CHARACTERIZATION OF BIDIMENSIONAL GEOMETRY THROUGH SOLID MODELING SOFTWARE USING A C++ INTERFACE

By

Neit Josafat Nieves-Flores

A project submitted in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING

IN

MECHANICAL ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS
2009

Approved by:

_____	_____
Vijay K. Goyal, Ph.D. Chair, Graduate Committee	Date
_____	_____
Ricky Valentín, Ph.D. Member, Graduate Committee	Date
_____	_____
Stefano Leonardi, Ph.D. Member, Graduate Committee	Date
_____	_____
Lev Steinberg, Ph. D. Representative of Graduate Studies	Date
_____	_____
Pablo Cáceres, Ph.D. Chairperson of the Department	Date

CHARACTERIZATION OF BIDIMENSIONAL GEOMETRY THROUGH SOLID MODELING SOFTWARE USING A C++ INTERFACE

ABSTRACT

In any design process a detailed description and communication of design ideas or specifications is essential. As part of the final design description are the engineering drawings that represent a detailed characterization of the design geometry. In some cases, the final design may be corrupted with additional geometry, which is not part of the problem, and lack of dimensions. In such a case, we will have to remove any unwanted items manually before being able to apply dimensions and constraints to characterize the geometry. Here we develop a software tool, which integrates UGS-NX5 and C++, to improve the characterization process for bidimensional geometry. The tool was successfully applied to several different bidimensional cross-sections; it is effective in the removal of unwanted objects from the geometry and the application of dimensions and constraints to the selected parts within the geometry. This makes the characterization process faster and user-friendly.

CARACTERIZACIÓN DE GEOMETRIA BIDIMENSIONAL EN UN PROGRAMA DE MODELACION USANDO UNA INTERFACE DE C++

RESUMEN

En todo proceso de diseño una descripción detallada y la comunicación de ideas de diseño o especificaciones son esenciales. Parte de la descripción final del diseño son los dibujos de ingeniería que representan la caracterización detallada de la geometría del diseño. En algunos casos el diseño final puede resultar corrupto con geometría adicional, que no es parte del diseño, y falta de dimensiones. En tal caso, tendríamos que remover manualmente cualquier objeto no deseado antes de poder aplicar dimensiones y restricciones para caracterizar la geometría. Aquí desarrollamos una herramienta, que integra UGS-NX5 y C++, para mejorar el proceso de caracterización de geometría bidimensional. Esta herramienta fue aplicada exitosamente a varias secciones bidimensionales; resultó ser efectiva en la remoción de objetos indeseados de la geometría y la aplicación de dimensiones y restricciones a las partes seleccionadas en la geometría. Esto hace que el proceso de caracterización sea más rápido y de fácil aplicación.

Faculty Advisor: Dr. Vijay K. Goyal
Graduate Student: Neit J. Nieves-Flores

ACKNOWLEDGEMENTS

Thanks to God, my Mother, Sister and other members of my Family, my girlfriend Karina, my Friends Victor, Manuel, Danny, Mical and others in Puerto Rico (PR) and Michigan (MI) for their love, support and encouragement. Thanks to my faculty advisor Dr. Vijay K. Goyal, the Mechanical Engineering Department Chairs Dr. Paul Sundaram and Dr. Pablo Cáceres and the members of my Graduate Committee for their wisdom, continuous support and guidance, and for giving me the opportunity to return to PR to achieve this goal. Thanks to Catalina Camacho for her continuous and always friendly support, to Evangeline Jiménez for her friendship and encouragement, and to everyone in the Mechanical Engineering Faculty that were part of this achievement. Very special thanks and appreciation to my great friends Juan J. Reinés, José E. Lugo, Byron Zambrano, and all other friends and colleagues in and out of the Mechanical Engineering Department for being there, for their friendship, giving me support and advice when needed and in all, making this experience the best it could have possibly been. Thank you!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: PRELIMINARY REMARKS

1.1 Background

The communication of design ideas and specifications through the use of engineering drawings is an essential part of the design and manufacturing processes. Dimensions and constraints are characteristics used in these drawings to describe a design geometrically and functionally. This detailed description of the design is necessary before any manufacturing process can be performed. Through this work we will develop a tool to improve the characterization of bidimensional geometry by removing any unwanted, or remaining, objects in the drawing and applying desired constraints and dimensions. Excess lines or other unwanted geometry may remain visible in a drawing when a bidimensional section is taken from a three-dimensional CAD object due to existing geometry outside of the plane of interest. Figure 1.1 gives an example of unwanted objects in the drawing; this tool will provide us with a clean bidimensional drawing with the user selected objects constrained and dimensioned, as shown in Figure 1.2.

