface, but allowing for the user to check the simulation to ensure all parameters are correct and make minor adjustments as desired. The work was commissioned by Jouni Könönen of Saimaa University of Applied Sciences.

The information of this thesis was collected from literature, internet sources, practical experience, and by conversations with people knowledgeable in either the field of machining or programming. The system works by logically deciding which machining operations are necessary, the toolholders needed, the desired insert shape and size, and the machining parameters. This information is all uploaded to Catia V5 which does the actual simulation, allowing the development to focus on the CAPP algorithms.

The results of this study demonstrate that this is a very viable option for industry production of simple geometries produced by turning operations. It is recommended that more work be carried out to better integrate the system into one standalone program which has the capability of working with more complicated geometries. It is also recommended to see the possibility of full automation with different CAM programs.

Keywords: Automated CNC-code generation, CAD/CAM, CAPP

Contents

1	Introduction	5
1.1	Background	5
1.2	Objectives	5
1.3	Scope and Operations	6
1.4	Thesis Format	6
2	Theory Review	8
2.1	Turning	8
2.1.1	Metal Cutting and Chip Formation	8
2.1.2	Parameters	10
2.1.3	Turning Operations	12
2.1.4	Stages of the Turning Process	13
2.1.5	Turning Strategy	14
2.2	CATIA V5	16
2.3	Visual Basic for Applications	17
2.4	Catia V5 Programming	18
2.4.1	Catia Machining Process Modelling	19
2.4.2	Catia Programming within Excel	23
3	Program Description	25
3.1	Catalog	26
3.2	Process Planning	28
3.3	Geometry Production	28
3.4	Toolholder and Insert Selection	28
3.5	Exportation to Catia	29
3.6	Report Production	29
4	Program Demonstration	30
5	Conclusion	38
5.1	Limitations	38
5.2	Recommendations	38
5.3	Reflections	39
5.4	Final Conclusion	39

Terminology and Concepts

API- Application Programming Interface

CAD- Computer Aided Drawing

CAE- Computer Aided Engineering

CAM- Computer Aided Manufacturing

CAPP- Computer Aided Process Planning

CNC- Computer Numerical Control

DLL- Dynamic Linked Library

IDE- Integrated Development Environment

NC- Numerical Control

OOP- Object Oriented Programming

RAD- Rapid Application Development

UDF- User Defined Function

VB- Visual Basic

VBA- Visual Basic for Applications

1 Introduction

This thesis investigates the feasibility of automating the production of turning simulations of simple geometry components within Catia v5 using CAPP algorithms with VBA as the programming language and Excel as the host program. The purpose of creating the turning simulation instead of just generating the NC code is that it allows the user to verify that the program's output is suitable and to have the ability to adjust any parameters as desired in a way which does not require fine-tuning the actual NC code. Turning simulations are something that Catia does very well, but it is not something that has yet been able to be automated. While many companies, universities, and research groups are attempting to achieve completely automated CAPP, it is still outside of our grasp.

1.1 Background

There are an abundance of CAD, CAM, and CAE programs available on the market, yet there has been no single commercially successful CAPP program thus far. There have been many that have been made, with varying degrees of success and accuracy. These programs have not yet taken advantage of other CAD/CAM programs in order to allow the program to only focus on the CAPP; allowing the CAD program to make the geometries while allowing the CAM program to make a simulation of the manufacturing process. The simulation step is the real focus here, because it would allow the user to make any necessary changes and observe that everything is working correctly while still allowing the computer to do the bulk of the calculations.

1.2 Objectives

The objectives of this thesis are simple. First: that the program is able to produce turning simulations within CATIA, and from these simulations is able to produce NC for parts produced by turning with simple geometry. Secondly, that the program is able to produce simulations that will not require much, if any, user input afterwards due to errors within the CAPP logic. The third objective is that the general form of the written program will be easy to adapt to other types of turning operations and other CAM programs, and also

allows the program to be built and expanded upon to accommodate more complicated geometries.

1.3 Scope and Operations

Since this field is still relatively open and unfulfilled it is not realistic that the entire problem of CAPP be solved within the scope of this thesis. Rather, it is more important that the general manner of programming be resolved and that the program is able to solve the presented challenges. It is planned that the program should be able to solve the following; tool selection, tool path selection, and machining parameters selection for turned parts involving rather simple geometry which can be defined as composed of straight lines, notches, chamfers, and radii which are all performed by external turning. The program requires that the user enters in the necessary details which then creates the necessary geometries for the simulation. The program will first use the parameters of the machine, the billet size, and the geometries to determine which operations will be used and in which order. Next using a certain logic scheme the tools will be selected and applied along the proper tool path. After the program is finished running the simulation will be prepared using the parameters given by the program and can be tested for any errors. After the simulation has been verified, it is simply a matter of running the simulation generated through a post-processor corresponding to the turning machine which will be used in the machining process and generate the relevant NC code.

1.4 Thesis Format

In order to carry out this thesis a lot of preparatory work, particularly in the field of machining and programming was carried out. It is necessary to have a very good understanding of machining in order to understand how machining parameters, machining tools, and machining tool paths are selected from raw, geometrical data. A deep knowledge of programming is mandatory in order to be able to generate the proper algorithms which encode this knowledge in a programming language; allowing the process to be automated as well as enabling the different applications used to pass information to and from each other. This thesis contains necessary knowledge and preparatory work in the following fields:

- Turning
- Catia V5
- Computer Programming (VBA)
- VBA programming with Catia V5

The thesis is broken down into three main tasks: Theory Review, Program Description, and Program Demonstration.

Task 1: Theory Review

The theory involved within the thesis is reviewed: this involves every part of the turning process from turning parameters to tool path selection, how Catia works and why it is used so widely, basic programming with VBA, and VBA programming within Catia V5.

Task 2: Program Description

This section describes the thought process behind the formation of the program and how the program completes its tasks.

Task 3: Program Demonstration

The final selection goes through an example of the program in use and demonstrates the final output which the system gives using a real-life example.

The thesis then concludes with observations and a final conclusion.

2 Theory Review

This chapter provides a basic review of turning, Catia V5, computer programming with VBA, and the use of VBA within Catia V5.

2.1 Turning

Turning is a fairly simple metal cutting process which generates cylindrical forms with a single point tool during a process where the workpiece is usually rotating and the tool is stationary. Although the turning process is very simple, it is the most widely used machining process and thus has had lots of optimization which has led to turning having highly specialized tools and operations developed through, and based on, decades of experience (1, p. VI-2).

2.1.1 Metal Cutting and Chip Formation

Metal cutting is an intrinsically difficult process to fully understand. This is because it is such a dynamic process involving considerations such as material of the workpiece, material of the tool, high speeds, temperature, and pressure. Through decades of studies which have produced fantastic theoretical and empirical models describing the process it is possible to bring metal cutting to such a well-defined process that through logical process planning, cleverly designed cutting edges, and proper tool material selection it is possible to produce desired results from a workpiece. This is only possible through the combination of the strengths of both analytical results and theoretical knowledge.

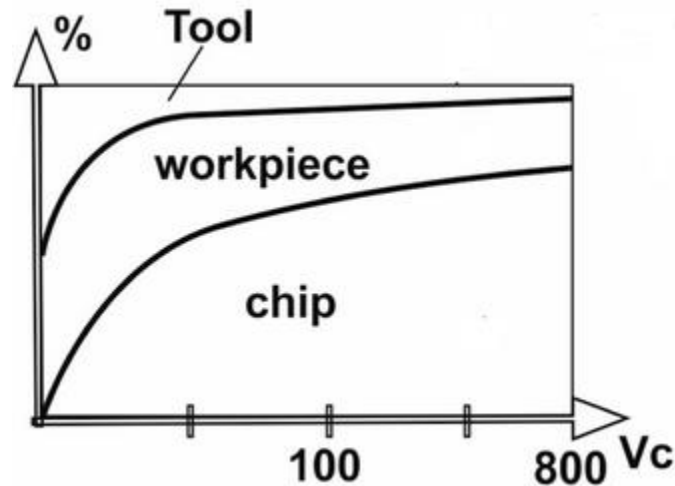


Figure 1. Temperature Dissipation of Metal Cutting Processes

The most critical part of the metal cutting process is the chip formation process. The reason for this is that the removed metal must be pushed away from the cutting area in order to not interfere with the process as well as to carry the majority of the heat produced away from the workpiece: preventing extra thermal stress to the workpiece. Understanding the chip formation process involves an accurate prediction of deformation, temperature, and forces as these are the dominant factors which have an effect upon the chip formation process. Temperatures affect the cutting process itself and can negatively affect the materials of the tool and the workpiece if enough of it is not properly evacuated. The forces affect how much power and strength is necessary within the process.

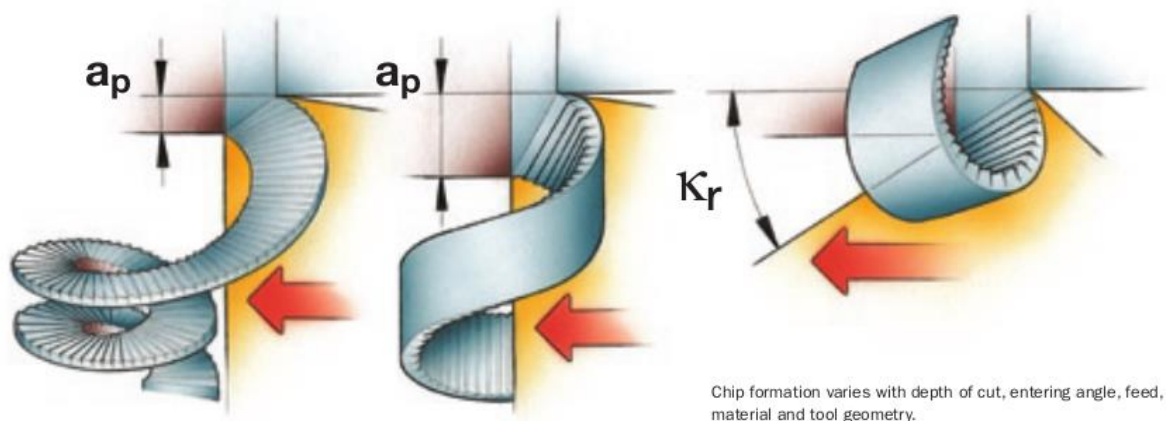


Figure 2. Chip formation factors.

There are many different factors which affect how the chips are actually formed. Factors such as depth of cut, entering angle, feed, material, and tool geometry all have an effect

as noted within Figure 2. While these all have important effects upon the chip formation process, it is sufficient if these are only understood as having an effect and instead following the recommendations of companies which are producing machining tools in order to select the correct tools for each process. These recommendations are based upon empirical data and allow the machinist to concentrate less upon the tool design and chip breaking aspect and more upon the planning of the process as a whole.

(1, p I2-3; 2, p A6-7)

2.1.2 Parameters

There are several parameters in turning which are vital to having the operation go smoothly and without fault. These parameters are: spindle speed, cutting speed, feed speed, cutting depth, entering angle, nose radius, specific cutting force, metal removal rate, and power consumption. Selection of these parameters generally comes from tool manufacturers' recommendations, but it is important to understand these concepts.

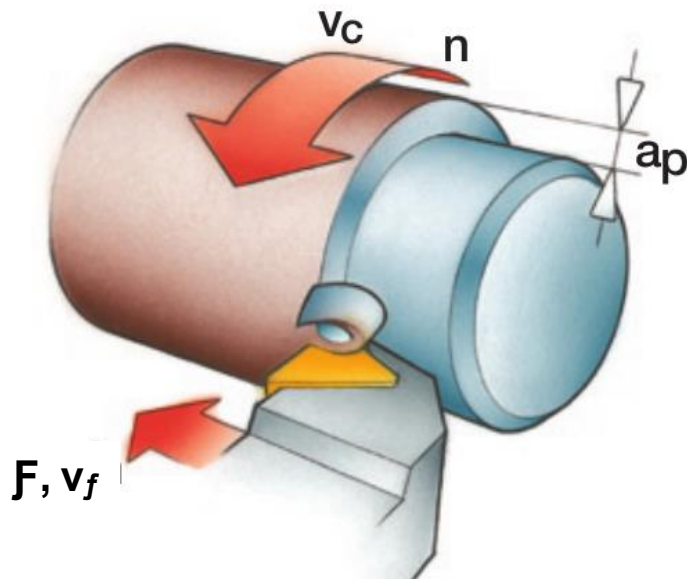


Figure 3. Example of V_c , f , a_p , and n .

Spindle speed (n) is defined as the speed at which the workpiece rotates within the lathe and is given in rotations per minute (rpm).

Cutting speed (V_c) is the speed at which the cutting edge of the tool is moving along the workpiece. Cutting speed generally stays constant, but in operations moving towards or away from the center of the workpiece it can change as it is dependent on the diameter of the cut. Cutting speed is expressed in meters per minute (m/min) and is defined by the following equation.

$$V_c = \frac{D \times \pi \times n}{1000} \quad (1)$$

Feed speed (v_f) is the speed at which the machine feeds the tool along at. Feed speed is expressed in meters per minute (m/min). Feed per rev (f) is an important subset of feed speed and is the amount of feed per revolution of the workpiece expressed in millimeters per revolution (mm/rev).

Cutting depth (a_p) is the depth of the cut which the tool is making. It is always measured as perpendicular from the feed direction and expressed in millimeters (mm).

Entering angle (κ) is the angle between the direction of the feed and the cutting edge (degrees).

Nose radius (r_ϵ) is the radius that exists on the 'point' of the tool. It exists in order to remove the weakness from the point. The nose radius means that there will always be some kind of radius when there is any type of angle along the finished workpiece. A larger nose radius means that there are less vibrations and a higher feed rate can be used. A smaller nose radius allows the machining process to be of higher quality. Nose radius is expressed in millimeters (mm).

Specific cutting force (k_c) is defined as the force in the cutting direction needed to chip off the certain area being removed. It is generally based off of k_{c1} which is the specific force needed to remove 1 mm². The manufacturers' handbooks generally contain the specific cutting force which is equal to the max area which can be removed. The units used for specific cutting force are Newtons per square millimeter (N/mm²).

Metal Removal Rate (Q) is another important factor. This allows the user to compare the rate at which material is removed for each tool. By comparing the Q for different tools, it can be possible to select a tool which will allow the operation to be finished in less time. Q is calculated by the following formula and displayed in cubic millimeters per minute (mm³/min).10³

$$Q = Vc \times ap \times f \quad (2)$$

Power consumption (P_c) is another important factor to keep track of. This is because often a machine may have a limiting power which can be used and it is not possible to go over this power. This means that sometimes power may be the limiting factor in an operation. Power can be calculated by the following formula and displayed in kilowatts (kW).

$$P_c = \frac{Vc \times ap \times f \times kc}{60000} \quad (3)$$

These parameters are all vital components and if one is incorrect than it can easily throw off the entire operation (1, p VI.2-20).

2.1.3 Turning Operations

In order to simplify the tool applications the basic turning operations will first be described as any other turning operations are combinations of these basic operations. The most basic operation is longitudinal turning which is when the lathe is removing material horizontally. Facing is when the lathe is moving along the vertical face along the end of the workpiece. Profiling is when the lathe is following along whatever profile the desired output in one direction. Plunging is when it is necessary for a tool to 'plunge' into the side of the workpiece creating what is known as a notch (2, p A30).

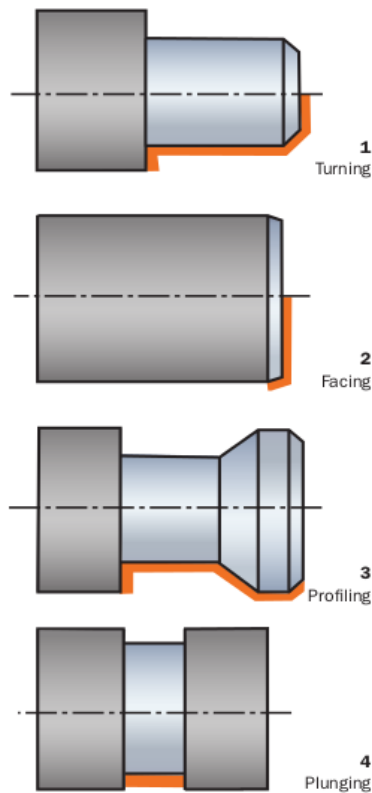


Figure 4. Visual Description of Turning Operations

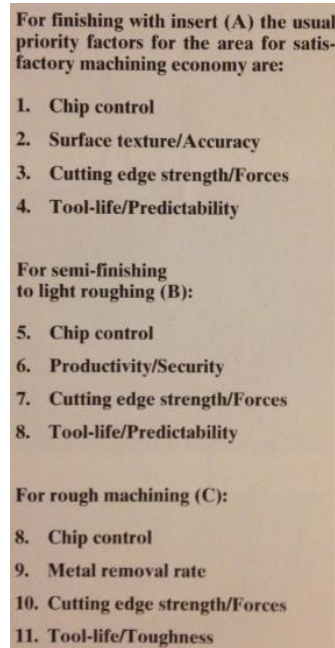
2.1.4 Stages of the Turning Process

There are different stages of almost every turning process. These stages can be broken down into the roughing, semi-finishing, and finishing stages.

The roughing stage consists of when it is the most important to remove as much material as possible in the quickest time possible. When doing roughing it is not seen as important for the surface quality that is being produced to be very accurate or smooth. The name actually derives from the rough surface which it leaves behind. Of course when doing a roughing process it is important to recall that you should always leave sufficient space between the roughing surface and the desired finished surface unless the finished surface desired surface quality is reachable by the roughing process.

The semi-finishing stage is when the tool is still quite productive but has a higher accuracy than the roughing stage and can allow for material to be removed that was too dangerous to be removed during the roughing stage.

The finishing stage consists of the final pass along the part which gives the piece the desired surface texture.



For finishing with insert (A) the usual priority factors for the area for satisfactory machining economy are:
1. Chip control
2. Surface texture/Accuracy
3. Cutting edge strength/Forces
4. Tool-life/Predictability
For semi-finishing to light roughing (B):
5. Chip control
6. Productivity/Security
7. Cutting edge strength/Forces
8. Tool-life/Predictability
For rough machining (C):
8. Chip control
9. Metal removal rate
10. Cutting edge strength/Forces
11. Tool-life/Toughness

Figure 5. Comparison of Stage Priority Factors

As demonstrated by Figure 5, the secondary feature of each stage is according to the process, but chip control is still the number one priority for any metal cutting process due to the high percentage of heat staying within the chips (1, p 119).

2.1.5 Turning Strategy

One of the key components of having a competitive business, especially a business with a lot of competition, is to keep costs low per worked piece. It thus serves the end line of any business if they can cut worked hours per piece as they will also cut cost at the same time; thus boosting competitiveness. Optimization of process planning is an aspect of machining in which a company can improve its profit margin by having an increased production rate while at the same time not increasing use of expensive tools too much.

Sandvik Coronant (1, p VI-37) describes the following as the main factors which influence the tools application during the turning process:

1. Workpiece material- machinability, condition, properties, etc.
2. Workpiece design- shape, dimensions and working allowance.
3. Limitations- accuracy, surface texture, etc.
4. Machine- type, power, condition and specifications.
5. Stability- from cutting edge to foundation.
6. Set-up- accessibility, holding, changing.
7. Tool Program- the right tool.
8. Performance- cutting data, tool-life and economics.
9. Quality- tool delivery and service.

There are certainly many different methods to use while selecting tools, tool holders, and parameters, but the general method used in the thesis is as charted out in Figure 6.

The general idea of selection is quite simple as there are almost always many different combinations of inserts, toolholders, and cutting parameters which will end up giving the desired results. The only difference is that some will give the results in a much quicker manner than the others and most likely end up being cheaper when applied to many pieces even though the inserts and toolholders themselves may end up being much more expensive.

The process can be described as follows. First of all, gather all of the necessary data for the part. Second, using critical thinking decide which type of operation will be done and whether positive or negative geometry will be used. This step can either be extremely obvious or quite complicated: the general idea is to limit the amount of tool changes as they usually take quite a bit of time and to do all of the work which is possible with the tool while it is in use. The next step is to look at the operation desired and choose the insert and then the toolholder taking into account the following factors: shape, size, geometry, application type, nose radius, and grade. After this the cutting parameters can be selected according to the manufacturer's recommendations. Then it is always advisable to check that all of the values resulting from these parameters are within the acceptable values for

the machine. If they are not acceptable, then maybe it is possible to slightly modify some of the cutting values, or it may be necessary to change the insert.

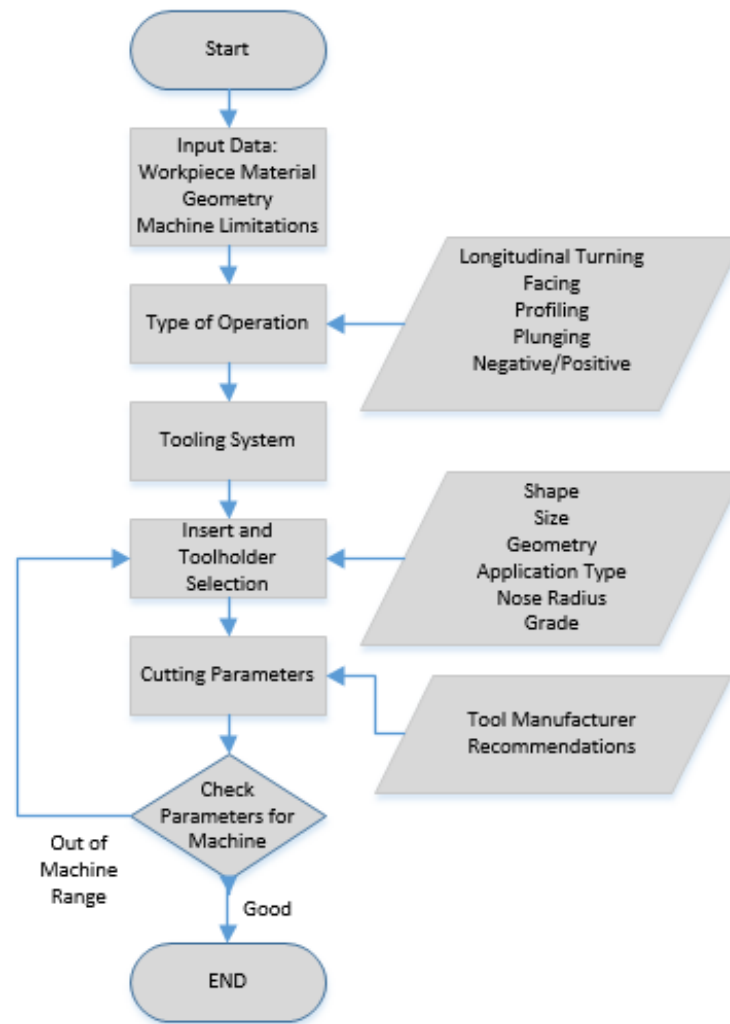


Figure 6. Workflow for Tool, Toolholder, and Cutting Parameters Selection.

2.2 CATIA V5

Catia (Computer Aided Three-dimensional Interactive Application) developed by Dassault Systemes, a French company, is a CAD/CAM/CAE commercial software suite. Catia has been developed in the C++ programming language and is the foundation of Dassault Systemes. Catia was first developed around the start of the 1980's with an intent of using it in the aerospace industry.

Catia has been referred to as a PLM (Project Lifecycle Management) system as the scope of the program is so large it can be used to model every part of the manufacturing process of a product. An example of Catia use could be the following: first the product itself is modelled, then the product is tested, then the manufacturing processes are simulated, and finally the factory itself is simulated. The entire process is done and all of the documentation is properly contained within Catia. Catia is used for many different types of disciplines and can be used for mechanical, electrical, fluid, and systems engineering. This versatility has led to Catia being used in most engineering industries. Catia is widely used in the automotive, aerospace, industrial equipment, plant design, packaged goods, life science, civil, architecture, process power, petroleum, and other related industries.

(3)

2.3 Visual Basic for Applications

For decades, BASIC has been used as a programming language as a 'basic' interface between man and machine and is a relatively low-level programming language. BASIC eventually evolved into VB which was used as a RAD and had the capacity of quickly creating applications which could run independently of other programs. VB forked into two directions, one of which was .NET which is still used today to create standalone applications, and the other which was VBA and transferred most of the functionality of VB into an easy to use language which is hosted inside of other applications. Examples of applications which can host this language are most of the wide range of Microsoft Office products available , notably Excel, and other applications such as CATIA V5 (4).

VBA, as noted, contains many of the elements of VB; such as an IDE which is almost the same as the original VB IDE. This contains important components such as a debugging window, a properties window, and so on (5). Inside of this environment it is possible for the developer to interact and access the object model of not only the host application, but other applications as well allowing VBA to work across several independent applications. The introduction of VBA allowed several other macro languages used within applications for simple automation to be retired as VBA does, in general, a quicker, better job while it is quite user-friendly as well.

VBA has the ability for the user to create UDFs, to automate simple tasks, and to be able to use DLLs to access low-level functions such as the Windows API. The user will create different macros which contain all of the actual functionality but do need to be within another application in order to run. Some of the common uses of VBA are as follows:

Automating a Task

Automating Repetitive Tasks

Creating Custom Sequences

Custom Commands

Creating Applications (Macro Based)

The most important thing to realize about VBA is that it is an event driven language. This means that in order for the program to start doing its work it must be triggered by some certain event. If the event driving the macro is not run, then the macro will simply stay silent. This means that it is important for the developer to ensure that the macro will always be triggered when the macro should start running (5).

2.4 Catia V5 Programming

Programming with Catia V5 could be done for many different reasons. For instance, a company could want to take information from either word or excel documents, write information to these applications, automate tasks, or use the computing power to do calculations and generate outputs within Catia automatically.

There are three different programming language options for Catia V5: VBscript, VBA, and VB6. VBscript is a simple subset of VBA which is the only Catia programming language usable in both Unix and Windows operating systems. It is used for very simple macros. The downsides of VBscript is that it is quite difficult to program in because its editor is very weak, it is sequential programming, and the user interface tools are not very good. VBA is, as mentioned before, a subset of BASIC which needs to be hosted within an application. The positives of VBA is that the IDE is quite good, the programming is event based, it allows collaboration with other VBA applications, and the UI (User Interface) is good. The downsides are that the security is not very strong and it is difficult to export the

program. VB6 has all of the advantages of VBA as well as code protection, the possibility to create DLLs, and the possibility to create servers. The only downside is that the software requires another license purchase from Microsoft and that VB6 is very outdated nowadays and does not work very well on a newer Microsoft OS.

Within this thesis, VBA was utilized. There are many reasons for this selection. First of all, as the thesis is simply the testing of an idea and not actually the full implementation the functionality of VBA will be sufficient. VBscript is simply too basic to be used in any easy manner, plus it lacks the functionality to connect to other applications outside of Catia. VB6 would be a good fit, other than the fact that VB6 is simply not available from Microsoft anymore because of the abandonment of its use in favor of .NET which is simply an expanded version of VB but is unfortunately not completely compatible with Catia v5. If the CAD/CAM program being used was Catia V6 then .NET would definitely be the best choice for the programming language. However, Catia v5 was utilized and that left one real choice for this thesis: VBA. (6)

2.4.1 Catia Machining Process Modelling

Now before the programming goes into detail, it is necessary for an explanation of how to model the machining process within Catia.

First of all, before anything else gets done it is necessary for the user to have a 3D model of the finished part, a 3D model of the billet being used, and a preselection of the details of the manufacturing process. For the 3D models it is always recommended to have the Z axis pointing away from the chuck so that the default axis for the machine does not need to be adjusted. It is also recommended to draw an extra sketch upon the 3D model of the finished part for each phase of the manufacturing process as this makes the tool path selection an extremely simple process. Once the 3D models are prepared it is necessary to put them all into one product and line them up in the way the finished product will be relative to the billet. Then it is necessary to put the origin of the product where the desired (0,0,0) point of the operation is. If it has not been specified for any operations it is put in the center of the billet on the face which faces away from the chuck. An example product is shown in Figure 7.

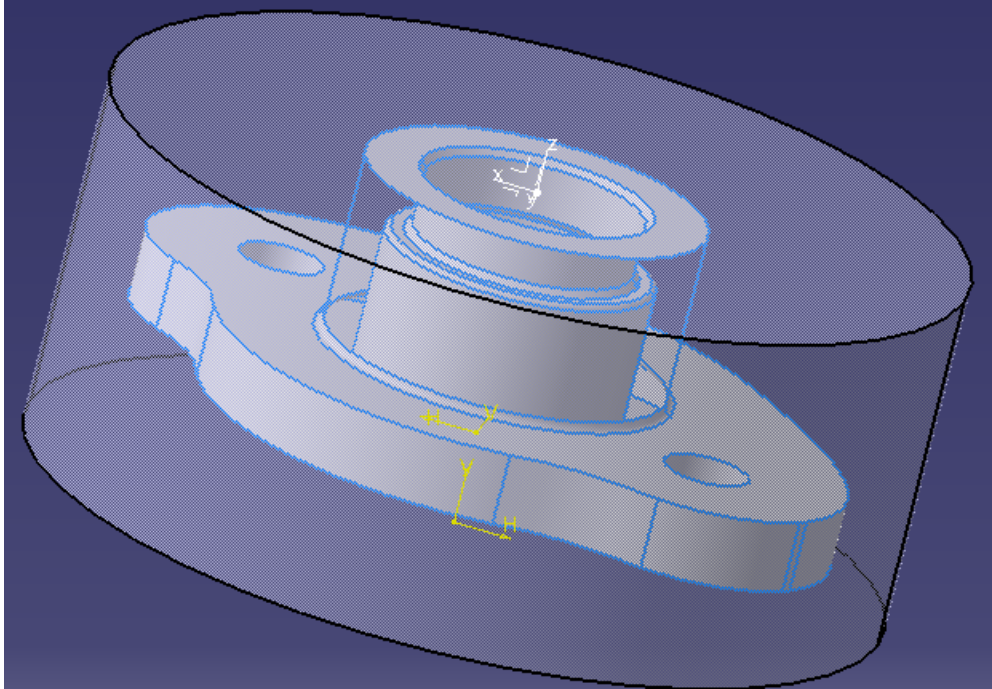


Figure 7. Product Ready for Manufacturing Process. Note the Origin on the Billet Face.

Once the product is ready the user clicks on start, then machining, and finally lathe machining. This opens up a process. The process has three main parts within what it calls the PPR and these are processes, products, and resources. This is because it may be necessary to have more than one process to finish a part and the product may need to be changed slightly between each process to include extra sketches. The resources include whatever resources are used in the manufacturing process.

The next step is to click on Part Operation.1 and add the relevant information. This includes machine type, default machining axis, design part, stock, tool changing point, and so on.

Then once this information is all filled out the user can select what type of operation is desired. The options are roughing, grooving, recessing, profile finishing, groove finishing, threading, ramp roughing, recess roughing, and drilling operations. Inside each different type of operation the options are roughly the same.

First of all the stock and the element to be machined are selected. This is when the sketches are extremely useful because they can be selected as the element to be

machined and the program will follow their geometries. Things such as the offset can be set so that the geometry used can be as close to the actual finished part as possible.

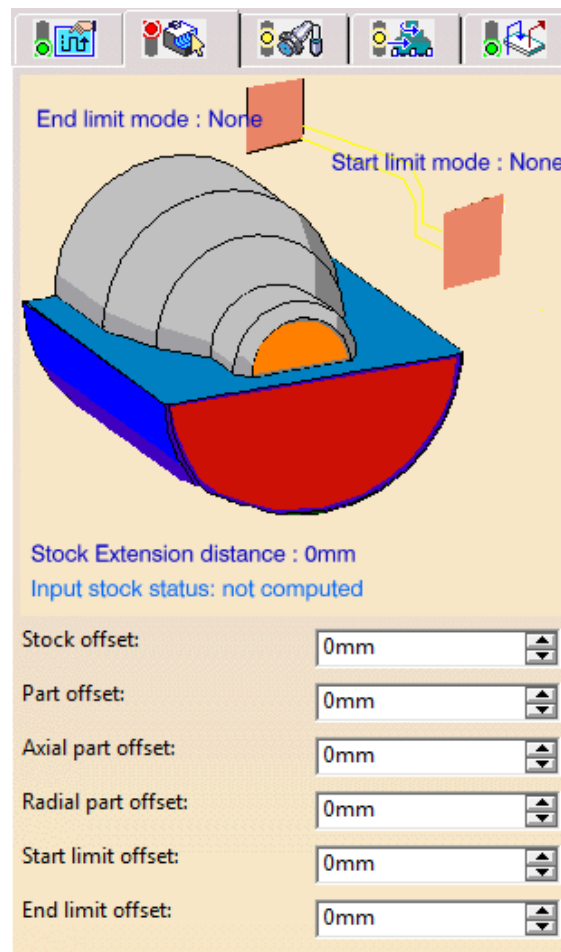


Figure 8. Tool Path Selection

The strategy can be selected next which gives the options as to how the tool will be directed along the piece, what the depth of cut will be, what the tool compensation will be, and so on.

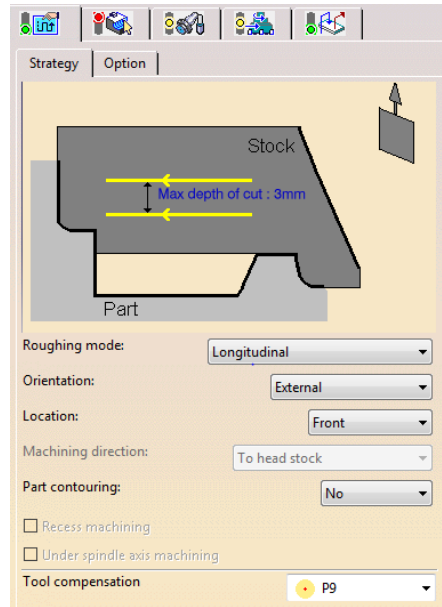


Figure 9. Operation Strategy

Next is the option to select the toolholder and the insert type. During this stage it is also necessary to enter in the feed and speed for the insert. The next window then can be set to run automatically off of the information entered during this part of the process.

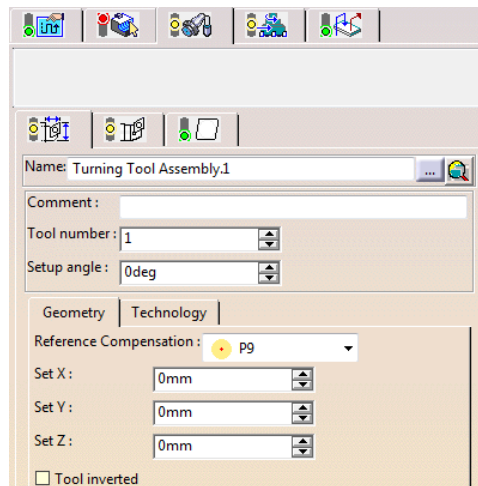


Figure 10. Toolholder, Insert, and Cutting Parameter Selection

The final screen allows the user to set up macros enabling the user to predefine methods of tool entry/exit. This allows the user to specifically design how the operation will work and ensure that the operation will not collide with anything during the entry/exit.

After all of the operations have been completed it is possible to simulate the operation from the tool path because the NC code has not yet been generated. This thesis will focus on getting to this point as the NC code generation simply requires setting the correct postprocessor for the machine being used and clicking generate NC code. (7)

2.4.2 Catia Programming within Excel

It is not completely necessary for the programming to be done within Excel but it was chosen for this thesis because it allows the selections to be made before everything else is done as the insert and toolholder catalogs are also contained within Excel. This means that only Excel needs to be run at the beginning and Excel will do the heavy work before opening Catia and exporting all of the selected data to the machining process. As Catia uses a lot of computing power it is advisable to only open it when it is needed. Before anything else is done it is mandatory for the user to go to the References within Excel and enable anything coming from Catia as this allows Catia functions, properties, and types to be used.

Catia uses OOP which means it consists of objects which all have certain attributes or data that make it a certain, unique object. Each object also can do certain actions which are called methods. The key point of object oriented programming is the ability to create so-called classes under which similar things can be placed. For instance under the class of computer there could be objects such as Apple, Microsoft, Asus, Toshiba, etc.

An object can be defined as an entity. Examples within Catia include things such as lines, points, shafts, and so on. Objects utilize aggregation to set the relations between different objects. For instance a line is simply an aggregation of the two points which make it up, while a point is simply the aggregation of the plane it is set upon and its coordinates.

Objects within Catia can be worked upon by using two different ways: Property and Method. The property is the characteristic of an object and the method is an action of an object. An example of a property would be the name it is given such as Point.1. An action is obviously when some form of action happens such as when a sketch opens via sketch.Open.

Another great help within Catia is there is a Macro Recorder tool which records and exports to code everything done within Catia while the tool is running. This is helpful in gaining the names of different objects, properties, and methods. However, it is not really recommended to use code generated by the Macro Recorder within personally developed code as it has a different set of preferences than a human developer and may change names of objects, may only work in certain situations, and numerous other issues which can come from trying to mix different developers' code together when they are both using a completely different system (6).

3 Program Description

This program's purpose is to demonstrate the possibility of using CAPP to producing turning simulations within Catia V5 and thus will have some limitations. In the Observations section there will be descriptions of possible manners of implementing the program with less limitations. First of all, since the catalogs all need to be created by hand, there will be a limited number of toolholders and inserts available. This is because each catalog coming from the manufacturer has 1000+ pages and this is simply too much information for one person to transfer for the sake of a thesis project. Second, in order to allow the catalog to be smaller only one type of material was selected as the type of material being machined. Once again, this is because otherwise the data transfer would be overwhelming. Third, the geometry used will be fairly simple. Of course it is possible to adjust this program to deal with more complicated geometries, but that is something to build towards and not to start with. These limitations are only for the extent of this thesis and are not things that are impossible with this type of program, but simply would require more than one developer and/or an extended period of time in order to produce macros which can dig through catalogs or deal with complicated geometries.

The program can be broken down into several semi-independent sections which each contain a critical part of the overall program. Each section will be thoroughly explained and documented. The sections are as follows:

- Catalog
- Process Planning
- Geometry Production
- Tool holder and Insert Selection
- Excel exportation to Catia
- Report Production

The general workflow of the program is as demonstrated in Figure 11.

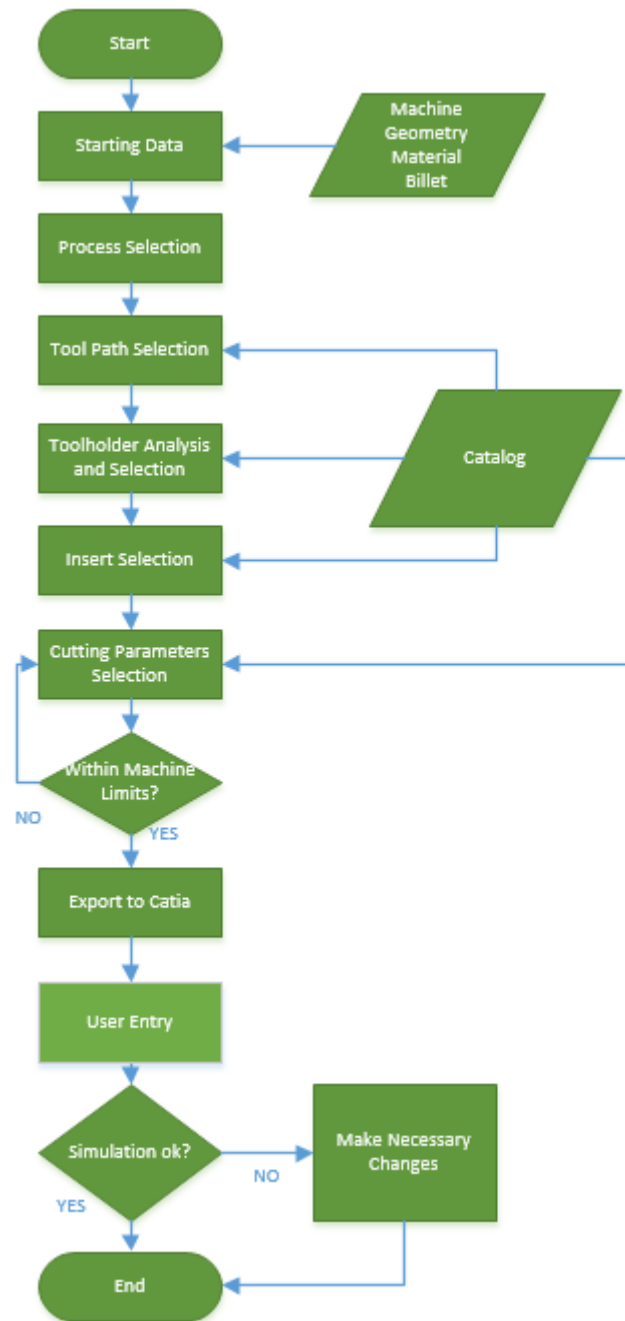


Figure 11. Program Workflow

3.1 Catalog

In order for the program to have the necessary information regarding toolholders, inserts, and cutting parameters it is necessary for the user to create a catalog which Catia can

use and contains all of the necessary items for the operations. This catalog has been created in Excel and uploaded into Catia. If this program was analyzing the process for different materials the recommended cutting parameters would have to be actively linked to the VBA macro and then the catalog would have to be altered within Catia to permit the correct cutting parameters to be used. As mentioned earlier, this catalog is a bit simpler just for purposes of time saving. There are well defined processes for making tool catalogs within Catia and the process used for the purpose of this thesis does not deviate greatly from this normalized process. It involves making a CSV sheet and uploading it into Catia through VBScript and then placing the created library where it can be accessed upon start-up by Catia. A brief look at the tool catalog is shown in Figure 12.