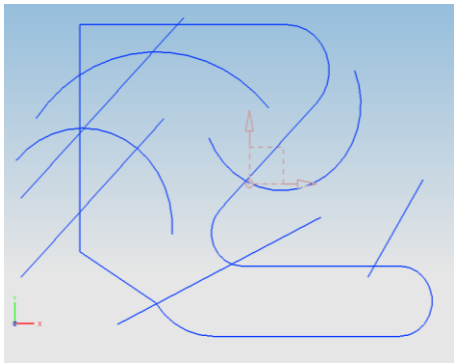

Figure 1.1: Geometry with Unwanted Objects

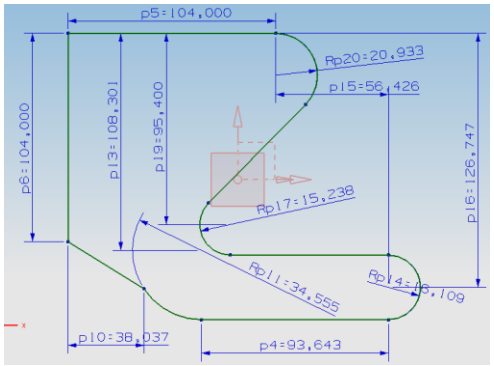

Figure 1.2: Resulting Geometry

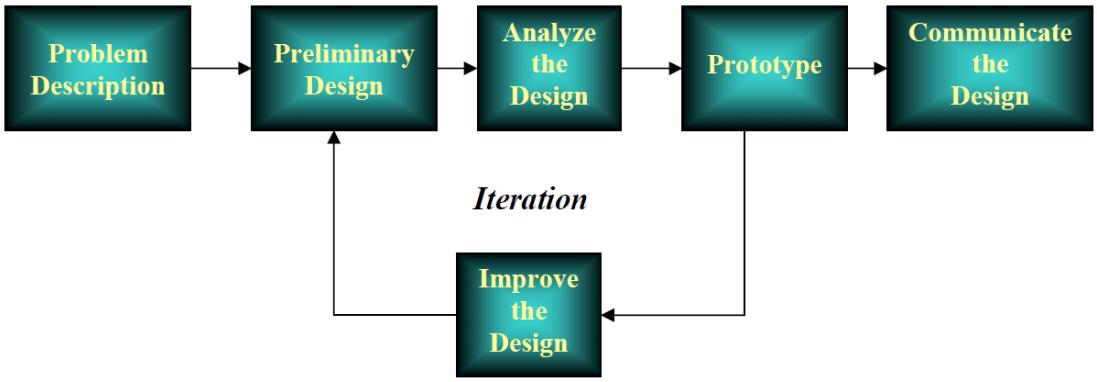

Figure 1.3: Product Design Process (Goyal, 2008).

The design process is iterative in nature, as shown in Figure 1.3. Often it may be necessary to modify a design several times in order to reach an optimized design to meet specifications. A detailed characterization of the design is thus required to allow all the members of a design or manufacturing team to understand it, reducing errors due to insufficient of information. This work is an integral part of the design analysis and improvement iteration loop as shown above. It facilitates the analysis of bidimensional sections of new or existing designs and leads to the optimization of the design and more efficient communication of the design geometry.

In certain circumstances, the user (or designer) may face challenges when analyzing a design due to the great variety of Solid Modeling software packages available and the compatibility issues they may have. Another challenge arises when a designer wants to analyze a bidimensional section taken from a three-dimensional model. To obtain this section an intersecting plane is used at the required location to "cut" the geometry and allow

the designer to have a closer look. Using this technique, out-of-plane objects, guide lines, or other excess or unnecessary objects may remain in the section view and make the characterization of the desired geometry more challenging, as shown in Figure 1.1.

Hence, in this work we propose the development of a software tool that helps the designer in the removal of unnecessary objects and application of dimensions and constraints to the desired parts of a bidimensional section. The bidimensional drawing characterization process will become easier and faster, helping the designer in the completion and optimization of the design. This is the major contribution of this work.

1.2 Literature Survey

1.2.1 Overview

Solid modeling is the representation of the solid parts of an object, that is, models of solid objects suitable for computer processing. Solid modeling software creates a virtual representation of components for machine design, analysis and manufacturing. There are many ways to describe a solid model some of which are parametric modeling, using parameters in the description of the model, or constraint-based modeling which uses its dimensions. Solid Modeling encompasses many tools and descriptions that were not part of the scope of this work; a simple representation as described here is shown in Figure 1.4. These modeling methods allow the designer to accurately describe a design. Our work is part of the modeling and analysis of bidimensional views taken from three-dimensional solid models.

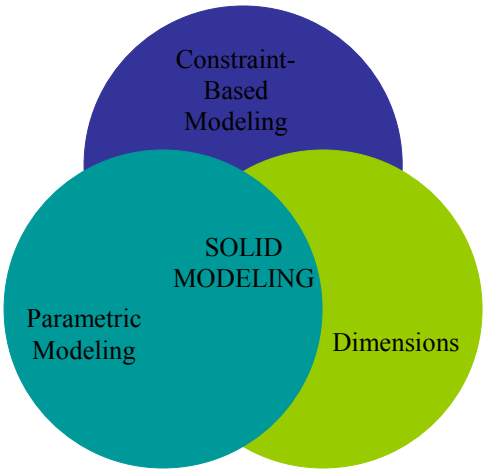

Figure 1.4: Simple Representation of Solid Modeling.