CHAPTER	NC_RESOURCES			
Keywords	Name			
Types	String			
NC_TOOLS	NC_TOOLS			
NC_INSERTS	NC_INSERTS			
END				
CHAPTER	NC_TOOLS			
Keywords	Name			
Types	String			
MfgExternalTool	MfgExternalTool			
MfgDrillTool	MfgDrillTool			
MfgGrooveExternalTool	MfgGrooveExternalTool			
END				
CHAPTER	NC_INSERTS			
Keywords	Name			
Types	String			
MfgRoundInsert	MfgRoundInsert			
MfgDiamondInsert	MfgDiamondInsert			
MfgGrooveInsert	MfgGrooveInsert			
END				
ENDCHAPTER	MfgExternalTool			
Keywords	MFG_NAME	MFG_COMMENT	MFG_HAND_STYLE	MFG_KAPPA_R
Types	String	String	String	deg
External Lathe Holder	DRSNL 2525M 12	DRSNRL 2525M 12 R	LEFT_HAND	
External Lathe Holder	DRSNR 2525M 12	DRSNR 2525M 12 R	RIGHT_HAND	
External Lathe Holder	DRSNL 4040S 25	DRSNL 4040S 25 R	LEFT_HAND	
External Lathe Holder	DRSNR 4040S 25	DRSNR 4040S 25 R	RIGHT_HAND	
External Lathe Holder	C8-DVJL-55080-13	C8-DVJL-55080-13	LEFT_HAND	93
External Lathe Holder	C8-DVJR-55080-13	C8-DVJR-55080-13	RIGHT_HAND	93

Figure 12. Part of the tool catalog

3.2 Process Planning

The process planning part of this program leans heavily upon the strengths of Catia. Since Catia has many machining safeguards in place, such as collision protection, it is possible to create a rough outline and take advantage of the work which Catia does for the program. For instance it is always normal that the first operation will be a roughing operation. The program simply produces a roughing sketch which serves as the tool path and then selects toolholder and insert according to the geometry of the final piece and how much can actually be machined during the roughing phase. The same general process can be used for the finishing process; a simple sketch used as the path for the machining to follow. For the notch feature the logic is exactly the same. There are of course many different options and variables which may improve the machining process but they are very easily entered via the Catia Lathe Machining Workbench.

3.3 Geometry Production

The original input geometry production is quite simple and always starts at the (0,0,0) point and moves lengthwise towards the positive direction of the Z-axis which simplifies our exportation to Catia afterwards. The input gives a section view of the part which will then be revolved around the main axis creating a cylindrical part.

The geometrical input is as follows. First the user is asked to specify billet length, billet diameter, and finished piece length. These are all mandatory for the geometry creation. Then the user enters information regarding how the geometry will change. There are possibilities of line, chamfer, radius, and notch features. Inside each feature the code is outputting the correct points to the Excel worksheet and they are combined into lines when exported to Catia. There is also the possibility to edit the values which have already been entered through the userform which makes it quite userfriendly.

3.4 Toolholder and Insert Selection

Once the process has been selected it is a simple manner to select the toolholder and the insert. The toolholder must be selected first, and simply involves matching the process with the correct sort of toolholder. The program for this thesis will select one of each

toolholder that is possible with the geometry of the process and then select the insert which will have the fastest machining time with this insert. This generally means the part which has the largest cutting depth, but not always. Using practical knowledge the program will only use tools that are effective for the geometries demanded as changing tools too often can mean a longer machining time. Effectively this means that the program looks at the volume to be machined and makes a case on what type of tool should be selected according to how well that certain tool can machine that area.

3.5 Exportation to Catia

The exportation to Catia will only occur once all of the values that are necessary have been preselected for each process. Unfortunately, there is no accessible library for the lathe machining workbench within Catia. This means that everything will be produced, up until the point where the machining workbench would be opened. The user can utilize the report which is produced in order to enter the relevant values in the machining workbench. In the Program Demonstration chapter the entire process will be shown as one without any real difficulties. This also allows some human knowledge to come within the process and ensure a highly optimized process is produced on the first time.

3.6 Report Production

The production of a final report is a simple process as the relevant parameters were already resolved before any exportation to Catia. The report contains a list of toolholders and inserts used, processes, power used, and all of the relevant machining values. This is the necessary information for any machining process so although the values will not be entered automatically it spares the user from having to sift through the massive metal cutting tool catalogs in order to collect all of the relevant data.

4 Program Demonstration

This chapter will mostly just consist of pictures demonstrating the process which must be gone through in order for the simulation to be produced

The first step is to introduce the data. Figure 13 demonstrates how the program writes the necessary points to Excel. First the user must enter in all the information above the Geometry Entry box. Then the user can select which kind of feature to introduce and follow the instructions given. The Z and X coordinates are placed into an Excel sheet but are also able to be edited within the userform by clicking the edit button, making the necessary change in the Points section, and then clicking save edit. This will update all coordinates which are affected by the change both within the userform and within the Excel sheet as well.

Geometry Enter

Machine Selection: Catia predetermined(max n=10000rpm)

Material Selection: Aluminum alloy

Billet Size (diameter) mm: 75

Billet Size (length) mm: 170

Finish Quality: ISO 2768-mK

Finished Piece Length (mm): 165

Instructions

Geometry Entry: Notch

15 Length/Radii Send

5 Angle/Depth Center

Export1

Export2

Edit

Save Edit

Close

Points

Z coord.	X coord.	Feature	Special
0	0		
2.5	0		
2.5000000	15		
37.5	15		
37.5000000	34.4746		
38.350903	35		
0	0		
0	0		
0	0		
0	0		
0	0		
0	0		
0	0		
0	0		
0	0		
0	0		

Figure 13. Excel Userform

Figure 14 demonstrates the points which have been uploaded into the Excel file. It also shows some of the extra cells which are filled that give information on features which will be enacted once the data is sent to Catia. The third column represents origin points for instance, while the fourth represents a radius value.

0	0	1						
2.5	0							
2.5	15							
37.5	15							
37.5	34.47468							
38.3509	35		0.850904	CommandButton1				
77.5	35							1
77.5	30							1
92.5	30							1
92.5	35							1
117.5	35	2						
117.5	20						1	
147.5	20							
147.5	10							
167.5	10							
167.5	0							
170	0	1						

Figure 14. Excel Points

Figure 15 is the geometrical output. This output contains everything needed for the piece to be placed in the lathe machining workbench and be prepared for simulation and eventual NC code output. It consists of the final piece placed within the billet which it will be machined out of. This way it is possible for the machining simulation to keep track of what has been machined and what has not been.

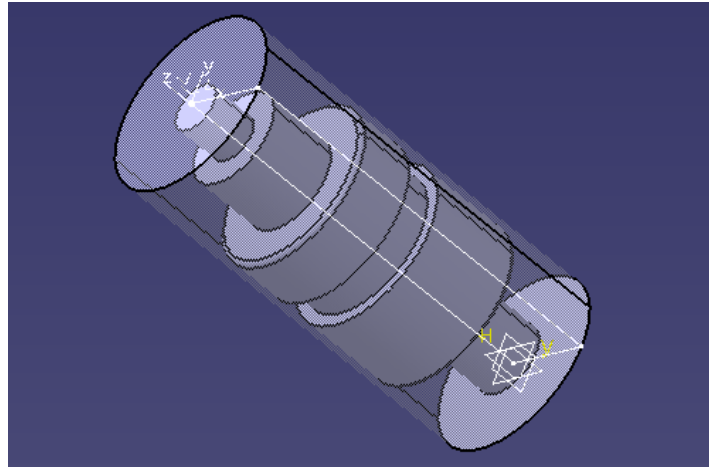


Figure 15. Prepared Product: Ready for Machining

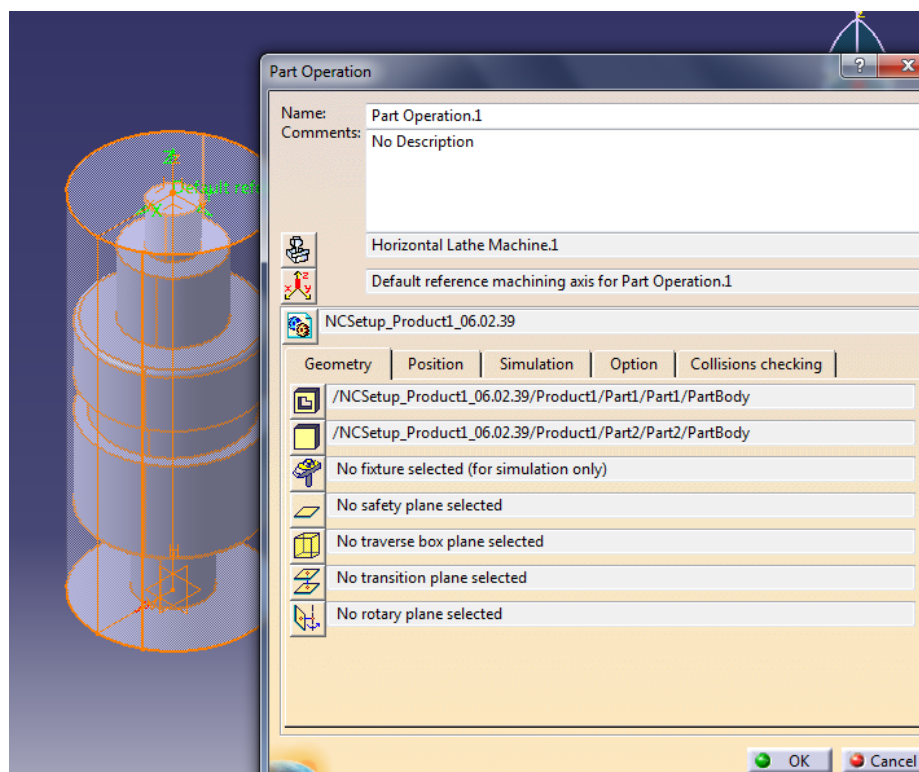


Figure 16. Part Operation 1 Preparation

Once the lathe machining workbench is open the user must insert a part operation. From the part operation setup screen they will select the machine to be used. In this case the Catia predetermined Horizontal Lathe Machine is being used. Then the user places the

default reference machining axis upon the other axis which is shown in the product already. This is very important as it tells the machine where the part is and where to change tools. If the axis is accidentally on the wrong end then the machine may designate a tool changing point which is inside the part's geometry. After the Origin is selected; Part 1 is inserted as the design piece and Part 2 as the billet. After these all have been entered then it is already time to start entering the machining parameters.

Type	Name	Nose Radius	V(m/s)	Operation Type	depth (mm)	feed(mm/rev)	kc(n/mm²)	Toolholder Compensation	Power (Kv MMR (cm³/min)	
55degree	TR-DC1308-M	1.2	2000	Rough	2	0.3	400	C8-DDHL-1 P3	8	1200
35degree	TR-VB1308-F	0.4	1900	Finish	1	0.24	400	C8-DVJL-5 P3	3.04	456
Round	RCMX 25 07 00	25	2000	Rough	3	0.23	400	DRSNL 404 P8	9.2	1380
35degree	TR-VB1308-F	0.4	1900	Finish	1	0.24	400	C8-DVJL-5 P3	3.04	456
Groove	N123E2-0500-000	0.2	1900	Groove	<19	0.1	400	LF123E17- P8	3	760

Figure 17. Excel Machining Report

This report is produced by clicking the button export 2 and it shows the two part operations which will be needed to manufacture the finished piece. There may be some pieces which only need to be manufactured from one side and for those it is advised to simply mark the center point on the other end of the billet so that the program does not want to machine anything on that end. It is also important to realize that the user will need to enter in the correct tool compensation when selecting the toolholder and insert or else the piece will not be machined in the anticipated manner. The user should also do a quick computation of the tool path just for a quick verification that the process is proceeding correctly.

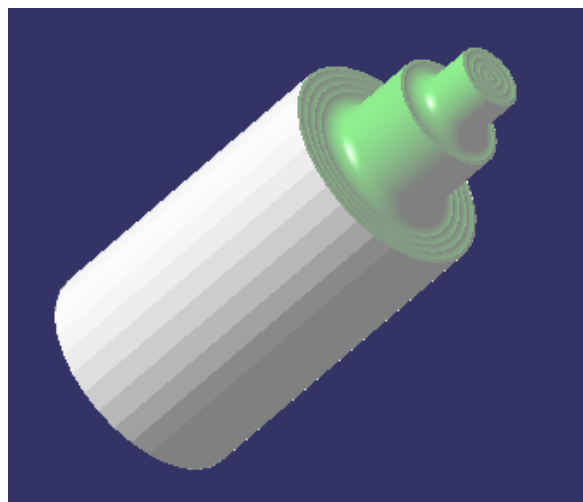


Figure 18. Roughing Process

Once all of the operations have been entered into the program then the user can right click on the part operation and click on 'Start Video Simulation Using Tool Path'. This will go through each component of the part operation and enable the user to verify that the desired output is being produced. This is the main point of producing the simulations; the ability to be able to verify that all parts of the machining operation are going smoothly.

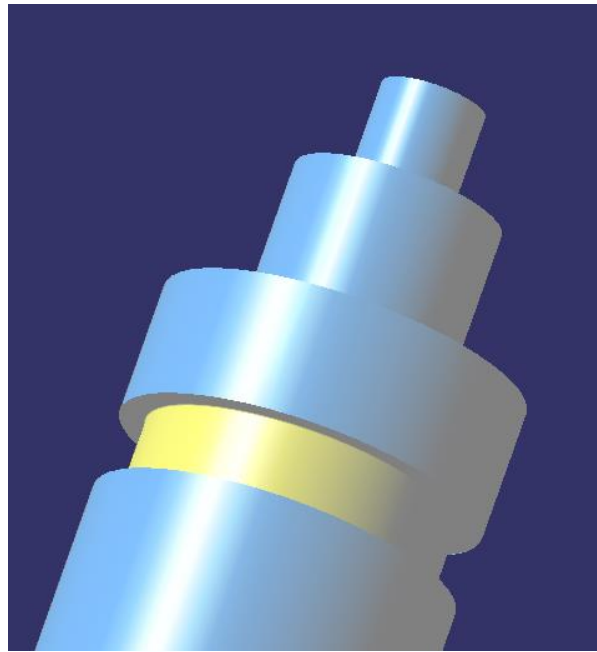


Figure 19. Part Operation 1 Finalized

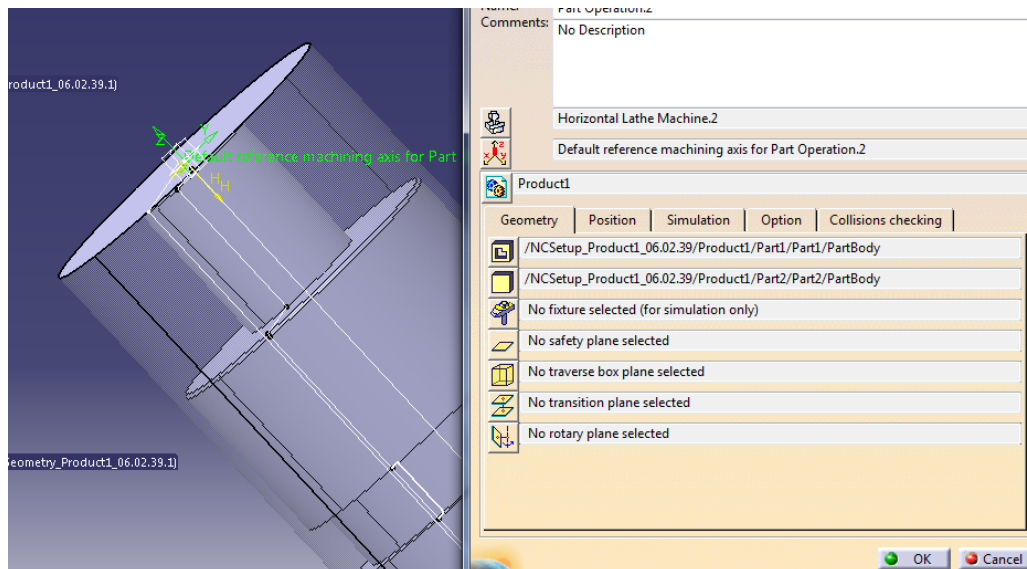


Figure 20. Part Operation 2 Preparation

When setting up the second side everything is identical as when setting up the first side, except that when setting up the axis system it is critical that one marks the z axis as negative so that the tool changing point is once again in the correct position away from the piece.

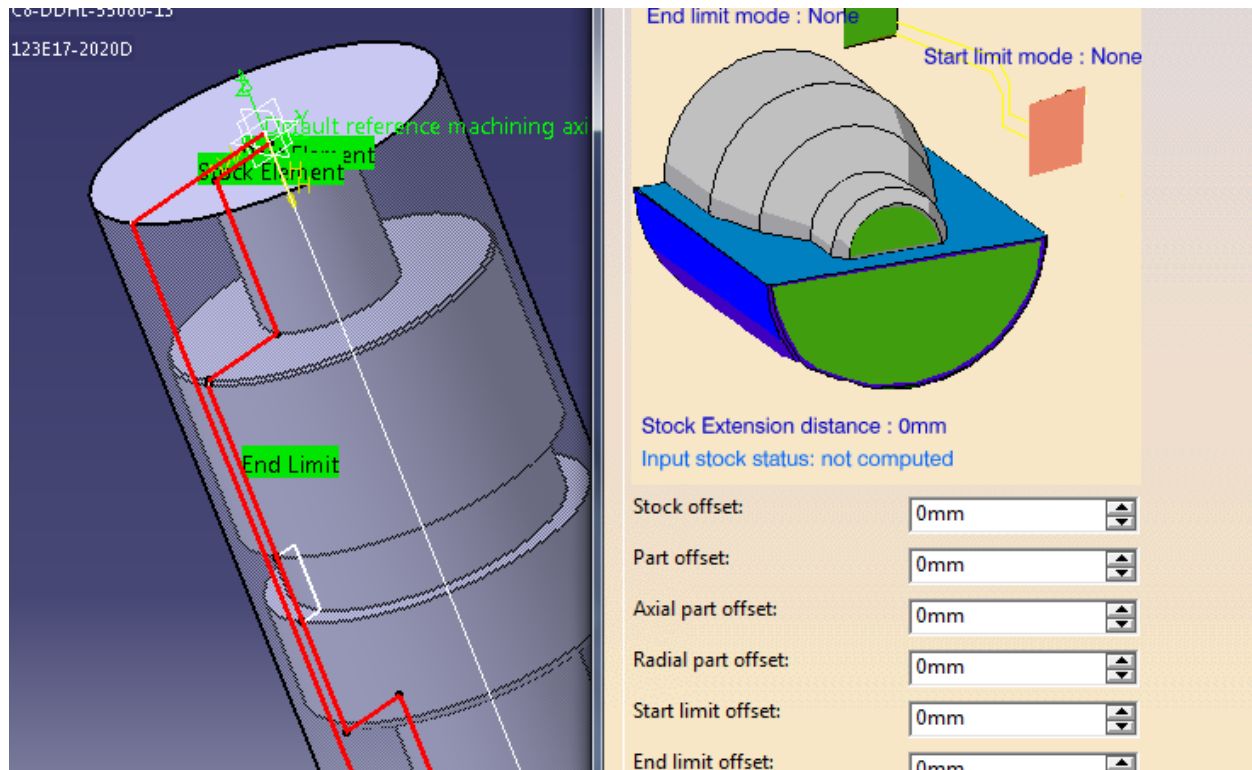


Figure 21. Tool Path Selection Using Pre-Defined Sketches

Here is an example of how easy it is to set up the machining processes when the sketches have already been defined. One simply selects the sketches for both the stock element and the input stock and then marks an end limit. The end limit can be defined as having a certain offset so that it is not necessary to make any new lines within the sketch. All one does is find the end limit which is perpendicular to the desired end limit and then define the offset as desired.

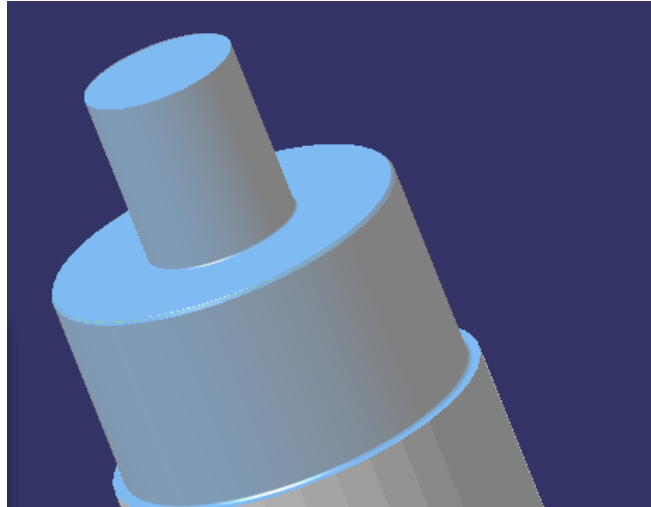


Figure 22. Part Operation 2 Finalized

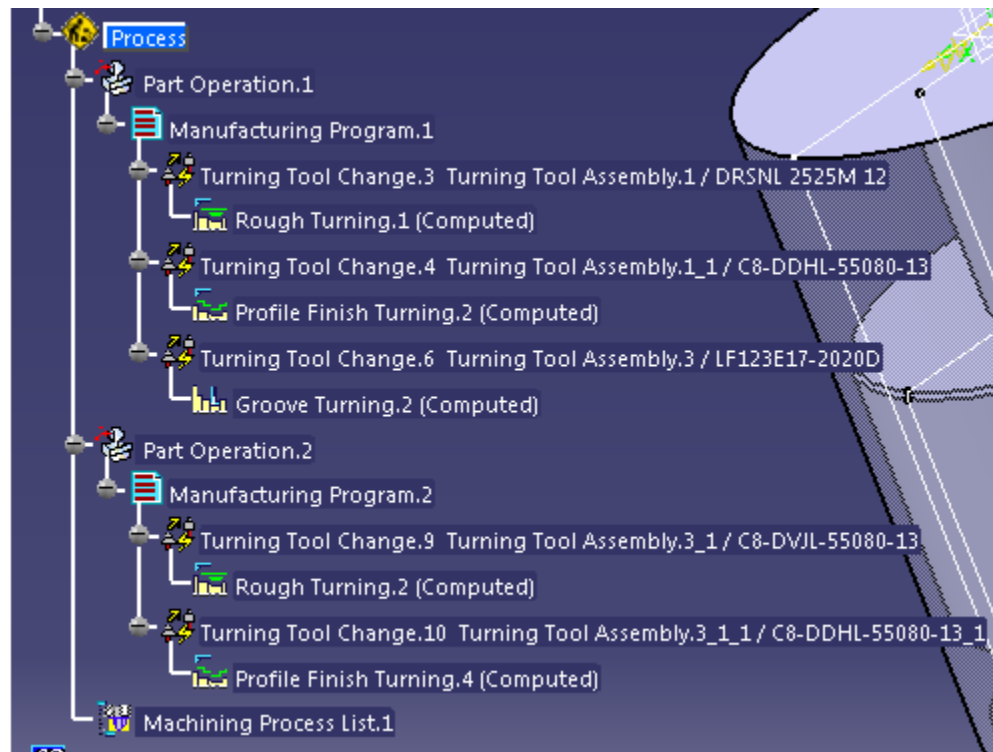


Figure 23. Tool Changes and Machining Operations

If there is multiple changing geometries within an operation then it may be necessary for the user to have two of the roughing and finishing operations with different tool compensations. For simple geometries the part operations should look something similar to Figure 23.