1.2.2 Parametric Modeling

Parametric modeling is a characterization process that uses parameters to define a model. The parameters may be modified later, and the model will update to reflect the modification. Figure 1.5 shows an airfoil model geometry constrained by user defined parameters. These parameters are highlighted on the left. Typically, there is a relationship between parts, assemblies, and drawings. A part consists of multiple features, and an assembly consists of multiple parts. Drawings can be made from either parts or assemblies. Constraint-based geometry can be visualized as a general framework from which a parameterized system is a special case. Parametric design specification is one of the possible alternatives available within a constraint-based representation (Dones, 1991).

Maarten and Van (1989) discussed the creation and modification of parametric solid models by graphical interaction. Their primary concern was to define relations between geometric properties of constructive solid geometry primitive. The system they describe is

capable of changing dimensions and position of a geometric model as a whole, without changing the object description. The system cannot handle user defined constraints and does not address the removal of any excess geometry, restricting the editing and characterization process.

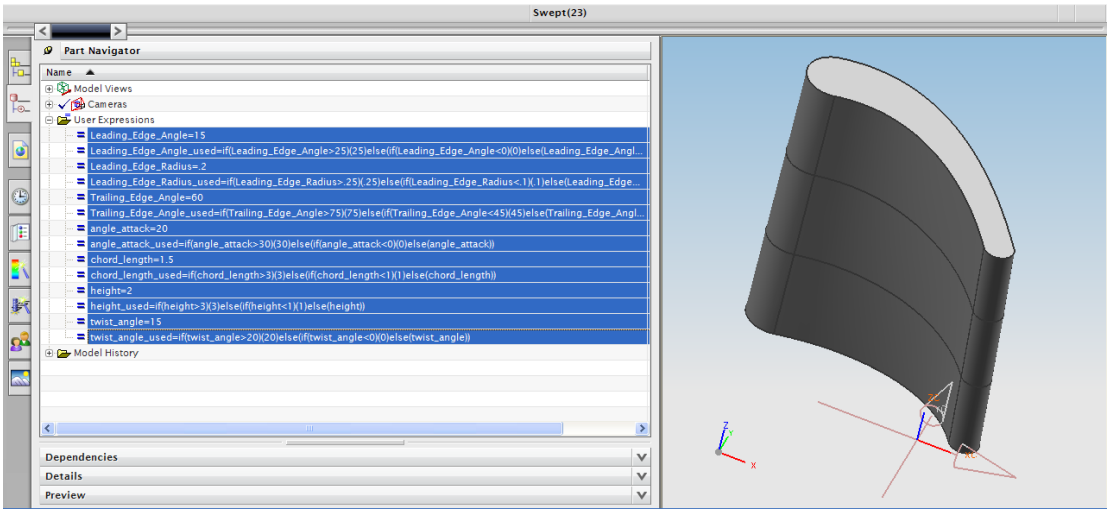

Figure 1.5: Parameterized Airfoil Geometry with List of Parameters Displayed

1.2.3 Constraint-Based Modeling

Constraint-based geometry is a technique by which an arbitrary geometry is defined or constrained using its dimensions, as shown in Figure 1.6. In constraint-based geometry an object is determined by a number of characteristic points in a three-dimensional space. Instead of using the geometry to define the dimensions, constraint-based geometry uses the dimensions to define its geometry. The dimensions are used to relate geometrical elements such as points, lines, arcs or circles. These relations can be graphically presented in the design's drawings using dimension parameters. Dimensioning and constraint-based geometry help the designer in the definition of the object and the use of geometrical dimensions as part of design specifications. Our work allows a designer to define constraints to selected objects within bidimensional geometry.

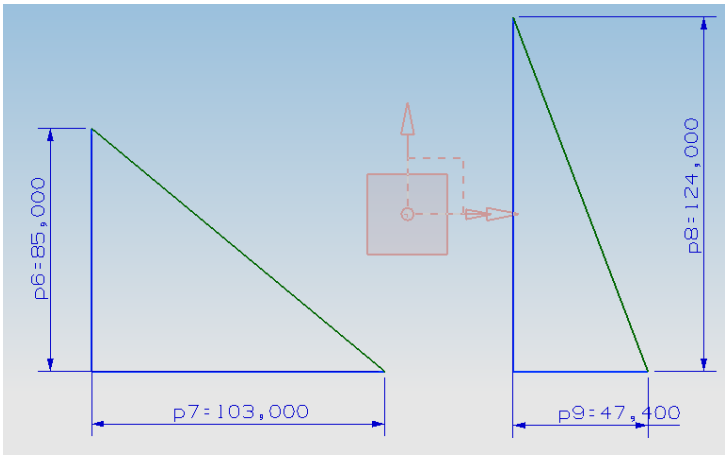

Figure 1.6: Triangular Geometry Constrained by its Dimensions

Light et al (1982) discuss a constraint-based method for modification of geometric models. They present a procedure for minimizing computational effort in the solution of the constraints and a procedure for detecting invalid dimensioning schemes. Suzuki et al (1990) discuss the importance of geometric constraints and reasoning in CAD systems. One characteristic of design is constraint solving, an essential portion of these constraints being geometrical. They present a consistent framework for representing geometric models and constraints, and a geometric reasoning mechanism to solve those constraints. Their implementation is limited to the bidimensional case and they use constraint propagation to determine the parameter values. They clearly state that their method can only make geometrical changes to a fixed topology.