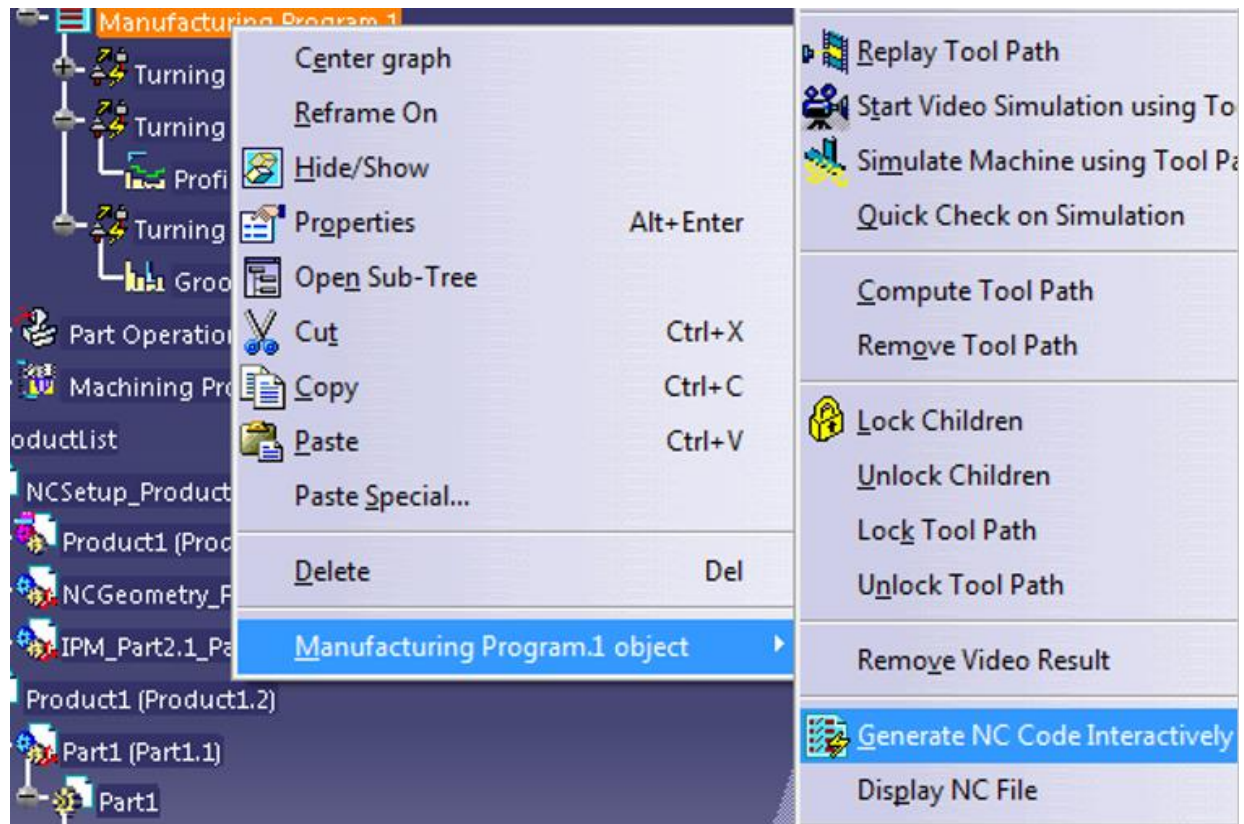


Figure 24. NC Code Generation

As mentioned earlier, producing the NC code once the simulation has been run is a very simple task. With a right click upon the manufacturing program it is possible to get to the menu to generate NC code. With this the purpose of the simulation is done and the part can be ready to be manufactured provided everything has been entered correctly.

5 Conclusion

The conclusion section states the limitations of this project, makes recommendations on how to further improve upon this program, reflects on the program and the process of creating it, and closes with a final conclusion.

5.1 Limitations

One limitation for this project is the fact that the tool catalog is so small. This is something that could easily be remedied. For instance, if a machine shop already knows which inserts and tools they are usually using then it is simple to enter these and add new ones as necessary.

Another issue with this program is it can hang up if the geometry starts to get more complicated and then takes a rather long time to solve the geometries and machining parameters or it will simply end up shutting down Excel. This may have something to do with the programming not being done professionally or by someone with a lot of experience.

Obviously the biggest issue with this project has been the fact that it is not possible to write VBA macros that allow one to manipulate the data within the machining workbench. This has meant that the process planning part has fallen far short of what it could be and the simulation process has not been able to be automated.

5.2 Recommendations

It is recommended that whoever is expanding upon this program expand the catalog and link all the necessary information regarding the cutting parameters.

Another upgrade that would improve the project immensely would be the availability of a way to read geometry and plot points directly from either a 2D drawing or a 3D model as it would really streamline the process.

The process planning situation could definitely be improved since it does not have the Catia capabilities to depend on to perform automatically, but it is suitable for simple

geometries at the moment. If the program is to be expanded to tackle more complicated geometries then it is necessary to take a more exhaustive approach.

It would likely be nicer to be able to program in .Net which would require an upgrade to Catia v6. It would be nice to program in .Net because then the user can have a bit more flexibility and would also be able to create a proper, standalone application. Excel VBA works, but sometimes its limitations are quite obvious and things just do not work correctly. It also requires some very interesting workarounds at points where it just does not have the functions which some other, more advanced programming languages will have.

Another recommendation is to look into using a different CAM program which allows one to fully program within the machining workbench. This could allow the program to be able to be fully fleshed out if the proper CAM program is found which effectively produces high quality machining simulations and that can be fully accessed via programming.

5.3 Reflections

This thesis project has in some ways been disappointing, but in other ways very eye-opening and educational. Despite the author not having much programming experience, everything has been able to be programmed. Although it is certain that some of the solutions are not optimal and could be improved, this programming has helped lead to a different way to look at the machining process as well as how CAD/CAM programs are structured and operate.

5.4 Final Conclusion

In closing, this is definitely a type of system that could be improved upon and adapted to more difficult geometries, but it should not solely be an individual project as the amount of work is rather large and errors are difficult to spot and resolve with just one person. This program would definitely be a help to any machine shop which is making these sort of simple turning geometries because it will enable the user to quickly reach the simulation verification point and NC production. These simulations enable the user to ensure the process is running smoothly and the Catia lathe machining workbench offers a lot of ways to easily adjust and optimize the machining program and produce NC code.

This thesis work has resulted in a much better understanding of programming, machining, and CAD/CAM programs and has still succeeded in fulfilling two out of three objectives. The process does not require too much user input and also will be easy to adopt for more complicated geometries, but of course it falls short on one objective which was the fully automated turning simulation. In conclusion the program is a success, but not a complete success: it is something that enables the engineer to save time and concentrate on optimizing the turning process while being easy and quick to use.

Figures

Figure 1. Temperature Dissipation of Metal Cutting Processes (2)

Figure 2. Chip formation factors. (2)

Figure 3. Example of V_c , f , a_p , and n . (2)

Figure 4. Visual Description of Turning Operations (2)

Figure 5. Comparison of Stage Priority Factors (3)

Figure 6. Workflow for Tool, Toolholder, and Cutting Parameters Selection.

Figure 7. Product Ready for Manufacturing Process. Note the Origin on the Billet Face.

Figure 8. Tool Path Selection

Figure 9. Operation Strategy

Figure 10. Toolholder, Insert, and Cutting Parameter Selection

Figure 11. Program Workflow

Figure 12. Part of the tool catalog

Figure 13. Excel Userform

Figure 14. The Ready Product

Figure 15. Prepared Product: Ready for Machining

Figure 16. Part Operation 1 Preparation

Figure 17. Excel Machining Report

Figure 18. Roughing Process

Figure 19. Part Operation 1 Finalized

Figure 20. Part Operation 2 Preparation

Figure 21. Tool Path Selection Using Pre-Defined Sketches

Figure 22. Part Operation 2 Finalized

Figure 23. Tool Changes and Machining Operations

Figure 24. NC Code Generation

List of references

1. Sandvik Coromant Technical Editorial Dept, p. 1994. *Modern Metal Cutting*. Sandviken: Idereklam, 1994. Print.
2. Sandvik Coromant Technical Editorial Dept. *Sandvik General Turning*. Sandvik Coromant: 2007. Sandvik Coromant English. Web. 16 Oct 2015.
3. Balci, Tolga. "Getting Started with Microsoft Excel Visual Basic for Applications (VBA)." *Bright Hub*. Ed. Michele McDonough. N.p., 28 Feb. 2010. Web. 6 Oct. 2015. <<http://www.brighthub.com/computing/windowsplatform/articles/32278.aspx>>.
4. Chi, Cheung Ching. *Semi-automated Process Planning and Cost Estimation of Turned Components Based on CATIA V5 Machining*. Thesis. Jönköping University, 2008. Web. 10 Oct. 2015.
5. Lomax, P., (1998) "VB & VBA in a Nutshell: The Language", O'Reilly Media, Inc., Boston, USA.
6. Lefebvre, Gerald. "Introduction to CATIA V5 Automation." Web log post. *CATIA V5 Design & Automation*. N.p., n.d. Web. 08 Oct. 2015.
7. Dassault Systemes. "Catia NC Programming." *Catia Resources*. Vélizy-Villacoublay: Dassault Systemes. 2003. Catia. Web. 8 Oct. 2015.

Appendices

A. VBA Script

```
*****
```

```
'*This sub marks the center point according to the user's desire.
```

```
*****
```

```
Private Sub Center_Click()
```

```
Sheet1.Activate
```

```
LastRowA = Cells(Rows.Count, "A").End(xlUp).Row
```

```
Cells>LastRowA, 9) = 1
```

```
End Sub
```

```
Private Sub Edit_Click()
```

```
*****
```

'*This sub allows the Editing field to be activated

Points.Enabled = True

End Sub

Private Sub EnterEdit_Click()

'*This sub puts out the proper output from any editing which has occurred

'*Closes the editing process upon the succesful upload of variables

Sheet1.Activate

'Declaration

Dim arrayz(15)

'Storing values in the array

arrayz(1) = z1.Value

arrayz(2) = z2.Value

arrayz(3) = z3.Value

arrayz(4) = z4.Value

arrayz(5) = z5.Value

arrayz(6) = z6.Value

arrayz(7) = z7.Value

arrayz(8) = z8.Value

arrayz(9) = z9.Value

arrayz(10) = z10.Value

arrayz(11) = z11.Value

arrayz(12) = z12.Value

arrayz(13) = z13.Value

arrayz(14) = z14.Value

arrayz(15) = z15.Value

'Declaration

Dim arrayx(15)

'Storing values in the array

arrayx(1) = x1.Value

arrayx(2) = x2.Value

arrayx(3) = x3.Value

arrayx(4) = x4.Value

arrayx(5) = x5.Value

arrayx(6) = x6.Value

arrayx(7) = x7.Value

arrayx(8) = x8.Value

arrayx(9) = x9.Value

arrayx(10) = x10.Value

arrayx(11) = x11.Value

```
arrayx(12) = x12.Value  
arrayx(13) = x13.Value  
arrayx(14) = x14.Value  
arrayx(15) = x15.Value
```

'Declaration

Dim arrayf(15)

'Storing values in the array

```
arrayf(1) = f1.Value  
arrayf(2) = f2.Value  
arrayf(3) = f3.Value  
arrayf(4) = f4.Value  
arrayf(5) = f5.Value  
arrayf(6) = f6.Value  
arrayf(7) = f7.Value  
arrayf(8) = f8.Value  
arrayf(9) = f9.Value  
arrayf(10) = f10.Value  
arrayf(11) = f11.Value  
arrayf(12) = f12.Value  
arrayf(13) = f13.Value  
arrayf(14) = f14.Value  
arrayf(15) = f15.Value
```

'Declaration

Dim arrays(15)

'Storing values in the array

```
arrays(1) = s1.Value  
arrays(2) = s2.Value  
arrays(3) = s3.Value  
arrays(4) = s4.Value  
arrays(5) = s5.Value  
arrays(6) = s6.Value  
arrays(7) = s7.Value  
arrays(8) = s8.Value  
arrays(9) = s9.Value  
arrays(10) = s10.Value  
arrays(11) = s11.Value  
arrays(12) = s12.Value  
arrays(13) = s13.Value  
arrays(14) = s14.Value  
arrays(15) = s15.Value
```

'closes the ability to edit until the edit button is once again pressed

Points.Enabled = False

End Sub

!*

!*This sub sets up the product. It first creates the geometry of the finished piece according to the points entered

!*Using geometrical methods, the sketch is completely produced.

!*Then it produces the geometry of the billet

!*Next it produces a product of the two combined

!*Finally it places an axis which will be used for machining on the product

!*

Private Sub Export_Click()

!*

!*Set up connection between Catia and Excel

!*If Catia is not on, then it will be started

!*

Dim CATIA As Object

On Error Resume Next

Set CATIA = GetObject("CATIA.Application")

If Err.Number <> 0 Then

Set CATIA = CreateObject("CATIA.Application")

CATIA.Visible = True

End If

On Error GoTo 0

!*

!*Connect to the relevant Catia libraries

!*Create the finished piece geometry

!*Import all points

!*Connect points with lines or radius as desired

!*Rotate sketch around center axis to create revolved shape

!*

Dim myDocument As Documents

Set myDocument = CATIA.Documents

Dim partDocument1 As PartDocument

Set partDocument1 = myDocument.Add("Part")

```

Dim part1 As Part
Set part1 = partDocument1.Part

Dim bodies1 As Bodies
Set bodies1 = part1.Bodies

Dim body1 As Body
Set body1 = bodies1.Item("PartBody")

Dim sketches1 As sketches
Set sketches1 = body1.sketches

Dim originelements1 As OriginElements
Set originelements1 = part1.OriginElements

Dim reference1 As Reference
Set reference1 = originelements1.PlaneZX

Dim sketch1 As Sketch
Set sketch1 = sketches1.Add(reference1)

Dim factory2D1 As Factory2D
Set factory2D1 = sketch1.OpenEdition

*****
'import all points
*****

Sheet1.Activate
LastRowA = Cells(Rows.Count, "A").End(xlUp).Row
Dim i As Integer
i = 1
Do While i < (LastRowA + 1)
Dim point2D1 As Point2D
Set point2D1 = factory2D1.CreatePoint(Cells(i, 1).Value, Cells(i, 2).Value)
i = i + 1
Loop

*****
'Connect all points with line or radius as desired
*****

Dim j As Integer
j = 2
Do While j < (LastRowA - 1)
Dim line2D1 As Line2D

```

```

Set line2D1 = factory2D1.CreateLine(Cells(j, 1).Value, Cells(j, 2).Value, Cells(j + 1, 1).Value, Cells(j + 1, 2).Value)
j = j + 1
Loop

```

```

!*****

```

```

'Close sketch along the axis

```

```

!*****

```

```

Set line2D1 = factory2D1.CreateLine(Cells(2, 1).Value, Cells(2, 2).Value, Cells(LastRowA - 1, 1).Value, Cells(LastRowA - 1, 2).Value)

```

```

sketch1.CloseEdition

```

```

!*****

```

```

'Revolve sketch to create finished part

```

```

!*****

```

```

Dim My_Shift As Shaft
Set My_Shift = part1.ShapeFactory.AddNewShift(sketch1)
Dim geometricElements1 As GeometricElements
Set geometricElements1 = sketch1.GeometricElements
Dim axis2d1 As Axis2D
Set axis2d1 = geometricElements1.Item("AbsoluteAxis")
Dim reference2 As Reference
Set reference2 = axis2d1.GetItem("HDirection")
My_Shift.RevolveAxis = reference2

```

```

Dim sketches11 As sketches
Set sketches11 = body1.sketches

```

```

Dim originelements11 As OriginElements
Set originelements11 = part1.OriginElements

```

```

Dim reference11 As Reference
Set reference11 = originelements11.PlaneZX

```

```

Dim sketch11 As Sketch
Set sketch11 = sketches11.Add(reference11)

```

```

Dim factory2D11 As Factory2D
Set factory2D11 = sketch11.OpenEdition

```

```

!*****

```

```

'Sketches Notch sketches

```

```

!*****

```

```

Dim k As Integer
k = 1

```

```

Do While k <= LastRowA
  If Cells(k, 10).Value = 1 Then
    Dim sketch12 As Sketch
    Set sketch12 = sketches1.Add(reference1)

```

```

    Dim factory2D12 As Factory2D
    Set factory2D12 = sketch12.OpenEdition

```

```

    !*****

```

'The following creates an open 'U' shape which the Catia Lathe Machining Workbench uses for notch operations

```

    !*****

```

```

    Dim line2D12 As Line2D
    Set line2D12 = factory2D12.CreateLine(Cells(k, 1).Value, Cells(k, 2).Value, Cells(k + 1, 1).Value, Cells(k + 1, 2).Value)
    Set line2D12 = factory2D12.CreateLine(Cells(k + 1, 1).Value, Cells(k + 1, 2).Value, Cells(k + 2, 1).Value, Cells(k + 2, 2).Value)
    Set line2D12 = factory2D12.CreateLine(Cells(k + 2, 1).Value, Cells(k + 2, 2).Value, Cells(k + 3, 1).Value, Cells(k + 3, 2).Value)
    k = k + 3
    sketch12.CloseEdition
  Else
  End If

```

```

  k = k + 1
Loop

```

```

!*****

```

```

!*

```

'*The following will create a sketch which can be used for both Roughing and Finishing operations

'*Catia will automatically detect collisions if you enable it

'*This allows the program to use the strengths of Catia

```

!*

```

```

!*****

```

```

!*****

```

'Find roughing and finishing sketch

```

!*****

```

```

LastRowA = Cells(Rows.Count, "A").End(xlUp).Row
Dim I As Integer
I = 2
Dim sketch13 As Sketch
Set sketch13 = sketches1.Add(reference1)

```

```

Dim factory2D13 As Factory2D

```

```

Set factory2D13 = sketch13.OpenEdition
Dim line2D13 As Line2D
Do While I < LastRowA
    *****

    'Creates a straight line over the notch feature
    *****

    If Cells(I, 10).Value = 1 Then

        Set line2D13 = factory2D13.CreateLine(Cells(I, 1).Value, Cells(I, 2).Value, Cells(I +
3, 1).Value, Cells(I + 3, 2).Value)
        I = I + 3
    Else
    End If
    *****

    'This gets the rest of the sketch from the Excel sheet allowing the sketch to be
connected
    *****

    Set line2D13 = factory2D13.CreateLine(Cells(I, 1).Value, Cells(I, 2).Value, Cells(I +
1, 1).Value, Cells(I + 1, 2).Value)
    I = I + 1
    *****

    'This calls the radius creation
    'This feature is still not always bug free
    'If the program does not execute correctly then add radius in manually
    *****

    If Cells(I + 1, 4) > 0 Then
        Set line2D13 = factory2D13.CreateLine(Cells(I, 1).Value, Cells(I, 2).Value, Cells(I +
1, 1).Value, Cells(I + 1, 2).Value)
        Dim lc1 As line2D13
        Set line2D13 = factory2D13.CreateLine(Cells(I + 1, 1).Value, Cells(I + 1, 2).Value,
Cells(I + 2, 1).Value, Cells(I + 2, 2).Value)
        Dim lc2 As line2D13

        Dim hybridBodies1 As HybridBodies
        Set hybridBodies1 = part1.HybridBodies

        Dim hb1 As HybridBody
        Set hb1 = hybridBodies1.Item("Geometrical Set.1")

        sketch13 = hb1.HybridSketches.Add(BasePlane)
        spa As SPAWorkbench
        spa = CATIA.ActiveDocument.GetWorkbench("SPAWorkbench")
        Dim c1(1), c2(1)
        Dim b1(1), b2(1)
        Dim vRef = spa.GetMeasurable (Line1).GetMinimumDistance (Line2)