Pérez (1993) proposed a methodology to be implemented in a three dimensional constraint-based finite element modeler to allow a designer to interactively construct a geometric model by dimensional changes which are propagated to the finite element model. The system also allowed for optimization analysis without requiring explicit parametric definitions from the user. Pérez validated the system by conducting experiments where a significant time saving in the process was observed compared to previously available systems.

All of these systems propose mathematical methods for constraint or dimension calculation and management but do not provide an actual application to use in an existing drawing or an application for removal of objects that are not needed in the geometry.

1.2.4 Dimensioning

The purpose of dimensioning engineering design drawings is to help the designer understand the spatial arrangement of the objects being drawn, as shown in Figure 1.7. Dimensions for size and position must be complete so that the part is defined for manufacture. There is more than one way to dimension single parts (Dones, 1991). The dimensioning process must be performed individually for each component in a design. A part is defined as a collection of geometric elements having a closed topology. Two parts do not share geometric elements and can be isolated without changing their individual characteristics (Dones, 1991). This project allows the user to apply dimensions to the desired parts of the bidimensional geometry, as shown in Figure 1.7.

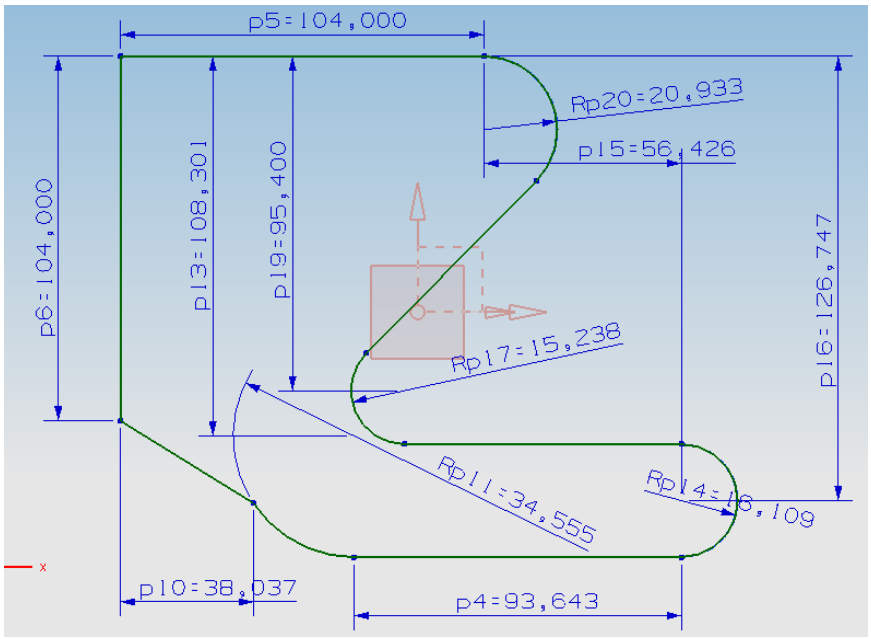

Figure 1.7: Example Dimensioning Scheme

Yuen et al (1988) developed a method to automatically generate dimensions based on boundary representations of solid models expressed as linear and angular dimensions. This solid modeling system can generate an adequate dimensioning layout for a defined solid. Light and Gossard (1982) presented a procedure that represents a geometry by a set of dimensions. Dimensions are used as constraints limiting the locations of the characteristic points of the object. Aldefield (1988) represented a geometric model by a rule-based system for propagating constraint information on geometric structures (Pabón, 1996).

Jaramillo (1993) presented a methodology for the automatic dimensioning and use of tolerances for bidimensional geometry. Dones (1991) presented an automatic generation of dimensioning schemes using constraint-based geometry. Dimensions are represented as dimensional constraint equations, in terms of the characteristic points defining the geometry and dimension parameters. Dones (1991) determined possible dimensioning schemes using a modification of the constraint management theory as presented by Serrano (1987, 1991). Pabón (1996) proposed an automatic dimensioning layout methodology to help any CAD system with automatic dimensioning tools to properly locate the generated dimensions on the drawing layout using an intelligent rule based system.

Mathematical methods for constraint or dimension calculation and management are presented by these systems but do not provide an application to use in an existing drawing or an application for removal of unwanted objects from the geometry.

1.3 Problem Description

Many tools are currently available for assisting an engineer, or designer, in the dimensioning or constraint characterization process of a solid model or bidimensional drawing. A good example includes applications designed for products such as Autodesk AutoCAD among others (Borduin, 2000). These systems and tools each advance the design process and help make dimensioning or constraint application more useful within the modeling software environment in use but may require significant and time consuming user input to achieve the desired results.

Current solid modeling software requires manual and time consuming manipulation of the geometry for the removal of excess objects that may remain visible in a bidimensional section taken from a three-dimensional solid model being analyzed. Currently, the user must carefully select the excess objects and manually remove them. This practice can be quite cumbersome as it is sometimes difficult to differentiate parts of the actual design from unnecessary or out-of-plane objects. This process will take a tremendous amount of time if the design consists of hundreds of objects. Also the dimensioning and constraint application will be difficult to accomplish, usually requiring multiple steps in the process.

The problem considered throughout this work consists in developing a more integrated software tool for characterizing bidimensional geometry that automatically removes excess geometry from the cross section to be analyzed and advances the user application of dimensions and geometric constraints. Such a tool will help the designer reduce the characterization time and will make the work easier to accomplish. Also, by pursuing the development of this tool in an external programming language that can interface

with current solid modeling software, it can potentially be developed into a more universal tool that can help analyze cross sectional drawings within different solid modeling software packages, facilitating the geometry characterization and design analysis and optimization process.

1.4 Goals

1.4.1 Overall Goals of This Work

Our main goals are to understand how current solid modeling software such as UGS-NX5 (UGS Corp., 2007; Tickoo et al, 2007; Carlson, 2003) handle the dimensioning and geometric constraint characterization process as well as removal of unneeded objects or features in bidimensional geometry and apply previous research and current knowledge to develop an integrated software tool that interfaces with the UGS-NX5 environment to analyze the geometry and make the characterization process more user friendly.

Microsoft Visual C++ software (Ullman et al, 2006; Liberty et al, 2005) will be used as the programming language and UGS-NX5 solid modeling software will be used to create the bidimensional part views and as a working environment for our software tool for the removal of unneeded objects, dimensioning and constraint application. These software packages were chosen due to UGS-NX5's open architecture that allows external programs to be written for it, and the availability of specific programming libraries and functions that allow C++ to be used to write code that interfaces with UGS-NX5.

1.4.2 Intellectual Merit of This Work

Our proposed application will advance the field of computer aided design and modeling by being able to interact with existing solid modeling software and enhance its capabilities to analyze bidimensional cross sections. The designer will be able to view any bidimensional drawing, remove unwanted objects and apply characteristic dimensions and constraints to the geometry, improving the characterization and analysis process and thus being able to complete the work faster.

Prior research in the area provides a good foundation for developers to further improve the capabilities of this software tool. Automating the application of dimensions, providing the user with different options for the dimensioning schemes and optimizing constraint management are some areas that were outside the scope of this project but can be developed by future researchers.

1.4.3 Broader Impacts of This Work

Due to the external architecture of the proposed software tool it will be able to interface with a Solid Modeling software package and assist the designer in the geometry characterization process. For the purpose of this project the goal was to achieve the interaction between the software tool and UGS-NX5 but with further development it is possible to enable a broader application, interfacing with different Solid Modeling software packages such as AutoCAD or CATIA. The potential exists for this software tool to become "universal" in its application and being able to interact with many different software packages. This will make this work an attractive proposition for the industry.

1.5 Approach

This project will be focused on applying previous research and currently available solid modeling and programming technology to develop a software tool to help improve the bidimensional cross-section analysis and characterization process. By analyzing the cross-sectional view and determining unneeded features and choosing the objects desired to have dimensions and geometric constraints applied, computer based design and modeling will become more efficient and productive.

The project will pursue the development of a software tool for the removal of unneeded objects, dimensioning and application of constraints for bidimensional cross-section drawings within solid modeling software. Microsoft Visual C++ software (Ullman et al, 2006; Liberty et al, 2005) will be used as computing environment and programming language with applicable functions and programming libraries in order to interface with UGS-NX5 (UGS Corp., 2007; Tickoo et al, 2007; Carlson, 2003) solid modeling software and analyze a bidimensional cross-section drawing of interest. The desired output within the UGS-NX5 environment will be a bidimensional drawing free of unneeded objects and with the user selected objects dimensioned and constrained that will help the designer with the product development process.

Figure 1.8 presents a problem approach flow chart. The bidimensional drawing of interest will be loaded into the UGS-NX5 environment where the C++ software tool will be launched. By analyzing and comparing the coordinates of the points that describe geometry and using position tolerances we will be able to determine if the objects in the drawing are part of an enclosed area, which describes a sectional view. This will allow the software tool

to segregate non-descriptive objects and remove them from the drawing and presenting a clean bidimensional section for the user to analyze. After the user selects the desired parts to be dimensioned and constrained, the software tool will apply the required functions to accomplish the dimensioning and constraint definition and in the end will display the result.