```



```

Dim refPoint1 As Point2D
Dim refPoint2 As Point2D

Dim C As Constraint
Line1.StartPoint.GetCoordinates (c1)
Line1.EndPoint.GetCoordinates (c2)

Line2.StartPoint.GetCoordinates (b1)
Line2.EndPoint.GetCoordinates (b2)

If
Math.Round(spa.GetMeasurable(Line1.StartPoint).GetMinimumDistance(Line2.StartPoint) - vRef, 4) = 0 Then
    refPoint1 = Line1.StartPoint
    refPoint2 = Line2.StartPoint
Elseif
Math.Round(spa.GetMeasurable(Line1.StartPoint).GetMinimumDistance(Line2.EndPoint) - vRef, 4) = 0 Then
    refPoint1 = Line1.StartPoint
    refPoint2 = Line2.EndPoint
Elseif
Math.Round(spa.GetMeasurable(Line1.EndPoint).GetMinimumDistance(Line2.StartPoint) - vRef, 4) = 0 Then
    refPoint1 = Line1.EndPoint
    refPoint2 = Line2.StartPoint
Elseif
Math.Round(spa.GetMeasurable(Line1.EndPoint).GetMinimumDistance(Line2.EndPoint) - vRef, 4) = 0 Then
    refPoint1 = Line1.EndPoint
    refPoint2 = Line2.EndPoint
End If

c1(0) = (c1(0) + b1(0) + c2(0) + b2(0)) / 4
c1(1) = (c1(1) + b1(1) + c2(1) + b2(1)) / 4

Dim Fix1 As Constraint
Fix1 = sketch13.Constraints.AddMonoEltCst(catCstTypeReference, Line1)
Dim Fix2 As Constraint
Fix2 = sketch13.Constraints.AddMonoEltCst(catCstTypeReference, Line2)

Dim CenterPoint As Point2D
CenterPoint = factory2D13.CreatePoint(c1(0), c1(1))
CenterPoint.Construction = True

C = sketch13.Constraints.AddBiEltCst(catCstTypeDistance, CenterPoint, Line1)

```

```

C.Dimension.Value = Radius

C = sketch13.Constraints.AddBiEltCst(catCstTypeDistance, CenterPoint, Line2)
C.Dimension.Value = Radius

CenterPoint.GetCoordinates (c1)

Dim Arc As Circle2D
Arc = factory2D13.CreateCircle(c1(0), c1(1), Radius, 0, 1)
C = sketch13.Constraints.AddMonoEltCst(catCstTypeRadius, Arc)
C.Dimension.Value = Radius

C = sketch13.Constraints.AddBiEltCst(catCstTypeTangency, Arc, Line1)
C = sketch13.Constraints.AddBiEltCst(catCstTypeTangency, Arc, Line2)

Dim ct1 As Constraint
ct1 = sketch13.Constraints.AddBiEltCst(catCstTypeOn, Arc.StartPoint, Line1)
Dim ct2 As Constraint
ct2 = sketch13.Constraints.AddBiEltCst(catCstTypeOn, Arc.EndPoint, Line2)

If spa.GetMeasurable(Arc).Length > 3.14 * Radius Then
    Change = True
    sketch13 = Clear()
    sketch13.Add (ct1)
    sketch13.Add (ct2)
    sketch13 = Delete()
    ct1 = sketch13.Constraints.AddBiEltCst(catCstTypeOn, Arc.EndPoint, Line1)
    ct2 = sketch13.Constraints.AddBiEltCst(catCstTypeOn, Arc.StartPoint, Line2)
End If
sketch13 = Clear()
sketch13.Add (Fix1)
sketch13.Add (Fix2)
sketch13 = Delete()
Loop
sketch13.CloseEdition

part1.Update

*****
'save finished piece as "part1.CATPart"
'to save in a different location this must be changed manually
*****
CATIA.ActiveDocument.SaveAs      "C:\Users\Raisanen\Documents\CATIA\Raisanen
Thesis\part1"
*****

```

```

!*
!*Create the billet geometry
!*Create all points
!*Connect points with lines
!*Rotate sketch around center axis to create revolved shape
!*

```

```

*****

```

```

Dim myDocument2 As Documents
Set myDocument2 = CATIA.Documents

```

```

Dim partDocument2 As PartDocument
Set partDocument2 = myDocument.Add("Part")

```

```

Dim part2 As Part
Set part2 = partDocument2.Part

```

```

Dim bodies2 As Bodies
Set bodies2 = part2.Bodies

```

```

Dim body2 As Body
Set body2 = bodies2.Item("PartBody")

```

```

Dim sketches2 As sketches
Set sketches2 = body2.sketches

```

```

Dim originelements2 As OriginElements
Set originelements2 = part2.OriginElements

```

```

Dim reference3 As Reference
Set reference3 = originelements2.PlaneZX

```

```

Dim sketch2 As Sketch
Set sketch2 = sketches2.Add(reference3)

```

```

Dim factory2D2 As Factory2D
Set factory2D2 = sketch2.OpenEdition

```

```

*****

```

```

'Enter necessary points according to the billet geometry

```

```

*****

```

```

Dim point2D2 As Point2D
Set point2D2 = factory2D2.CreatePoint(0, 0)
Set point2D2 = factory2D2.CreatePoint(0, BilletDia / 2)
Set point2D2 = factory2D2.CreatePoint(BilletLength, BilletDia / 2)
Set point2D2 = factory2D2.CreatePoint(BilletLength, 0)

```

```
*****
```

```
'Set up all lines for the billet
```

```
*****
```

```
Dim line2D2 As Line2D
Set line2D2 = factory2D2.CreateLine(0, 0, 0, (BilletDia / 2))
Set line2D2 = factory2D2.CreateLine(0, (BilletDia / 2), BilletLength, (BilletDia / 2))
Set line2D2 = factory2D2.CreateLine(BilletLength, (BilletDia / 2), BilletLength, 0)
Set line2D2 = factory2D2.CreateLine(BilletLength, 0, 0, 0)
```

```
sketch1.CloseEdition
```

```
*****
```

```
'Create the revolved solid
```

```
*****
```

```
Dim My_Shaft2 As Shaft
Set My_Shaft2 = part2.ShapeFactory.AddNewShaft(sketch2)
Dim geometricElements2 As GeometricElements
Set geometricElements2 = sketch2.GeometricElements
Dim axis2d2 As Axis2D
Set axis2d2 = geometricElements2.Item("AbsoluteAxis")
Dim reference4 As Reference
Set reference4 = axis2d2.GetItem("HDirection")
My_Shaft2.RevolveAxis = reference4
```

```
*****
```

```
!*
```

```
!*Create the proper axis system for the first phase of machining
```

```
!*Involves defining a new axis
```

```
!*Similar process will have to occur once more for the second phase
```

```
!*
```

```
*****
```

```
Dim oPartDocument As PartDocument
Dim oPart As Part
Dim oAxis As Object 'Actual type is AxisSystem, but some properties & methods are
marked "restricted" unless we late bind the type.
```

```
'Get the part document and part objects
```

```
Set oPartDocument = CATIA.ActiveDocument
```

```
Set oPart = oPartDocument.Part
```

```
'Create the axis, configure how the vectors & origin are defined
```

```
Set oAxis = oPart.AxisSystems.Add
```

```
oAxis.Type = catAxisSystemStandard
```

```

Dim oReference As Reference
Set                                     oReference                                     =
oPart.CreateReferenceFromObject(oPart.HybridBodies.Item(1).HybridShapes.Item("Point.10"))

oAxis.OriginType = catAxisSystemOriginByPoint '0
oAxis.OriginPoint = oReference

oAxis.XAxisType = catAxisSystemAxisSameDirection '0
oAxis.YAxisType = catAxisSystemAxisSameDirection '0
oAxis.ZAxisType = catAxisSystemAxisSameDirection '0

'Set the Z-Axis by coordinates
Set                                     oReference                                     =
oPart.CreateReferenceFromObject(oPart.HybridBodies.Item(1).HybridShapes.Item("Axis.1"))

oAxis.ZAxisDirection = oReference

'Update the axis system
oPart.UpdateObject oAxis
oPart.Update

*****

'Save billet part as "part2.CATPart"
'To save in a different location this must be changed manually
*****

part2.Update
CATIA.ActiveDocument.SaveAs           "C:\Users\Raisanen\Documents\CATIA\Raisanen
Thesis\part2"

*****

!*
!*This saves the product which contains the new axis system as "product1.CATProduct"
!*The path must be manually changed to save in a different location
!*
*****

Set documents1 = CATIA.Documents
Set productDocument1 = documents1.Add("Product")
Set product1 = productDocument1.Product
Set products1 = product1.Products
Dim arrayOfVariantOfBSTR1(0)

arrayOfVariantOfBSTR1(0)              =      "C:\Users\Raisanen\Documents\CATIA\Raisanen
Thesis\part1.CATPart"
products1.AddComponentsFromFiles arrayOfVariantOfBSTR1, "All"

```

```
Dim arrayOfVariantOfBSTR2(0)
```

```
arrayOfVariantOfBSTR2(0) = "C:\Users\Raisanen\Documents\CATIA\Raisanen  
Thesis\part2.CATPart"  
products1.AddComponentsFromFiles arrayOfVariantOfBSTR2, "All"
```

```
CATIA.ActiveDocument.SaveAs "C:\Users\Raisanen\Documents\CATIA\Raisanen  
Thesis\product1"
```

```
End Sub
```

```
!*****
```

```
!*
```

```
!*This sub calculates which tools should be used for which operation
```

```
!*Outputs to a small report form in Sheet3
```

```
!*The data comes from the tool catalog
```

```
!*
```

```
!*****
```

```
Private Sub Export2_Click()
```

```
Sheet1.Activate
```

```
Sheet2.Activate
```

```
LastRowA = Cells(Rows.Count, "A").End(xlUp).Row
```

```
!*****
```

```
'calculate if round inserts will work for the roughing and if it is worth using
```

```
'volume to be removed if it is high proportionwise then circle insert will be used, high>33%
```

```
'then applies tools for finishing and notch features as appropriate
```

```
!*****
```

```
Dim m As Integer
```

```
m = 1
```

```
Dim Catalog As Worksheet
```

```
Dim Report As Worksheet
```

```
Dim sourceRange As Range
```

```
Dim destRange As Range
```

```
' Define worksheet object variables
```

```
Set wscat = Worksheets("Sheet2")
```

```
Set wsrep = Worksheets("Sheet3")
```

```
If BilletDia - Sheets("Sheet1").Range("B" & 3) > BilletDia / 3 And BilletDia -  
Sheets("Sheet1").Range("B" & 4) > BilletDia / 3 And Sheets("Sheet1").Range("A" & 3) -  
Sheets("Sheet1").Range("A" & 4) > 25 Then
```

```
    If BilletDia - Sheets("Sheet1").Range("B" & 4) > 30 Then
```

```

    With wsrep
    Set destRange = .Range(.Cells(2, 1), .Cells(2, 12))
    End With
    With wscat
    Set sourceRange = .Range(.Cells(3, 1), .Cells(3, 12))
    End With
    destRange.Value = sourceRange.Value
Else
    With wsrep
    Set destRange = .Range(.Cells(2, 1), .Cells(2, 12))
    End With
    With wscat
    Set sourceRange = .Range(.Cells(2, 1), .Cells(2, 12))
    End With
    destRange.Value = sourceRange.Value
End If
Else
    With wsrep
    Set destRange = .Range(.Cells(2, 1), .Cells(2, 12))
    End With
    With wscat
    Set sourceRange = .Range(.Cells(5, 1), .Cells(5, 12))
    End With
    destRange.Value = sourceRange.Value
End If
    With wsrep
    Set destRange = .Range(.Cells(3, 1), .Cells(3, 12))
    End With
    With wscat
    Set sourceRange = .Range(.Cells(4, 1), .Cells(4, 12))
    End With
    destRange.Value = sourceRange.Value

If BilletDia - Sheets("Sheet1").Range("B" & LastRowA - 2) > BilletDia / 3 And BilletDia -
Sheets("Sheet1").Range("B" & LastRowA - 3) > BilletDia / 3 And
Sheets("Sheet1").Range("A" & LastRowA - 2) - Sheets("Sheet1").Range("A" & LastRowA
- 3) > 25 Then
    If BilletDia - Sheets("Sheet1").Range("A" & LastRowA - 2) > 30 Then
        With wsrep
        Set destRange = .Range(.Cells(5, 1), .Cells(5, 12))
        End With
        With wscat
        Set sourceRange = .Range(.Cells(3, 1), .Cells(3, 12))
        End With
        destRange.Value = sourceRange.Value
    Else

```

```

        With wsrep
        Set destRange = .Range(.Cells(5, 1), .Cells(5, 12))
        End With
        With wscat
        Set sourceRange = .Range(.Cells(2, 1), .Cells(2, 12))
        End With
        destRange.Value = sourceRange.Value
    End If
    Else
    With wsrep
    Set destRange = .Range(.Cells(5, 1), .Cells(5, 12))
    End With
    With wscat
    Set sourceRange = .Range(.Cells(5, 1), .Cells(5, 12))
    End With
    destRange.Value = sourceRange.Value
End If
    With wsrep
    Set destRange = .Range(.Cells(6, 1), .Cells(6, 12))
    End With
    With wscat
    Set sourceRange = .Range(.Cells(4, 1), .Cells(4, 12))
    End With
    destRange.Value = sourceRange.Value
    With wsrep
    Set destRange = .Range(.Cells(7, 1), .Cells(7, 12))
    End With
    With wscat
    Set sourceRange = .Range(.Cells(6, 1), .Cells(6, 12))
    End With
    destRange.Value = sourceRange.Value

End Sub

*****
!*
!*This sub sends the user entered data to the Excel sheet as coordinates
!*It automatically solves for radius, chamfer, and notch features
!*Solves for correct point placement
!*
*****

Private Sub SendData_Click()
Dim emptyRow As Long

*****

```


'*Enters in the first few values which are only dependent on finished length and billet length

```
Sheet1.Activate
Cells(1, 1).Value = 0
z1 = 0
Cells(1, 2).Value = 0
x1 = 0
Cells(1, 3).Value = 1
s3 = "Origin"
i = (BilletLength - FinishedLength) / 2
Cells(2, 1).Value = i
z2 = i
Cells(2, 2).Value = 0
x2 = 0
Dim arrayz(15)
Dim arrayx(15)
Dim arrayf(15)
Dim arrays(15)
emptyRow = WorksheetFunction.CountA(Range("A:A")) + 1
LastRowA = Cells(Rows.Count, "A").End(xlUp).Row
Dim z As Integer
Const radians As Double = 3.14159265358979 / 180
```

'Automatically detects if the piece has finished already

'*Check 1

If Cells(LastRowA, 2).Value = 0 And Cells(LastRowA, 1).Value > 0 And Cells(LastRowA - 1, 1).Value > 0 Then

Instructions.Text = "Geometry is now closed, Press Export to produce simulation and Excel Report"

Cells(emptyRow, 1).Value = FinishedLength

arrayz(emptyRow) = FinishedLength

Cells(emptyRow, 2).Value = 0

arrayx(emptyRow) = 0

Cells(emptyRow, 1).Value = BilletLength

arrayz(emptyRow) = BilletLength

Cells(emptyRow, 2).Value = 0

arrayx(emptyRow) = 0

Cells(emptyRow, 3).Value = 1

arrayf(emptyRow) = "Origin"

Else

End If

'Solves for the line feature depending on the length and the angle entered
'Second check if the piece is finished
'Angle is taken as 0 degrees coming from the clamping system in the Z direction
!*****

If UserEntry = "Line" Then

!*****

'Checks if there is a chamfer feature. If there is then the distance
'is counted from the line from where the chamfer begins.
'This first if statement is for a positive angled chamfer
!*****

If Cells(LastRowA, 5).Value > 0 Then
Cells(emptyRow, 1).Value = ((Cells(LastRowA, 1).Value - Cells(LastRowA, 5).Value)
+ Cos(Angles.Value * radians) * Dimensions.Value)
Cells(emptyRow, 2).Value = (Cells(LastRowA, 2).Value + Sin(Angles.Value * radians)
* Dimensions.Value)
arrayz(emptyRow) = Cells(emptyRow, 1).Value
arrayx(emptyRow) = Cells(emptyRow, 2).Value
Else
End If

!*****

'Checks if there is a chamfer feature. If there is then the distance
'is counted from the line from where the chamfer begins.
'This second if statement is for a negative angled chamfer
!*****

If Cells(LastRowA, 5).Value < 0 Then
Cells(emptyRow, 1).Value = (Cells(LastRowA, 1).Value + Cos(Angles.Value * radians)
* Dimensions.Value)
Cells(emptyRow, 2).Value = ((Cells(LastRowA, 2).Value + Cells(LastRowA, 5)) +
Sin(Angles.Value * radians) * Dimensions.Value)
arrayz(emptyRow) = Cells(emptyRow, 1).Value
arrayx(emptyRow) = Cells(emptyRow, 2).Value
Else
End If

!*****

'Normal line entry

!*****

If Cells(LastRowA, 5).Value = 0 Then
Cells(emptyRow, 1).Value = (Cells(LastRowA, 1).Value + Cos(Angles.Value * radians)
* Dimensions.Value)
Cells(emptyRow, 2).Value = (Cells(LastRowA, 2).Value + Sin(Angles.Value * radians)
* Dimensions.Value)
arrayz(emptyRow) = Cells(emptyRow, 1).Value
arrayx(emptyRow) = Cells(emptyRow, 2).Value
Else
End If
UserEntry = ""

```

If Cells(LastRowA, 2).Value = 0 And Cells(LastRowA - 1, 2).Value > 0 Then
    Instructions.Text = "Geometry is now closed, Press Export to produce simulation and
Excel Report"

```

```

    Cells(emptyRow, 1).Value = FinishedLength
    Cells(emptyRow, 2).Value = 0
    Cells(emptyRow, 3).Value = 1
    arrayz(emptyRow) = Cells(emptyRow, 1).Value
    arrayx(emptyRow) = Cells(emptyRow, 2).Value
    arrayf(emptyRow) = "Origin"
Else
End If

```

```

Else
End If

```

```

!*****

```

'Marks that for this point a radius must be produced within Catia when doing the sketches

```

!*****

```

```

If UserEntry = "Radius" Then
    Cells(LastRowA, 4).Value = Dimensions
    UserEntry = ""
    arrayf(LastRowA) = "Radius"
    arrays(LastRowA) = Cells(LastRowA, 4).Value
Else
End If

```

```

!*****

```

'Solves for the Chamfer

'Modifies the last row to reflect the new end point

'Adds the second point according to the length

```

!*****

```

```

If UserEntry = "Chamfer" Then

```

```

    Dim m As Double

```

```

    Dim n As Double

```

```

    If Angles > 0 Then

```

```

        !*****

```

!*Positive geometry

```

        !*****

```

```

        m = Cells(LastRowA, 1).Value

```

```

        n = Cells(LastRowA, 2).Value

```

```

        Cells(LastRowA, 2).Value = n - (Cos(Angles) * Dimensions)

```

```

        arrayx(LastRowA) = Cells(LastRowA, 2).Value

```

```

        Cells(emptyRow, 1).Value = m + (Sin(Angles) * Dimensions)

```

```

        arrayz(emptyRow) = Cells(emptyRow, 1).Value

```

```

        Cells(emptyRow, 2).Value = n

```

```

        arrayx(emptyRow) = Cells(emptyRow, 2).Value

```

```

Cells(emptyRow, 5).Value = (Sin(Angles) * Dimensions)
arrays(emptyRow) = Cells(emptyRow, 5).Value
arrayf(emptyRow) = "Chamfer"
Else
!*****
!*Negative geometry
!*****

m = Cells(LastRowA, 1).Value
n = Cells(LastRowA, 2).Value
Cells(LastRowA, 1).Value = m - (Sin(Angles) * Dimensions)
arrayz(LastRowA) = Cells(LastRowA, 1).Value
Cells(emptyRow, 1).Value = m
arrayz(emptyRow) = Cells(emptyRow, 1).Value
Cells(emptyRow, 2).Value = n - (Cos(Angles) * Dimensions)
arrayx(emptyRow) = Cells(emptyRow, 2).Value
Cells(emptyRow, 5).Value = (Cos(Angles) * Dimensions)
arrays(emptyRow) = Cells(emptyRow, 5).Value
arrayf(emptyRow) = "Chamfer"
UserEntry = ""
End If
End If

!*****
'Solves for the notch feature based on length and depth
!*****

If UserEntry = "Notch" Then
Cells(LastRowA, 10).Value = 1
arrayf(LastRowA) = "Notch"
Cells(emptyRow, 1).Value = Cells(LastRowA, 1).Value
arrayz(emptyRow) = Cells(emptyRow, 1).Value
Cells(emptyRow, 2).Value = Cells(LastRowA, 2).Value - Angles
arrayx(emptyRow) = Cells(emptyRow, 2).Value
Cells(emptyRow, 10).Value = 1
arrayf(emptyRow) = "Notch"
Cells(emptyRow + 1, 1).Value = Cells(emptyRow, 1).Value + Dimensions
arrayz(emptyRow + 1) = Cells(emptyRow + 1, 1).Value
Cells(emptyRow + 1, 2).Value = Cells(emptyRow, 2).Value
arrayx(emptyRow + 1) = Cells(emptyRow + 1, 2).Value
Cells(emptyRow + 1, 10).Value = 1
arrayf(emptyRow + 1) = "Notch"
Cells(emptyRow + 2, 1).Value = Cells(emptyRow + 1, 1).Value
arrayz(emptyRow + 2) = Cells(emptyRow + 2, 1).Value
Cells(emptyRow + 2, 2).Value = Cells(emptyRow + 1, 2).Value + Angles
arrayx(emptyRow + 2) = Cells(emptyRow + 2, 2).Value
Cells(emptyRow + 2, 10).Value = 1
arrayf(emptyRow + 2) = "Notch"