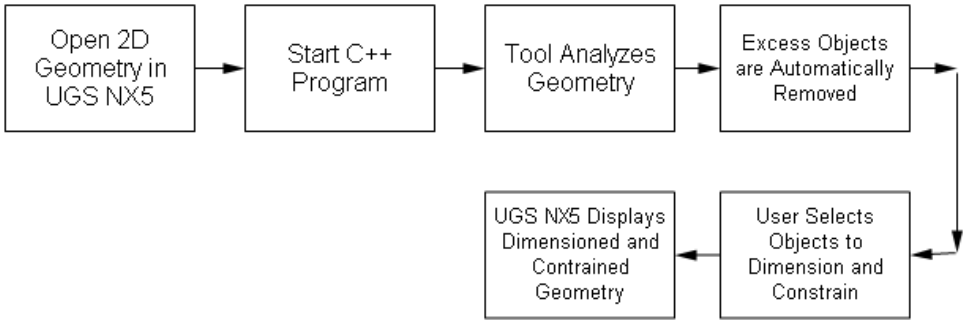

Figure 1.8: Problem Approach Flow Chart

Chapter 2: OVERVIEW OF GEOMETRIC MODELING

To characterize a design within Solid Modeling software packages the designer will have to use several tools including parametric and constraint-based modeling and a proper dimensioning scheme for the accurate description of said design. We propose a software tool that will enhance the characterization process of bidimensional geometry by providing an easier method of removal of unneeded geometry and application of constraints and dimensions. In the following sections we will present different aspects of the creation and tools for characterization of a design.

2.1 Solid Modeling

Solid modeling software creates a virtual representation of components for machine design, analysis and manufacture. A solid model generally consists of a group of features, added one at a time, that define the complete model. The model can be viewed as a three-dimensional solid or as bidimensional representations from different perspectives, as shown in Figure 2.1. For the purpose of this work, we will focus on the bidimensional representations either from different perspective views or by taking planar sections (or slices) from within the body of a solid model. There are several different Solid Modeling software packages currently available to designers to support the product development process and they all provide different tools for the characterization of designs. In this work, we are using UGS-NX5, shown in Figure 2.2, because it provides an open architecture that allows us to use an external C++ code to interface with it. Figure 2.2 shows the basic UGS-NX5 interface

which consists of the main graphical display, the utility and modeling menus, and the parts navigator. The main graphical display is where the geometry will be displayed and manipulated by the user. The *utility menu* contains general use commands like opening new projects, saving current drawings, etc. The *modeling menu* displays the most frequently used drawing and modeling tools and provides access to more available modeling commands. In the *parts navigator* the modeling process is tracked by displaying a history of used commands, provides viewpoint tools, access to parameter lists and also a history of previously opened drawings.

The interaction between UGS-NX5 and C++ is shown in Figure 2.3; after opening a bidimensional drawing into the UGS-NX5 environment, the user needs to press *CTRL+u* to open the external code. The user will select the C++ code from its location in the computer's hard drive and the program will run automatically within the UGS-NX5 environment. First, the user selects all the geometry in the drawing and hits *ok*, the program will remove the objects that are not part of the intended bidimensional geometry. After the geometry is cleaned up, the individual line select dialog boxes will be displayed. Here the user selects the horizontal, vertical, diagonal, or arc lines that the user is interested in having dimensioned and constrained. Our program will then display the dimensions and create a geometric and dimensional constraint of the selected objects.

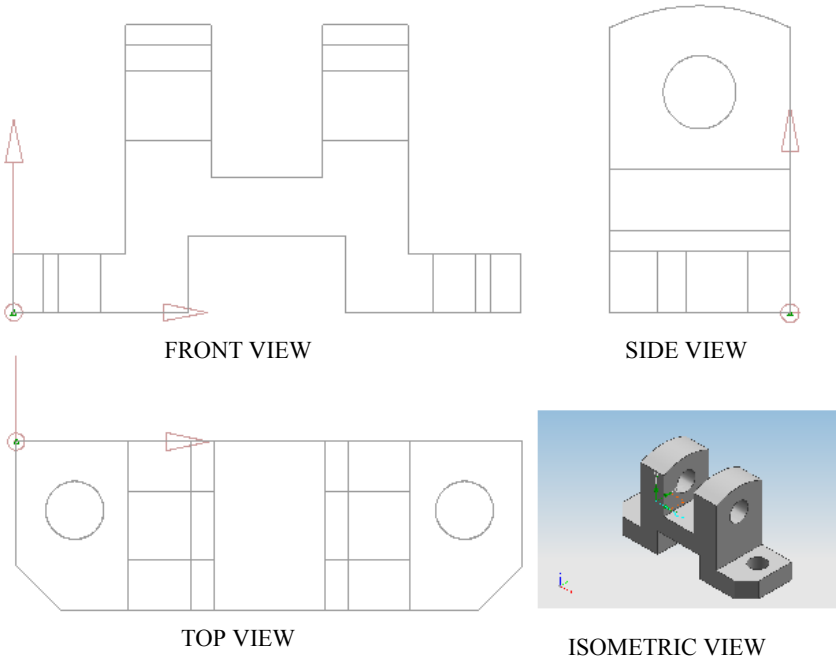

Figure 2.1: Bidimensional Perspectives of a Three-dimensional Model

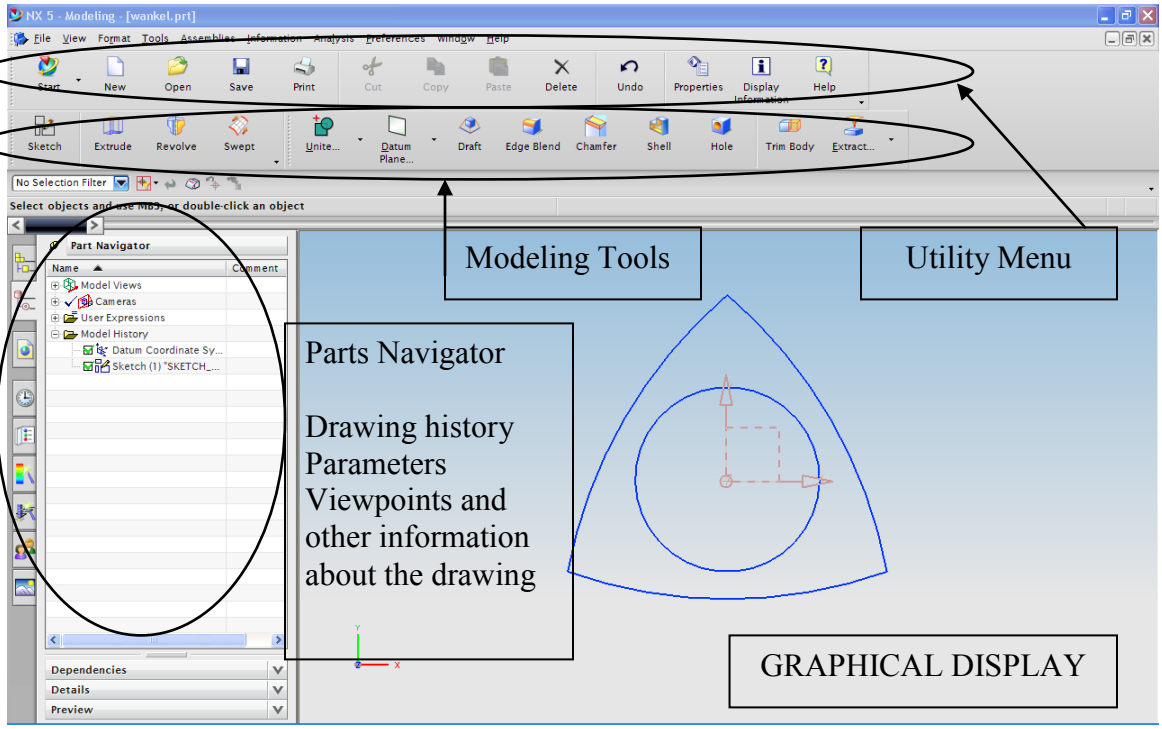

Figure 2.2: Example UGS-NX5 Interface

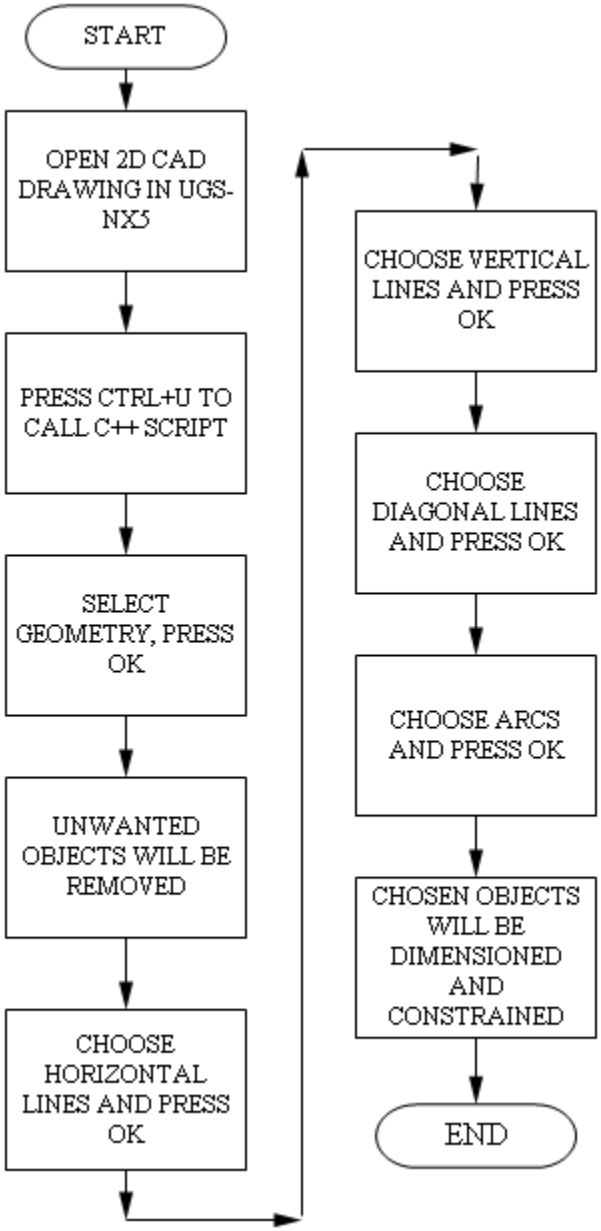

Figure 2.3: General UGS-NX5 and C++ Code Interaction

2.2 Parametric Modeling

Models defined by a set of parameters are defined as *parametric*. The parameters may be modified later, and the model will update to reflect the modifications. Typically, there is a relationship between parts, assemblies, and drawings. Example parameters can be parallelism, perpendicularity, dimensional relationships and others. A part consists of multiple features, and an assembly consists of multiple parts. Drawings can be made from either parts or assemblies. The airfoil in Figure 2.4 is described by the set of parameters shown in Figure 2.5; there are also dimensional constraints in the model characterization, these are discussed in the next section.