```

```

    UserEntry = ""

Else
End If

'Storing values in the array
For z = 1 To LastRowA
    arrayz(z) = Cells(z, 1).Value
Next
For z = LastRowA + 1 To 15
    arrayz(z) = 0
Next

'Storing values in the array
For z = 1 To LastRowA
    arrayx(z) = Cells(z, 2).Value
Next
For z = LastRowA + 1 To 15
    arrayx(z) = 0
Next

'definition of all textboxes in the edit section
z1 = arrayz(1)
z2 = arrayz(2)
z3 = arrayz(3)
z4 = arrayz(4)
z5 = arrayz(5)
z6 = arrayz(6)
z7 = arrayz(7)
z8 = arrayz(8)
z9 = arrayz(9)
z10 = arrayz(10)
z11 = arrayz(11)
z12 = arrayz(12)
z13 = arrayz(13)
z14 = arrayz(14)
z15 = arrayz(15)

x1 = arrayx(1)
x2 = arrayx(2)
x3 = arrayx(3)
x4 = arrayx(4)
x5 = arrayx(5)
x6 = arrayx(6)
x7 = arrayx(7)
x8 = arrayx(8)

```

```
x9 = arrayx(9)
x10 = arrayx(10)
x11 = arrayx(11)
x12 = arrayx(12)
x13 = arrayx(13)
x14 = arrayx(14)
x15 = arrayx(15)
```

```
f1 = arrayf(1)
f2 = arrayf(2)
f3 = arrayf(3)
f4 = arrayf(4)
f5 = arrayf(5)
f6 = arrayf(6)
f7 = arrayf(7)
f8 = arrayf(8)
f9 = arrayf(9)
f10 = arrayf(10)
f11 = arrayf(11)
f12 = arrayf(12)
f13 = arrayf(13)
f14 = arrayf(14)
f15 = arrayf(15)
```

```
s1 = arrays(1)
s2 = arrays(2)
s3 = arrays(3)
s4 = arrays(4)
s5 = arrays(5)
s6 = arrays(6)
s7 = arrays(7)
s8 = arrays(8)
s9 = arrays(9)
s10 = arrays(10)
s11 = arrays(11)
s12 = arrays(12)
s13 = arrays(13)
s14 = arrays(14)
s15 = arrays(15)
```

```
End Sub
```

```
Private Sub UserEntry_Change()
```

```
*****
```

```
'Depending on the operation chosen will give different instructions to the user
```

```
*****
```

```
If UserEntry = "Line" Then
```

```
    Instructions.Text = "Enter length in mm and angle in degrees in the boxes below. Hit  
SEND when done."
```

```
Else
```

```
End If
```

```
If UserEntry = "Radius" Then
```

```
    Instructions.Text = "Enter the radius value in mm in the appropriate box below. Hit  
SEND when done."
```

```
Else
```

```
End If
```

```
If UserEntry = "Chamfer" Then
```

```
    Instructions.Text = "Enter length of the chamfered surface in mm and the angle in  
degrees in the boxes below. Hit SEND when done."
```

```
Else
```

```
End If
```

```
If UserEntry = "Notch" Then
```

```
    Instructions.Text = "Enter length in mm in the first box and the depth in mm in the  
second box. Hit SEND when done."
```

```
Else
```

```
End If
```

```
End Sub
```

```
*****
```

```
!*
```

```
'Initializes the UserForm for use
```

```
!*
```

```
*****
```

```
Private Sub UserForm_Initialize()
```

```
'Empty MachineType
```

```
MachineType.Clear
```

```
'Fill MachineType
```

```
With MachineType
```

```
    .AddItem "Catia predetermined(max n=10000rpm)"
```

```
    .AddItem "Only Catia predetermined currently available"
```

```
End With
```

```
'Empty MaterialType
MaterialType.Clear
```

```
'Fill MaterialType
With MaterialType
    .AddItem "Aluminum alloy"
    .AddItem "Only Aluminum alloy with current catalog"
End With
```

```
'Empty BilletDia
BilletDia.Value = ""
```

```
'Empty BilletLength
BilletLength.Value = ""
```

```
'Fill FinishQuality
With FinishQuality
    .AddItem "ISO 2768-mK"
    .AddItem "Only ISO 2768-mK with current catalog"
End With
```

```
'Empty Finished Length
FinishedLength.Value = ""
```

```
'Fill UserEntry
With UserEntry
    .AddItem "Line"
    .AddItem "Radius"
    .AddItem "Chamfer"
    .AddItem "Notch"
    .AddItem ""
End With
```

```
'Empty Dimensions
Dimensions.Clear
```

```
End Sub
```

```
!*****
```

```
!*
```

```
'Allows the user to edit the variables without having to exit the macro
```

```
'There is one sub for each textbox with the 'Z' variable
```

```
'Only the first will be explained
```

```
!*
```

```
!*****
```

```
Private Sub z3_AfterUpdate()
```


Dim val As Long

'array as an efficient way to store the data already entered

Dim arrayz(14) As Variant

arrayz(0) = z1.Value

arrayz(1) = z2.Value

arrayz(2) = z3.Value

arrayz(3) = z4.Value

arrayz(4) = z5.Value

arrayz(5) = z6.Value

arrayz(6) = z7.Value

arrayz(7) = z8.Value

arrayz(8) = z9.Value

arrayz(9) = z10.Value

arrayz(10) = z11.Value

arrayz(11) = z12.Value

arrayz(12) = z13.Value

arrayz(13) = z14.Value

arrayz(14) = z15.Value

Sheet1.Activate

'Each if loop edits the value by the change entered in the textbox

'if the previous value was zero then the if loop will not execute

val = arrayz(2) - Cells(3, 1).Value

 If arrayz(2) <> 0 Then

 z3 = arrayz(2) + val

 arrayz(2) = z3

 End If

 If arrayz(3) <> 0 Then

 z4 = arrayz(3) + val

 arrayz(3) = z4

 End If

 If arrayz(4) <> 0 Then

 z5 = arrayz(4) + val

 arrayz(4) = z5

 End If

 If arrayz(5) <> 0 Then

 z6 = arrayz(5) + val

 arrayz(5) = z6

 End If

 If arrayz(6) <> 0 Then

 z7 = arrayz(6) + val

 arrayz(6) = z7

 End If

 If arrayz(7) <> 0 Then

 z8 = arrayz(7) + val

 arrayz(7) = z8

```

End If
If arrayz(8) <> 0 Then
z9 = arrayz(8) + val
arrayz(8) = z9
End If
If arrayz(9) <> 0 Then
z10 = arrayz(9) + val
arrayz(9) = z10
End If
If arrayz(10) <> 0 Then
z11 = arrayz(10) + val
arrayz(10) = z11
End If
If arrayz(11) <> 0 Then
z12 = arrayz(11) + val
arrayz(11) = z12
End If
If arrayz(12) <> 0 Then
z13 = arrayz(12) + val
arrayz(12) = z13
End If
If arrayz(13) <> 0 Then
z14 = arrayz(13) + val
arrayz(13) = z14
End If
If arrayz(14) <> 0 Then
z15 = arrayz(14) + val
arrayz(14) = z15
End If

```

'loads the array values to the excel sheet

```

Dim Destination As Range
Set Destination = Range("A1")
Set Destination = Destination.Resize(UBound(arrayz), 1)
Destination.Value = Application.Transpose(arrayz)

```

End Sub

```

Private Sub z4_AfterUpdate()
Dim val As Double
Dim arrayz(14) As Variant

```

```

arrayz(0) = z1.Value
arrayz(1) = z2.Value
arrayz(2) = z3.Value

```

```

arrayz(3) = z4.Value
arrayz(4) = z5.Value
arrayz(5) = z6.Value
arrayz(6) = z7.Value
arrayz(7) = z8.Value
arrayz(8) = z9.Value
arrayz(9) = z10.Value
arrayz(10) = z11.Value
arrayz(11) = z12.Value
arrayz(12) = z13.Value
arrayz(13) = z14.Value
arrayz(14) = z15.Value
Sheet1.Activate
val = arrayz(3) - Cells(4, 1).Value
If arrayz(3) <> 0 Then
    z4 = arrayz(3) + val
    arrayz(3) = z4
End If
If arrayz(4) <> 0 Then
    z5 = arrayz(4) + val
    arrayz(4) = z5
End If
If arrayz(5) <> 0 Then
    z6 = arrayz(5) + val
    arrayz(5) = z6
End If
If arrayz(6) <> 0 Then
    z7 = arrayz(6) + val
    arrayz(6) = z7
End If
If arrayz(7) <> 0 Then
    z8 = arrayz(7) + val
    arrayz(7) = z8
End If
If arrayz(8) <> 0 Then
    z9 = arrayz(8) + val
    arrayz(8) = z9
End If
If arrayz(9) <> 0 Then
    z10 = arrayz(9) + val
    arrayz(9) = z10
End If
If arrayz(10) <> 0 Then
    z11 = arrayz(10) + val
    arrayz(10) = z11
End If

```

```

If arrayz(11) <> 0 Then
z12 = arrayz(11) + val
arrayz(11) = z12
End If
If arrayz(12) <> 0 Then
z13 = arrayz(12) + val
arrayz(12) = z13
End If
If arrayz(13) <> 0 Then
z14 = arrayz(13) + val
arrayz(13) = z14
End If
If arrayz(14) <> 0 Then
z15 = arrayz(14) + val
arrayz(14) = z15
End If

```

```

Dim Destination As Range
Set Destination = Range("A1")
Set Destination = Destination.Resize(UBound(arrayz), 1)
Destination.Value = Application.Transpose(arrayz)

```

```
End Sub
```

```

Private Sub z5_AfterUpdate()
Dim val As Long
Dim arrayz(14) As Variant

```

```

arrayz(0) = z1.Value
arrayz(1) = z2.Value
arrayz(2) = z3.Value
arrayz(3) = z4.Value
arrayz(4) = z5.Value
arrayz(5) = z6.Value
arrayz(6) = z7.Value
arrayz(7) = z8.Value
arrayz(8) = z9.Value
arrayz(9) = z10.Value
arrayz(10) = z11.Value
arrayz(11) = z12.Value
arrayz(12) = z13.Value
arrayz(13) = z14.Value
arrayz(14) = z15.Value
Sheet1.Activate

```

```

val = arrayz(4) - Cells(5, 1).Value
If arrayz(4) <> 0 Then
    z5 = arrayz(4) + val
    arrayz(4) = z5
End If
If arrayz(5) <> 0 Then
    z6 = arrayz(5) + val
    arrayz(5) = z6
End If
If arrayz(6) <> 0 Then
    z7 = arrayz(6) + val
    arrayz(6) = z7
End If
If arrayz(7) <> 0 Then
    z8 = arrayz(7) + val
    arrayz(7) = z8
End If
If arrayz(8) <> 0 Then
    z9 = arrayz(8) + val
    arrayz(8) = z9
End If
If arrayz(9) <> 0 Then
    z10 = arrayz(9) + val
    arrayz(9) = z10
End If
If arrayz(10) <> 0 Then
    z11 = arrayz(10) + val
    arrayz(10) = z11
End If
If arrayz(11) <> 0 Then
    z12 = arrayz(11) + val
    arrayz(11) = z12
End If
If arrayz(12) <> 0 Then
    z13 = arrayz(12) + val
    arrayz(12) = z13
End If
If arrayz(13) <> 0 Then
    z14 = arrayz(13) + val
    arrayz(13) = z14
End If
If arrayz(14) <> 0 Then
    z15 = arrayz(14) + val
    arrayz(14) = z15
End If

```

```
Dim Destination As Range
Set Destination = Range("A1")
Set Destination = Destination.Resize(UBound(arrayz), 1)
Destination.Value = Application.Transpose(arrayz)
```

```
End Sub
```

```
Private Sub z6_AfterUpdate()
Dim val As Long
Dim arrayz(14) As Variant
```

```
arrayz(0) = z1.Value
arrayz(1) = z2.Value
arrayz(2) = z3.Value
arrayz(3) = z4.Value
arrayz(4) = z5.Value
arrayz(5) = z6.Value
arrayz(6) = z7.Value
arrayz(7) = z8.Value
arrayz(8) = z9.Value
arrayz(9) = z10.Value
arrayz(10) = z11.Value
arrayz(11) = z12.Value
arrayz(12) = z13.Value
arrayz(13) = z14.Value
arrayz(14) = z15.Value
Sheet1.Activate
val = arrayz(5) - Cells(6, 1).Value
    If arrayz(5) <> 0 Then
        z6 = arrayz(5) + val
        arrayz(5) = z6
    End If
    If arrayz(6) <> 0 Then
        z7 = arrayz(6) + val
        arrayz(6) = z7
    End If
    If arrayz(7) <> 0 Then
        z8 = arrayz(7) + val
        arrayz(7) = z8
    End If
    If arrayz(8) <> 0 Then
        z9 = arrayz(8) + val
        arrayz(8) = z9
    End If
```

```

If arrayz(9) <> 0 Then
z10 = arrayz(9) + val
arrayz(9) = z10
End If
If arrayz(10) <> 0 Then
z11 = arrayz(10) + val
arrayz(10) = z11
End If
If arrayz(11) <> 0 Then
z12 = arrayz(11) + val
arrayz(11) = z12
End If
If arrayz(12) <> 0 Then
z13 = arrayz(12) + val
arrayz(12) = z13
End If
If arrayz(13) <> 0 Then
z14 = arrayz(13) + val
arrayz(13) = z14
End If
If arrayz(14) <> 0 Then
z15 = arrayz(14) + val
arrayz(14) = z15
End If

```

```

Dim Destination As Range
Set Destination = Range("A1")
Set Destination = Destination.Resize(UBound(arrayz), 1)
Destination.Value = Application.Transpose(arrayz)
Next
End Sub

```

```

Private Sub z7_AfterUpdate()
Dim val As Long
Dim arrayz(14) As Variant

```

```

arrayz(0) = z1.Value
arrayz(1) = z2.Value
arrayz(2) = z3.Value
arrayz(3) = z4.Value
arrayz(4) = z5.Value
arrayz(5) = z6.Value
arrayz(6) = z7.Value
arrayz(7) = z8.Value

```

```

arrayz(8) = z9.Value
arrayz(9) = z10.Value
arrayz(10) = z11.Value
arrayz(11) = z12.Value
arrayz(12) = z13.Value
arrayz(13) = z14.Value
arrayz(14) = z15.Value
Sheet1.Activate
val = arrayz(6) - Cells(7, 1).Value
    If arrayz(6) <> 0 Then
        z7 = arrayz(6) + val
        arrayz(6) = z7
    End If
    If arrayz(7) <> 0 Then
        z8 = arrayz(7) + val
        arrayz(7) = z8
    End If
    If arrayz(8) <> 0 Then
        z9 = arrayz(8) + val
        arrayz(8) = z9
    End If
    If arrayz(9) <> 0 Then
        z10 = arrayz(9) + val
        arrayz(9) = z10
    End If
    If arrayz(10) <> 0 Then
        z11 = arrayz(10) + val
        arrayz(10) = z11
    End If
    If arrayz(11) <> 0 Then
        z12 = arrayz(11) + val
        arrayz(11) = z12
    End If
    If arrayz(12) <> 0 Then
        z13 = arrayz(12) + val
        arrayz(12) = z13
    End If
    If arrayz(13) <> 0 Then
        z14 = arrayz(13) + val
        arrayz(13) = z14
    End If
    If arrayz(14) <> 0 Then
        z15 = arrayz(14) + val
        arrayz(14) = z15
    End If

```



```

Dim Destination As Range
Set Destination = Range("A1")
Set Destination = Destination.Resize(UBound(arrayz), 1)
Destination.Value = Application.Transpose(arrayz)
End Sub
Private Sub z8_AfterUpdate()
Dim val As Long
Dim arrayz(14) As Variant
arrayz(0) = z1.Value
arrayz(1) = z2.Value
arrayz(2) = z3.Value
arrayz(3) = z4.Value
arrayz(4) = z5.Value
arrayz(5) = z6.Value
arrayz(6) = z7.Value
arrayz(7) = z8.Value
arrayz(8) = z9.Value
arrayz(9) = z10.Value
arrayz(10) = z11.Value
arrayz(11) = z12.Value
arrayz(12) = z13.Value
arrayz(13) = z14.Value
arrayz(14) = z15.Value
Sheet1.Activate
val = arrayz(7) - Cells(8, 1).Value
    If arrayz(7) <> 0 Then
        z8 = arrayz(7) + val
        arrayz(7) = z8
    End If
    If arrayz(8) <> 0 Then
        z9 = arrayz(8) + val
        arrayz(8) = z9
    End If
    If arrayz(9) <> 0 Then
        z10 = arrayz(9) + val
        arrayz(9) = z10
    End If
    If arrayz(10) <> 0 Then
        z11 = arrayz(10) + val
        arrayz(10) = z11
    End If
    If arrayz(11) <> 0 Then
        z12 = arrayz(11) + val
        arrayz(11) = z12
    End If

```

```

If arrayz(12) <> 0 Then
z13 = arrayz(12) + val
arrayz(12) = z13
End If
If arrayz(13) <> 0 Then
z14 = arrayz(13) + val
arrayz(13) = z14
End If
If arrayz(14) <> 0 Then
z15 = arrayz(14) + val
arrayz(14) = z15
End If

```

```

Dim Destination As Range
Set Destination = Range("A1")
Set Destination = Destination.Resize(UBound(arrayz), 1)
Destination.Value = Application.Transpose(arrayz)
End Sub

```

```

Private Sub z9_AfterUpdate()
Dim val As Long
Dim arrayz(14) As Variant
arrayz(0) = z1.Value
arrayz(1) = z2.Value
arrayz(2) = z3.Value
arrayz(3) = z4.Value
arrayz(4) = z5.Value
arrayz(5) = z6.Value
arrayz(6) = z7.Value
arrayz(7) = z8.Value
arrayz(8) = z9.Value
arrayz(9) = z10.Value
arrayz(10) = z11.Value
arrayz(11) = z12.Value
arrayz(12) = z13.Value
arrayz(13) = z14.Value
arrayz(14) = z15.Value
Sheet1.Activate
val = arrayz(8) - Cells(9, 1).Value
If arrayz(8) <> 0 Then
z9 = arrayz(8) + val
arrayz(8) = z9
End If
If arrayz(9) <> 0 Then

```

```

z10 = arrayz(9) + val
arrayz(9) = z10
End If
If arrayz(10) <> 0 Then
z11 = arrayz(10) + val
arrayz(10) = z11
End If
If arrayz(11) <> 0 Then
z12 = arrayz(11) + val
arrayz(11) = z12
End If
If arrayz(12) <> 0 Then
z13 = arrayz(12) + val
arrayz(12) = z13
End If
If arrayz(13) <> 0 Then
z14 = arrayz(13) + val
arrayz(13) = z14
End If
If arrayz(14) <> 0 Then
z15 = arrayz(14) + val
arrayz(14) = z15
End If

```

```

Dim Destination As Range
Set Destination = Range("A1")
Set Destination = Destination.Resize(UBound(arrayz), 1)
Destination.Value = Application.Transpose(arrayz)
End Sub

```

```

Private Sub z10_AfterUpdate()
Dim val As Long
Dim arrayz(14) As Variant
arrayz(0) = z1.Value
arrayz(1) = z2.Value
arrayz(2) = z3.Value
arrayz(3) = z4.Value
arrayz(4) = z5.Value
arrayz(5) = z6.Value
arrayz(6) = z7.Value
arrayz(7) = z8.Value
arrayz(8) = z9.Value
arrayz(9) = z10.Value
arrayz(10) = z11.Value
arrayz(11) = z12.Value
arrayz(12) = z13.Value

```

```

arrayz(13) = z14.Value
arrayz(14) = z15.Value
Sheet1.Activate
val = arrayz(9) - Cells(10, 1).Value
    If arrayz(9) <> 0 Then
        z10 = arrayz(9) + val
        arrayz(9) = z10
    End If
    If arrayz(10) <> 0 Then
        z11 = arrayz(10) + val
        arrayz(10) = z11
    End If
    If arrayz(11) <> 0 Then
        z12 = arrayz(11) + val
        arrayz(11) = z12
    End If
    If arrayz(12) <> 0 Then
        z13 = arrayz(12) + val
        arrayz(12) = z13
    End If
    If arrayz(13) <> 0 Then
        z14 = arrayz(13) + val
        arrayz(13) = z14
    End If
    If arrayz(14) <> 0 Then
        z15 = arrayz(14) + val
        arrayz(14) = z15
    End If

Dim Destination As Range
Set Destination = Range("A1")
Set Destination = Destination.Resize(UBound(arrayz), 1)
Destination.Value = Application.Transpose(arrayz)

End Sub

Private Sub z11_AfterUpdate()
Dim val As Long
Dim arrayz(14) As Variant
arrayz(0) = z1.Value
arrayz(1) = z2.Value
arrayz(2) = z3.Value
arrayz(3) = z4.Value
arrayz(4) = z5.Value
arrayz(5) = z6.Value

```

```

arrayz(6) = z7.Value
arrayz(7) = z8.Value
arrayz(8) = z9.Value
arrayz(9) = z10.Value
arrayz(10) = z11.Value
arrayz(11) = z12.Value
arrayz(12) = z13.Value
arrayz(13) = z14.Value
arrayz(14) = z15.Value
Sheet1.Activate
val = arrayz(10) - Cells(11, 1).Value
    If arrayz(10) <> 0 Then
        z11 = arrayz(10) + val
        arrayz(10) = z11
    End If
    If arrayz(11) <> 0 Then
        z12 = arrayz(11) + val
        arrayz(11) = z12
    End If
    If arrayz(12) <> 0 Then
        z13 = arrayz(12) + val
        arrayz(12) = z13
    End If
    If arrayz(13) <> 0 Then
        z14 = arrayz(13) + val
        arrayz(13) = z14
    End If
    If arrayz(14) <> 0 Then
        z15 = arrayz(14) + val
        arrayz(14) = z15
    End If

Dim Destination As Range
Set Destination = Range("A1")
Set Destination = Destination.Resize(UBound(arrayz), 1)
Destination.Value = Application.Transpose(arrayz)

End Sub
Private Sub z12_AfterUpdate()
Dim val As Long
Dim arrayz(14) As Variant
arrayz(0) = z1.Value
arrayz(1) = z2.Value
arrayz(2) = z3.Value
arrayz(3) = z4.Value
arrayz(4) = z5.Value

```

```

arrayz(5) = z6.Value
arrayz(6) = z7.Value
arrayz(7) = z8.Value
arrayz(8) = z9.Value
arrayz(9) = z10.Value
arrayz(10) = z11.Value
arrayz(11) = z12.Value
arrayz(12) = z13.Value
arrayz(13) = z14.Value
arrayz(14) = z15.Value
Sheet1.Activate
val = arrayz(11) - Cells(12, 1).Value
    If arrayz(11) <> 0 Then
        z12 = arrayz(11) + val
        arrayz(11) = z12
    End If
    If arrayz(12) <> 0 Then
        z13 = arrayz(12) + val
        arrayz(12) = z13
    End If
    If arrayz(13) <> 0 Then
        z14 = arrayz(13) + val
        arrayz(13) = z14
    End If
    If arrayz(14) <> 0 Then
        z15 = arrayz(14) + val
        arrayz(14) = z15
    End If

Dim Destination As Range
Set Destination = Range("A1")
Set Destination = Destination.Resize(UBound(arrayz), 1)
Destination.Value = Application.Transpose(arrayz)

End Sub
Private Sub z13_AfterUpdate()
Dim val As Long
Dim arrayz(14) As Variant
arrayz(0) = z1.Value
arrayz(1) = z2.Value
arrayz(2) = z3.Value
arrayz(3) = z4.Value
arrayz(4) = z5.Value
arrayz(5) = z6.Value
arrayz(6) = z7.Value

```

```

arrayz(7) = z8.Value
arrayz(8) = z9.Value
arrayz(9) = z10.Value
arrayz(10) = z11.Value
arrayz(11) = z12.Value
arrayz(12) = z13.Value
arrayz(13) = z14.Value
arrayz(14) = z15.Value
Sheet1.Activate
val = arrayz(12) - Cells(13, 1).Value
    If arrayz(12) <> 0 Then
        z13 = arrayz(12) + val
        arrayz(12) = z13
    End If
    If arrayz(13) <> 0 Then
        z14 = arrayz(13) + val
        arrayz(13) = z14
    End If
    If arrayz(14) <> 0 Then
        z15 = arrayz(14) + val
        arrayz(14) = z15
    End If

Dim Destination As Range
Set Destination = Range("A1")
Set Destination = Destination.Resize(UBound(arrayz), 1)
Destination.Value = Application.Transpose(arrayz)

End Sub

Private Sub z14_AfterUpdate()
Dim val As Long
Dim arrayz(14) As Variant
arrayz(0) = z1.Value
arrayz(1) = z2.Value
arrayz(2) = z3.Value
arrayz(3) = z4.Value
arrayz(4) = z5.Value
arrayz(5) = z6.Value
arrayz(6) = z7.Value
arrayz(7) = z8.Value
arrayz(8) = z9.Value
arrayz(9) = z10.Value
arrayz(10) = z11.Value
arrayz(11) = z12.Value

```

```

arrayz(12) = z13.Value
arrayz(13) = z14.Value
arrayz(14) = z15.Value
Sheet1.Activate
val = arrayz(13) - Cells(14, 1).Value
    If arrayz(13) <> 0 Then
        z14 = arrayz(13) + val
        arrayz(13) = z14
    End If
    If arrayz(14) <> 0 Then
        z15 = arrayz(14) + val
        arrayz(14) = z15
    End If

```

```

Dim Destination As Range
Set Destination = Range("A1")
Set Destination = Destination.Resize(UBound(arrayz), 1)
Destination.Value = Application.Transpose(arrayz)

```

End Sub

```

Private Sub z15_AfterUpdate()
Dim arrayz(15)

```

```

    arrayz(15) = z15.Value

```

```

    For z = 15 To LastRowA
        Cells(z, 1).Value = arrayz(z)
    
```

End Sub

```

*****

```

```

!*

```

```

'Allows the user to edit the variables without having to exit the macro
'There is one sub for each textbox with the 'X' variable
'Only the first will be explained

```

```

!*

```

```

*****

```

```

Private Sub x3_AfterUpdate()
Dim val As Long

```

```

'array as efficient way to store all previously entered values
Dim arrayx(14) As Variant
arrayx(0) = x1.Value
arrayx(1) = x2.Value

```



```
arrayx(2) = x3.Value
arrayx(3) = x4.Value
arrayx(4) = x5.Value
arrayx(5) = x6.Value
arrayx(6) = x7.Value
arrayx(7) = x8.Value
arrayx(8) = x9.Value
arrayx(9) = x10.Value
arrayx(10) = x11.Value
arrayx(11) = x12.Value
arrayx(12) = x13.Value
arrayx(13) = x14.Value
arrayx(14) = x15.Value
Sheet1.Activate
```

'increments each value by the change necessary

'if previous value was 0 then the if loop does not execute

```
val = arrayx(2) - Cells(3, 2).Value
```

```
    If arrayx(2) <> 0 Then
```

```
        x3 = arrayx(2) + val
```

```
        arrayx(2) = x3
```

```
    End If
```

```
    If arrayx(3) <> 0 Then
```

```
        x4 = arrayx(3) + val
```

```
        arrayx(3) = x4
```

```
    End If
```

```
    If arrayx(4) <> 0 Then
```

```
        x5 = arrayx(4) + val
```

```
        arrayx(4) = x5
```

```
    End If
```

```
    If arrayx(5) <> 0 Then
```

```
        x6 = arrayx(5) + val
```

```
        arrayx(5) = x6
```

```
    End If
```

```
    If arrayx(6) <> 0 Then
```

```
        x7 = arrayx(6) + val
```

```
        arrayx(6) = x7
```

```
    End If
```

```
    If arrayx(7) <> 0 Then
```

```
        x8 = arrayx(7) + val
```

```
        arrayx(7) = x8
```

```
    End If
```

```
    If arrayx(8) <> 0 Then
```

```
        x9 = arrayx(8) + val
```

```
        arrayx(8) = x9
```

```
    End If
```

```
    If arrayx(9) <> 0 Then
```

```

x10 = arrayx(9) + val
arrayx(9) = x10
End If
If arrayx(10) <> 0 Then
x11 = arrayx(10) + val
arrayx(10) = x11
End If
If arrayx(11) <> 0 Then
x12 = arrayx(11) + val
arrayx(11) = x12
End If
If arrayx(12) <> 0 Then
x13 = arrayx(12) + val
arrayx(12) = x13
End If
If arrayx(13) <> 0 Then
x14 = arrayx(13) + val
arrayx(13) = x14
End If
If arrayx(14) <> 0 Then
x15 = arrayx(14) + val
arrayx(14) = x15
End If

```

'uploads the array to the excel sheet

```

Dim Destination As Range
Set Destination = Range("B1")
Set Destination = Destination.Resize(UBound(arrayx), 1)
Destination.Value = Application.Transpose(arrayx)

```

End Sub

Private Sub x4_AfterUpdate()

Dim val As Long

Dim arrayx(14) As Variant

```

arrayx(0) = x1.Value
arrayx(1) = x2.Value
arrayx(2) = x3.Value
arrayx(3) = x4.Value
arrayx(4) = x5.Value
arrayx(5) = x6.Value
arrayx(6) = x7.Value
arrayx(7) = x8.Value
arrayx(8) = x9.Value
arrayx(9) = x10.Value

```

```

arrayx(10) = x11.Value
arrayx(11) = x12.Value
arrayx(12) = x13.Value
arrayx(13) = x14.Value
arrayx(14) = x15.Value
Sheet1.Activate
val = arrayx(3) - Cells(4, 2).Value
    If arrayx(3) <> 0 Then
        x4 = arrayx(3) + val
        arrayx(3) = x4
    End If
    If arrayx(4) <> 0 Then
        x5 = arrayx(4) + val
        arrayx(4) = x5
    End If
    If arrayx(5) <> 0 Then
        x6 = arrayx(5) + val
        arrayx(5) = x6
    End If
    If arrayx(6) <> 0 Then
        x7 = arrayx(6) + val
        arrayx(6) = x7
    End If
    If arrayx(7) <> 0 Then
        x8 = arrayx(7) + val
        arrayx(7) = x8
    End If
    If arrayx(8) <> 0 Then
        x9 = arrayx(8) + val
        arrayx(8) = x9
    End If
    If arrayx(9) <> 0 Then
        x10 = arrayx(9) + val
        arrayx(9) = x10
    End If
    If arrayx(10) <> 0 Then
        x11 = arrayx(10) + val
        arrayx(10) = x11
    End If
    If arrayx(11) <> 0 Then
        x12 = arrayx(11) + val
        arrayx(11) = x12
    End If
    If arrayx(12) <> 0 Then
        x13 = arrayx(12) + val
        arrayx(12) = x13

```

```

End If
If arrayx(13) <> 0 Then
x14 = arrayx(13) + val
arrayx(13) = x14
End If
If arrayx(14) <> 0 Then
x15 = arrayx(14) + val
arrayx(14) = x15
End If

```

```

Dim Destination As Range
Set Destination = Range("B1")
Set Destination = Destination.Resize(UBound(arrayx), 1)
Destination.Value = Application.Transpose(arrayx)

```

```

End Sub
Private Sub x5_AfterUpdate()
Dim val As Long
Dim arrayx(14) As Variant

```

```

arrayx(0) = x1.Value
arrayx(1) = x2.Value
arrayx(2) = x3.Value
arrayx(3) = x4.Value
arrayx(4) = x5.Value
arrayx(5) = x6.Value
arrayx(6) = x7.Value
arrayx(7) = x8.Value
arrayx(8) = x9.Value
arrayx(9) = x10.Value
arrayx(10) = x11.Value
arrayx(11) = x12.Value
arrayx(12) = x13.Value
arrayx(13) = x14.Value
arrayx(14) = x15.Value
Sheet1.Activate
val = arrayx(4) - Cells(5, 2).Value
If arrayx(4) <> 0 Then
x5 = arrayx(4) + val
arrayx(4) = x5
End If
If arrayx(5) <> 0 Then
x6 = arrayx(5) + val
arrayx(5) = x6

```

```

End If
If arrayx(6) <> 0 Then
x7 = arrayx(6) + val
arrayx(6) = x7
End If
If arrayx(7) <> 0 Then
x8 = arrayx(7) + val
arrayx(7) = x8
End If
If arrayx(8) <> 0 Then
x9 = arrayx(8) + val
arrayx(8) = x9
End If
If arrayx(9) <> 0 Then
x10 = arrayx(9) + val
arrayx(9) = x10
End If
If arrayx(10) <> 0 Then
x11 = arrayx(10) + val
arrayx(10) = x11
End If
If arrayx(11) <> 0 Then
x12 = arrayx(11) + val
arrayx(11) = x12
End If
If arrayx(12) <> 0 Then
x13 = arrayx(12) + val
arrayx(12) = x13
End If
If arrayx(13) <> 0 Then
x14 = arrayx(13) + val
arrayx(13) = x14
End If
If arrayx(14) <> 0 Then
x15 = arrayx(14) + val
arrayx(14) = x15
End If

```

```

Dim Destination As Range
Set Destination = Range("B1")
Set Destination = Destination.Resize(UBound(arrayx), 1)
Destination.Value = Application.Transpose(arrayx)

```

```

End Sub
Private Sub x6_AfterUpdate()

```

```
Dim val As Long
Dim arrayx(14) As Variant
```

```
arrayx(0) = x1.Value
arrayx(1) = x2.Value
arrayx(2) = x3.Value
arrayx(3) = x4.Value
arrayx(4) = x5.Value
arrayx(5) = x6.Value
arrayx(6) = x7.Value
arrayx(7) = x8.Value
arrayx(8) = x9.Value
arrayx(9) = x10.Value
arrayx(10) = x11.Value
arrayx(11) = x12.Value
arrayx(12) = x13.Value
arrayx(13) = x14.Value
arrayx(14) = x15.Value
Sheet1.Activate
val = arrayx(5) - Cells(6, 2).Value
    If arrayx(5) <> 0 Then
        x6 = arrayx(5) + val
        arrayx(5) = x6
    End If
    If arrayx(6) <> 0 Then
        x7 = arrayx(6) + val
        arrayx(6) = x7
    End If
    If arrayx(7) <> 0 Then
        x8 = arrayx(7) + val
        arrayx(7) = x8
    End If
    If arrayx(8) <> 0 Then
        x9 = arrayx(8) + val
        arrayx(8) = x9
    End If
    If arrayx(9) <> 0 Then
        x10 = arrayx(9) + val
        arrayx(9) = x10
    End If
    If arrayx(10) <> 0 Then
        x11 = arrayx(10) + val
        arrayx(10) = x11
    End If
    If arrayx(11) <> 0 Then
```

```

x12 = arrayx(11) + val
arrayx(11) = x12
End If
If arrayx(12) <> 0 Then
x13 = arrayx(12) + val
arrayx(12) = x13
End If
If arrayx(13) <> 0 Then
x14 = arrayx(13) + val
arrayx(13) = x14
End If
If arrayx(14) <> 0 Then
x15 = arrayx(14) + val
arrayx(14) = x15
End If

```

```

Dim Destination As Range
Set Destination = Range("B1")
Set Destination = Destination.Resize(UBound(arrayx), 1)
Destination.Value = Application.Transpose(arrayx)

```

```

End Sub
Private Sub x7_AfterUpdate()
Dim val As Long
Dim arrayx(14) As Variant

```

```

arrayx(0) = x1.Value
arrayx(1) = x2.Value
arrayx(2) = x3.Value
arrayx(3) = x4.Value
arrayx(4) = x5.Value
arrayx(5) = x6.Value
arrayx(6) = x7.Value
arrayx(7) = x8.Value
arrayx(8) = x9.Value
arrayx(9) = x10.Value
arrayx(10) = x11.Value
arrayx(11) = x12.Value
arrayx(12) = x13.Value
arrayx(13) = x14.Value
arrayx(14) = x15.Value
Sheet1.Activate
val = arrayx(6) - Cells(7, 2).Value
If arrayx(6) <> 0 Then

```

```

x7 = arrayx(6) + val
arrayx(6) = x7
End If
If arrayx(7) <> 0 Then
x8 = arrayx(7) + val
arrayx(7) = x8
End If
If arrayx(8) <> 0 Then
x9 = arrayx(8) + val
arrayx(8) = x9
End If
If arrayx(9) <> 0 Then
x10 = arrayx(9) + val
arrayx(9) = x10
End If
If arrayx(10) <> 0 Then
x11 = arrayx(10) + val
arrayx(10) = x11
End If
If arrayx(11) <> 0 Then
x12 = arrayx(11) + val
arrayx(11) = x12
End If
If arrayx(12) <> 0 Then
x13 = arrayx(12) + val
arrayx(12) = x13
End If
If arrayx(13) <> 0 Then
x14 = arrayx(13) + val
arrayx(13) = x14
End If
If arrayx(14) <> 0 Then
x15 = arrayx(14) + val
arrayx(14) = x15
End If

```

```

Dim Destination As Range
Set Destination = Range("B1")
Set Destination = Destination.Resize(UBound(arrayx), 1)
Destination.Value = Application.Transpose(arrayx)

```

```

End Sub
Private Sub x8_AfterUpdate()
Dim val As Long
Dim arrayx(14) As Variant

```



```

arrayx(0) = x1.Value
arrayx(1) = x2.Value
arrayx(2) = x3.Value
arrayx(3) = x4.Value
arrayx(4) = x5.Value
arrayx(5) = x6.Value
arrayx(6) = x7.Value
arrayx(7) = x8.Value
arrayx(8) = x9.Value
arrayx(9) = x10.Value
arrayx(10) = x11.Value
arrayx(11) = x12.Value
arrayx(12) = x13.Value
arrayx(13) = x14.Value
arrayx(14) = x15.Value
Sheet1.Activate
val = arrayx(7) - Cells(8, 2).Value
    If arrayx(7) <> 0 Then
        x8 = arrayx(7) + val
        arrayx(7) = x8
    End If
    If arrayx(8) <> 0 Then
        x9 = arrayx(8) + val
        arrayx(8) = x9
    End If
    If arrayx(9) <> 0 Then
        x10 = arrayx(9) + val
        arrayx(9) = x10
    End If
    If arrayx(10) <> 0 Then
        x11 = arrayx(10) + val
        arrayx(10) = x11
    End If
    If arrayx(11) <> 0 Then
        x12 = arrayx(11) + val
        arrayx(11) = x12
    End If
    If arrayx(12) <> 0 Then
        x13 = arrayx(12) + val
        arrayx(12) = x13
    End If
    If arrayx(13) <> 0 Then
        x14 = arrayx(13) + val
        arrayx(13) = x14

```

```

End If
If arrayx(14) <> 0 Then
x15 = arrayx(14) + val
arrayx(14) = x15
End If

```

```

Dim Destination As Range
Set Destination = Range("B1")
Set Destination = Destination.Resize(UBound(arrayx), 1)
Destination.Value = Application.Transpose(arrayx)

```

```

End Sub
Private Sub x9_AfterUpdate()
Dim val As Long
Dim arrayx(14) As Variant

```

```

arrayx(0) = x1.Value
arrayx(1) = x2.Value
arrayx(2) = x3.Value
arrayx(3) = x4.Value
arrayx(4) = x5.Value
arrayx(5) = x6.Value
arrayx(6) = x7.Value
arrayx(7) = x8.Value
arrayx(8) = x9.Value
arrayx(9) = x10.Value
arrayx(10) = x11.Value
arrayx(11) = x12.Value
arrayx(12) = x13.Value
arrayx(13) = x14.Value
arrayx(14) = x15.Value
Sheet1.Activate
val = arrayx(8) - Cells(9, 2).Value
If arrayx(8) <> 0 Then
x9 = arrayx(8) + val
arrayx(8) = x9
End If
If arrayx(9) <> 0 Then
x10 = arrayx(9) + val
arrayx(9) = x10
End If
If arrayx(10) <> 0 Then
x11 = arrayx(10) + val
arrayx(10) = x11

```

```

End If
If arrayx(11) <> 0 Then
x12 = arrayx(11) + val
arrayx(11) = x12
End If
If arrayx(12) <> 0 Then
x13 = arrayx(12) + val
arrayx(12) = x13
End If
If arrayx(13) <> 0 Then
x14 = arrayx(13) + val
arrayx(13) = x14
End If
If arrayx(14) <> 0 Then
x15 = arrayx(14) + val
arrayx(14) = x15
End If

```

```

Dim Destination As Range
Set Destination = Range("B1")
Set Destination = Destination.Resize(UBound(arrayx), 1)
Destination.Value = Application.Transpose(arrayx)

```

```

End Sub
Private Sub x10_AfterUpdate()
Dim val As Long
Dim arrayx(14) As Variant

```

```

arrayx(0) = x1.Value
arrayx(1) = x2.Value
arrayx(2) = x3.Value
arrayx(3) = x4.Value
arrayx(4) = x5.Value
arrayx(5) = x6.Value
arrayx(6) = x7.Value
arrayx(7) = x8.Value
arrayx(8) = x9.Value
arrayx(9) = x10.Value
arrayx(10) = x11.Value
arrayx(11) = x12.Value
arrayx(12) = x13.Value
arrayx(13) = x14.Value
arrayx(14) = x15.Value
Sheet1.Activate

```

```
val = arrayx(9) - Cells(10, 2).Value
```

```
    If arrayx(9) <> 0 Then
```

```
        x10 = arrayx(9) + val
```

```
        arrayx(9) = x10
```

```
    End If
```

```
    If arrayx(10) <> 0 Then
```

```
        x11 = arrayx(10) + val
```

```
        arrayx(10) = x11
```

```
    End If
```

```
    If arrayx(11) <> 0 Then
```

```
        x12 = arrayx(11) + val
```

```
        arrayx(11) = x12
```

```
    End If
```

```
    If arrayx(12) <> 0 Then
```

```
        x13 = arrayx(12) + val
```

```
        arrayx(12) = x13
```

```
    End If
```

```
    If arrayx(13) <> 0 Then
```

```
        x14 = arrayx(13) + val
```

```
        arrayx(13) = x14
```

```
    End If
```

```
    If arrayx(14) <> 0 Then
```

```
        x15 = arrayx(14) + val
```

```
        arrayx(14) = x15
```

```
    End If
```

```
Dim Destination As Range
```

```
Set Destination = Range("B1")
```

```
Set Destination = Destination.Resize(UBound(arrayx), 1)
```

```
Destination.Value = Application.Transpose(arrayx)
```

```
End Sub
```

```
Private Sub x11_AfterUpdate()
```

```
Dim val As Long
```

```
Dim arrayx(14) As Variant
```

```
arrayx(0) = x1.Value
```

```
arrayx(1) = x2.Value
```

```
arrayx(2) = x3.Value
```

```
arrayx(3) = x4.Value
```

```
arrayx(4) = x5.Value
```

```
arrayx(5) = x6.Value
```

```
arrayx(6) = x7.Value
```

```
arrayx(7) = x8.Value
```

```

arrayx(8) = x9.Value
arrayx(9) = x10.Value
arrayx(10) = x11.Value
arrayx(11) = x12.Value
arrayx(12) = x13.Value
arrayx(13) = x14.Value
arrayx(14) = x15.Value
Sheet1.Activate
val = arrayx(10) - Cells(11, 2).Value
    If arrayx(10) <> 0 Then
        x11 = arrayx(10) + val
        arrayx(10) = x11
    End If
    If arrayx(11) <> 0 Then
        x12 = arrayx(11) + val
        arrayx(11) = x12
    End If
    If arrayx(12) <> 0 Then
        x13 = arrayx(12) + val
        arrayx(12) = x13
    End If
    If arrayx(13) <> 0 Then
        x14 = arrayx(13) + val
        arrayx(13) = x14
    End If
    If arrayx(14) <> 0 Then
        x15 = arrayx(14) + val
        arrayx(14) = x15
    End If

Dim Destination As Range
Set Destination = Range("B1")
Set Destination = Destination.Resize(UBound(arrayx), 1)
Destination.Value = Application.Transpose(arrayx)

End Sub
Private Sub x12_AfterUpdate()
Dim val As Long
Dim arrayx(14) As Variant

arrayx(0) = x1.Value
arrayx(1) = x2.Value
arrayx(2) = x3.Value
arrayx(3) = x4.Value

```

```

arrayx(4) = x5.Value
arrayx(5) = x6.Value
arrayx(6) = x7.Value
arrayx(7) = x8.Value
arrayx(8) = x9.Value
arrayx(9) = x10.Value
arrayx(10) = x11.Value
arrayx(11) = x12.Value
arrayx(12) = x13.Value
arrayx(13) = x14.Value
arrayx(14) = x15.Value
Sheet1.Activate
val = arrayx(11) - Cells(12, 2).Value
    If arrayx(11) <> 0 Then
        x12 = arrayx(11) + val
        arrayx(11) = x12
    End If
    If arrayx(12) <> 0 Then
        x13 = arrayx(12) + val
        arrayx(12) = x13
    End If
    If arrayx(13) <> 0 Then
        x14 = arrayx(13) + val
        arrayx(13) = x14
    End If
    If arrayx(14) <> 0 Then
        x15 = arrayx(14) + val
        arrayx(14) = x15
    End If

Dim Destination As Range
Set Destination = Range("B1")
Set Destination = Destination.Resize(UBound(arrayx), 1)
Destination.Value = Application.Transpose(arrayx)

End Sub
Private Sub x13_AfterUpdate()
Dim val As Long
Dim arrayx(14) As Variant

arrayx(0) = x1.Value
arrayx(1) = x2.Value
arrayx(2) = x3.Value
arrayx(3) = x4.Value

```

```

arrayx(4) = x5.Value
arrayx(5) = x6.Value
arrayx(6) = x7.Value
arrayx(7) = x8.Value
arrayx(8) = x9.Value
arrayx(9) = x10.Value
arrayx(10) = x11.Value
arrayx(11) = x12.Value
arrayx(12) = x13.Value
arrayx(13) = x14.Value
arrayx(14) = x15.Value
Sheet1.Activate
val = arrayx(12) - Cells(13, 2).Value
    If arrayx(12) <> 0 Then
        x13 = arrayx(12) + val
        arrayx(12) = x13
    End If
    If arrayx(13) <> 0 Then
        x14 = arrayx(13) + val
        arrayx(13) = x14
    End If
    If arrayx(14) <> 0 Then
        x15 = arrayx(14) + val
        arrayx(14) = x15
    End If

Dim Destination As Range
Set Destination = Range("B1")
Set Destination = Destination.Resize(UBound(arrayx), 1)
Destination.Value = Application.Transpose(arrayx)

End Sub
Private Sub x14_AfterUpdate()
Dim val As Long
Dim arrayx(14) As Variant

arrayx(0) = x1.Value
arrayx(1) = x2.Value
arrayx(2) = x3.Value
arrayx(3) = x4.Value
arrayx(4) = x5.Value
arrayx(5) = x6.Value
arrayx(6) = x7.Value
arrayx(7) = x8.Value

```

```

arrayx(8) = x9.Value
arrayx(9) = x10.Value
arrayx(10) = x11.Value
arrayx(11) = x12.Value
arrayx(12) = x13.Value
arrayx(13) = x14.Value
arrayx(14) = x15.Value
Sheet1.Activate
val = arrayx(13) - Cells(14, 2).Value
    If arrayx(13) <> 0 Then
        x14 = arrayx(13) + val
        arrayx(13) = x14
    End If
    If arrayx(14) <> 0 Then
        x15 = arrayx(14) + val
        arrayx(14) = x15
    End If

```

```

Dim Destination As Range
Set Destination = Range("B1")
Set Destination = Destination.Resize(UBound(arrayx), 1)
Destination.Value = Application.Transpose(arrayx)

```

```

End Sub
Private Sub x15_AfterUpdate()
Dim arrayx(15)

```

```

    arrayx(15) = x15.Value

```

```

    For z = 15 To LastRowA
Cells(z, 1).Value = arrayx(z)

```

```

End Sub

```

```

*****
!*
```

```

'Allows the user to edit the variables without having to exit the macro
'There is one sub for each textbox with the feature 'f' variable
'Only the first will be explained
'for the special 's' variable the process is essentially the same
'this is because the event only happens if either 's' changes or 'f' and 's' both change
!*
```

```

*****

```

```

Private Sub f3_AfterUpdate()
Dim val As Double

```



```

'checks if the user has entered chamfer and if the chamfer value is different
If f3.Value = "Chamfer" And CDbl(s3.Value) <> Cells(3, 5).Value Then
    val = CDbl(s3.Value) - Cells(3, 5).Value
    'positive geometry chamfer
    If Cells(3, 6).Value > 0 Then
        z3 = CDbl(z3.Value) + val
    End If
    'negative geometry chamfer
    If Cells(3, 6).Value < 0 Then
        x3 = CDbl(z3.Value) + val
    End If
End If
'changes radius value
If f3.Value = "Radius" And CDbl(s3.Value) <> Cells(3, 4).Value Then
    Cells(3, 4).Value = s3.Value
End If
'if 'f' has been emptied then 's' is also emptied
If f3.Value = "" Then
    s3 = ""
End Sub
Private Sub f4_AfterUpdate()
Dim val As Double
    If f4.Value = "Chamfer" And CDbl(s4.Value) <> Cells(4, 5).Value Then
        val = CDbl(s4.Value) - Cells(4, 5).Value
        If Cells(4, 6).Value > 0 Then
            z4 = CDbl(z4.Value) + val
        End If
        If Cells(4, 6).Value < 0 Then
            x4 = CDbl(z4.Value) + val
        End If
    End If
    If f4.Value = "Radius" And CDbl(s4.Value) <> Cells(4, 4).Value Then
        Cells(4, 4).Value = s4.Value
    End If
    If f4.Value = "" Then
        s4 = ""
    End Sub
Private Sub f5_AfterUpdate()
Dim val As Double
    If f5.Value = "Chamfer" And CDbl(s5.Value) <> Cells(5, 5).Value Then
        val = CDbl(s5.Value) - Cells(5, 5).Value
        If Cells(5, 6).Value > 0 Then
            z5 = CDbl(z5.Value) + val
        End If
        If Cells(5, 6).Value < 0 Then
            x5 = CDbl(z5.Value) + val

```

```

        End If
    End If
    If f5.Value = "Radius" And CDBl(s5.Value) <> Cells(5, 4).Value Then
        Cells(5, 4).Value = s4.Value
    End If
    If f5.Value = "" Then
        s5 = ""
    End Sub
    Private Sub f6_AfterUpdate()
    Dim val As Double
    If f6.Value = "Chamfer" And CDBl(s6.Value) <> Cells(6, 5).Value Then
        val = CDBl(s6.Value) - Cells(6, 5).Value
        If Cells(6, 6).Value > 0 Then
            z6 = CDBl(z6.Value) + val
        End If
        If Cells(6, 6).Value < 0 Then
            x6 = CDBl(z6.Value) + val
        End If
    End If
    If f6.Value = "Radius" And CDBl(s6.Value) <> Cells(6, 4).Value Then
        Cells(6, 4).Value = s4.Value
    End If
    If f6.Value = "" Then
        s6 = ""
    End Sub
    Private Sub f7_AfterUpdate()
    Dim val As Double
    If f7.Value = "Chamfer" And CDBl(s7.Value) <> Cells(7, 5).Value Then
        val = CDBl(s7.Value) - Cells(7, 5).Value
        If Cells(7, 6).Value > 0 Then
            z7 = CDBl(z7.Value) + val
        End If
        If Cells(7, 6).Value < 0 Then
            x7 = CDBl(z7.Value) + val
        End If
    End If
    If f7.Value = "Radius" And CDBl(s7.Value) <> Cells(7, 4).Value Then
        Cells(7, 4).Value = s7.Value
    End If
    If f7.Value = "" Then
        s7 = ""
    End Sub
    Private Sub f8_AfterUpdate()
    Dim val As Double
    If f8.Value = "Chamfer" And CDBl(s8.Value) <> Cells(8, 5).Value Then
        val = CDBl(s8.Value) - Cells(8, 5).Value

```

```

    If Cells(8, 6).Value > 0 Then
        z8 = CDbl(z8.Value) + val
    End If
    If Cells(8, 6).Value < 0 Then
        x8 = CDbl(z8.Value) + val
    End If
End If
If f8.Value = "Radius" And CDbl(s8.Value) <> Cells(8, 4).Value Then
    Cells(8, 4).Value = s8.Value
End If
If f8.Value = "" Then
    s8 = ""
End Sub
Private Sub f9_AfterUpdate()
Dim val As Double
    If f9.Value = "Chamfer" And CDbl(s9.Value) <> Cells(9, 5).Value Then
        val = CDbl(s9.Value) - Cells(9, 5).Value
        If Cells(9, 6).Value > 0 Then
            z9 = CDbl(z9.Value) + val
        End If
        If Cells(9, 6).Value < 0 Then
            x9 = CDbl(z9.Value) + val
        End If
    End If
    If f9.Value = "Radius" And CDbl(s9.Value) <> Cells(9, 4).Value Then
        Cells(9, 4).Value = s9.Value
    End If
    If f9.Value = "" Then
        s9 = ""
    End Sub
Private Sub f10_AfterUpdate()
Dim val As Double
i = 10
    If f10.Value = "Chamfer" And CDbl(s10.Value) <> Cells(i, 5).Value Then
        val = CDbl(s10.Value) - Cells(i, 5).Value
        If Cells(i, 6).Value > 0 Then
            z10 = CDbl(z10.Value) + val
        End If
        If Cells(i, 6).Value < 0 Then
            x10 = CDbl(z10.Value) + val
        End If
    End If
    If f10.Value = "Radius" And CDbl(s10.Value) <> Cells(i, 4).Value Then
        Cells(i, 4).Value = s10.Value
    End If
    If f10.Value = "" Then

```

```

        s10 = ""
End Sub
Private Sub f11_AfterUpdate()
Dim val As Double
i = 11
    If f11.Value = "Chamfer" And CDbl(s11.Value) <> Cells(i, 5).Value Then
        val = CDbl(s11.Value) - Cells(i, 5).Value
        If Cells(i, 6).Value > 0 Then
            z11 = CDbl(z11.Value) + val
        End If
        If Cells(i, 6).Value < 0 Then
            x11 = CDbl(z11.Value) + val
        End If
    End If
    If f11.Value = "Radius" And CDbl(s11.Value) <> Cells(i, 4).Value Then
        Cells(i, 4).Value = s11.Value
    End If
    If f11.Value = "" Then
        s11 = ""
    End Sub
Private Sub f12_AfterUpdate()
Dim val As Double
i = 12
    If f12.Value = "Chamfer" And CDbl(s12.Value) <> Cells(i, 5).Value Then
        val = CDbl(s12.Value) - Cells(i, 5).Value
        If Cells(i, 6).Value > 0 Then
            z12 = CDbl(z12.Value) + val
        End If
        If Cells(i, 6).Value < 0 Then
            x12 = CDbl(z12.Value) + val
        End If
    End If
    If f12.Value = "Radius" And CDbl(s12.Value) <> Cells(i, 4).Value Then
        Cells(i, 4).Value = s12.Value
    End If
    If f12.Value = "" Then
        s12 = ""
    End Sub
Private Sub f13_AfterUpdate()
Dim val As Double
i = 13
    If f13.Value = "Chamfer" And CDbl(s13.Value) <> Cells(i, 5).Value Then
        val = CDbl(s13.Value) - Cells(i, 5).Value
        If Cells(i, 6).Value > 0 Then
            z13 = CDbl(z13.Value) + val
        End If

```

```

        If Cells(i, 6).Value < 0 Then
            x13 = CDbl(z13.Value) + val
        End If
    End If
    If f13.Value = "Radius" And CDbl(s13.Value) <> Cells(i, 4).Value Then
        Cells(i, 4).Value = s13.Value
    End If
    If f13.Value = "" Then
        s13 = ""
    End Sub
    Private Sub s3_AfterUpdate()
    Dim val As Double
        If f3.Value = "Chamfer" And CDbl(s3.Value) <> Cells(3, 5).Value Then
            val = CDbl(s3.Value) - Cells(3, 5).Value
            If Cells(3, 6).Value > 0 Then
                z3 = CDbl(z3.Value) + val
            End If
            If Cells(3, 6).Value < 0 Then
                x3 = CDbl(z3.Value) + val
            End If
        End If
        If f3.Value = "Radius" And CDbl(s3.Value) <> Cells(3, 4).Value Then
            Cells(3, 4).Value = s3.Value
        End If
        If f3.Value = "" Then
            s3 = ""
        End Sub
    Private Sub s4_AfterUpdate()
    Dim val As Double
        If f4.Value = "Chamfer" And CDbl(s4.Value) <> Cells(4, 5).Value Then
            val = CDbl(s4.Value) - Cells(4, 5).Value
            If Cells(4, 6).Value > 0 Then
                z4 = CDbl(z4.Value) + val
            End If
            If Cells(4, 6).Value < 0 Then
                x4 = CDbl(z4.Value) + val
            End If
        End If
        If f4.Value = "Radius" And CDbl(s4.Value) <> Cells(4, 4).Value Then
            Cells(4, 4).Value = s4.Value
        End If
        If f4.Value = "" Then
            s4 = ""
        End Sub
    Private Sub s5_AfterUpdate()
    Dim val As Double

```

```

If f5.Value = "Chamfer" And CDbl(s5.Value) <> Cells(5, 5).Value Then
    val = CDbl(s5.Value) - Cells(5, 5).Value
    If Cells(5, 6).Value > 0 Then
        z5 = CDbl(z5.Value) + val
    End If
    If Cells(5, 6).Value < 0 Then
        x5 = CDbl(z5.Value) + val
    End If
End If
If f5.Value = "Radius" And CDbl(s5.Value) <> Cells(5, 4).Value Then
    Cells(5, 4).Value = s4.Value
End If
If f5.Value = "" Then
    s5 = ""
End Sub
Private Sub s6_AfterUpdate()
Dim val As Double
    If f6.Value = "Chamfer" And CDbl(s6.Value) <> Cells(6, 5).Value Then
        val = CDbl(s6.Value) - Cells(6, 5).Value
        If Cells(6, 6).Value > 0 Then
            z6 = CDbl(z6.Value) + val
        End If
        If Cells(6, 6).Value < 0 Then
            x6 = CDbl(z6.Value) + val
        End If
    End If
    If f6.Value = "Radius" And CDbl(s6.Value) <> Cells(6, 4).Value Then
        Cells(6, 4).Value = s4.Value
    End If
    If f6.Value = "" Then
        s6 = ""
    End Sub
Private Sub s7_AfterUpdate()
Dim val As Double
    If f7.Value = "Chamfer" And CDbl(s7.Value) <> Cells(7, 5).Value Then
        val = CDbl(s7.Value) - Cells(7, 5).Value
        If Cells(7, 6).Value > 0 Then
            z7 = CDbl(z7.Value) + val
        End If
        If Cells(7, 6).Value < 0 Then
            x7 = CDbl(z7.Value) + val
        End If
    End If
    If f7.Value = "Radius" And CDbl(s7.Value) <> Cells(7, 4).Value Then
        Cells(7, 4).Value = s7.Value
    End If

```

```

    If f7.Value = "" Then
        s7 = ""
    End Sub
    Private Sub s8_AfterUpdate()
    Dim val As Double
        If f8.Value = "Chamfer" And CDbl(s8.Value) <> Cells(8, 5).Value Then
            val = CDbl(s8.Value) - Cells(8, 5).Value
            If Cells(8, 6).Value > 0 Then
                z8 = CDbl(z8.Value) + val
            End If
            If Cells(8, 6).Value < 0 Then
                x8 = CDbl(z8.Value) + val
            End If
        End If
        If f8.Value = "Radius" And CDbl(s8.Value) <> Cells(8, 4).Value Then
            Cells(8, 4).Value = s8.Value
        End If
        If f8.Value = "" Then
            s8 = ""
        End Sub
    Private Sub s9_AfterUpdate()
    Dim val As Double
        If f9.Value = "Chamfer" And CDbl(s9.Value) <> Cells(9, 5).Value Then
            val = CDbl(s9.Value) - Cells(9, 5).Value
            If Cells(9, 6).Value > 0 Then
                z9 = CDbl(z9.Value) + val
            End If
            If Cells(9, 6).Value < 0 Then
                x9 = CDbl(z9.Value) + val
            End If
        End If
        If f9.Value = "Radius" And CDbl(s9.Value) <> Cells(9, 4).Value Then
            Cells(9, 4).Value = s9.Value
        End If
        If f9.Value = "" Then
            s9 = ""
        End Sub
    Private Sub s10_AfterUpdate()
    Dim val As Double
    i = 10
        If f10.Value = "Chamfer" And CDbl(s10.Value) <> Cells(i, 5).Value Then
            val = CDbl(s10.Value) - Cells(i, 5).Value
            If Cells(i, 6).Value > 0 Then
                z10 = CDbl(z10.Value) + val
            End If
            If Cells(i, 6).Value < 0 Then

```

```

        x10 = CDBl(z10.Value) + val
    End If
End If
If f10.Value = "Radius" And CDBl(s10.Value) <> Cells(i, 4).Value Then
    Cells(i, 4).Value = s10.Value
End If
If f10.Value = "" Then
    s10 = ""
End Sub
Private Sub s11_AfterUpdate()
Dim val As Double
i = 11
    If f11.Value = "Chamfer" And CDBl(s11.Value) <> Cells(i, 5).Value Then
        val = CDBl(s11.Value) - Cells(i, 5).Value
        If Cells(i, 6).Value > 0 Then
            z11 = CDBl(z11.Value) + val
        End If
        If Cells(i, 6).Value < 0 Then
            x11 = CDBl(z11.Value) + val
        End If
    End If
    If f11.Value = "Radius" And CDBl(s11.Value) <> Cells(i, 4).Value Then
        Cells(i, 4).Value = s11.Value
    End If
    If f11.Value = "" Then
        s11 = ""
    End Sub
Private Sub s12_AfterUpdate()
Dim val As Double
i = 12
    If f12.Value = "Chamfer" And CDBl(s12.Value) <> Cells(i, 5).Value Then
        val = CDBl(s12.Value) - Cells(i, 5).Value
        If Cells(i, 6).Value > 0 Then
            z12 = CDBl(z12.Value) + val
        End If
        If Cells(i, 6).Value < 0 Then
            x12 = CDBl(z12.Value) + val
        End If
    End If
    If f12.Value = "Radius" And CDBl(s12.Value) <> Cells(i, 4).Value Then
        Cells(i, 4).Value = s12.Value
    End If
    If f12.Value = "" Then
        s12 = ""
    End Sub
Private Sub s13_AfterUpdate()

```



```

Dim val As Double
i = 13
If f13.Value = "Chamfer" And CDBl(s13.Value) <> Cells(i, 5).Value Then
    val = CDBl(s13.Value) - Cells(i, 5).Value
    If Cells(i, 6).Value > 0 Then
        z13 = CDBl(z13.Value) + val
    End If
    If Cells(i, 6).Value < 0 Then
        x13 = CDBl(z13.Value) + val
    End If
End If
If f13.Value = "Radius" And CDBl(s13.Value) <> Cells(i, 4).Value Then
    Cells(i, 4).Value = s13.Value
End If
If f13.Value = "" Then
    s13 = ""
End Sub

```

2)VBScript

```

*****
'*This simply uploads the Catalog File into Catia
'*Catia produces a catalog file
'*It then must be placed within the proper folder
*****

```

```
Language="VBSCRIPT"
```

```
Sub CATMain()
```

```
InputFile="C:\Users\Raisanen\Documents\CATIA\Raisanen Thesis\Catalog.csv"
```

```
OutputFile="C:\Users\Raisanen\Documents\CATIA\Tool_Library.catalog"
```

```
Dim Catalog As Document
```

```
set Catalog=CATIA.Documents.Add("CatalogDocument")
```

```
Catalog.CreateCatalogFromcsv InputFile, OutputFile
```

```
Catalog.Close
```

```
End Sub
```