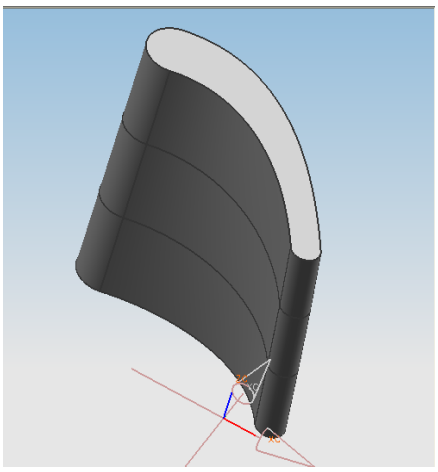

Figure 2.4: Parametric Airfoil Geometry

Name	Formula	Value
angle_attack	20	20
angle_attack_used	if(angle_attack>30)(30)else(if(angle_attack<0)(0)else(angle_attack))	20
chord_length	1.5	1.5
chord_length_used	if(chord_length>3)(3)else(if(chord_length<1)(1)else(chord_length))	1.5
height	2	2
height_used	if(height>3)(3)else(if(height<1)(1)else(height))	2
Leading_Edge_Angle	15	15
Leading_Edge_Angle_...	if(Leading_Edge_Angle>25)(25)else(if(Leading_Edge_Angle<0)(0)else(Leading_Edge_Angle))	15
Leading_Edge_Radius	.2	0.2
Leading_Edge_Radius_...	if(Leading_Edge_Radius>.25)(.25)else(if(Leading_Edge_Radius<.1)(.1)else(Leading_Edge_Radius))	0.2
Trailing_Edge_Angle	60	60
Trailing_Edge_Angle_u...	if(Trailing_Edge_Angle>75)(75)else(if(Trailing_Edge_Angle<45)(45)else(Trailing_Edge_Angle))	60
twist_angle	15	15
twist_angle_used	if(twist_angle>20)(20)else(if(twist_angle<0)(0)else(twist_angle))	15

Figure 2.5: Parameters and Constraints that Characterize the Airfoil in Fig. 2.4

Parametric design determines the representation of the graphical elements by the application of geometric constraints. A set of geometric constraints is defined by the parametric system to represent the design. Changing the relationship among parameters often requires modification of the parametric model. Parametric design specification is one of the possible alternatives available within a constraint-based design representation (Dones, 1991). Making a parametric model helps in the characterization of the design by describing it to the user within the Solid Modeling software and allowing modification of the model by modifying the parameters. This work will enable an easier application of constraints and dimensions which can help the user to parameterize the design.

2.3 Constraint-Based Geometry and Modeling

Related to parameters, but slightly different are *constraints*. Constraints are relationships between entities that make up a particular shape. For a window, the sides might be defined as being parallel, and of the same length.

Constraint-based geometry is a technique by which an arbitrary geometry is defined or constrained using its dimensions. In constraint-based geometry an object is considered to be determined by a number of characteristic points in a three-dimensional space. Instead of using the geometry to define the dimensions as in parametric modeling, constraint-based modeling uses the dimensions to define its geometry. The dimensions are used to relate geometrical elements such as points, lines, arcs or circles. Figure 2.6 shows examples of triangular geometry constrained by its dimensions and in Figure 2.7 we see the set of constraints defining the model in Figure 2.1.

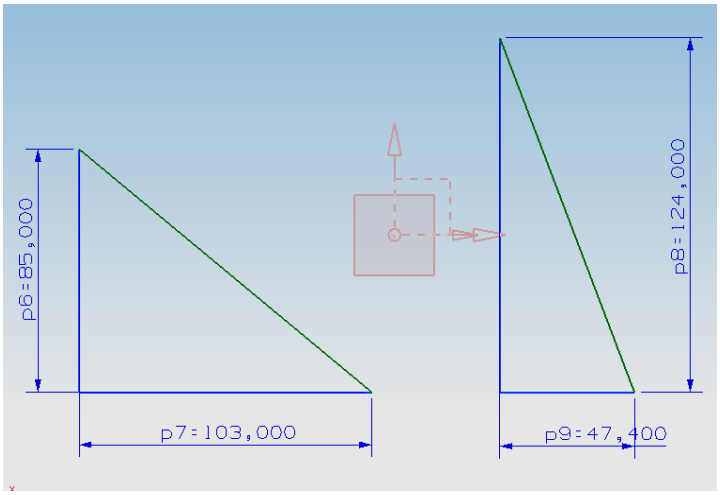

Figure 2.6: Example of a Triangular Geometry Constrained by its Dimensions

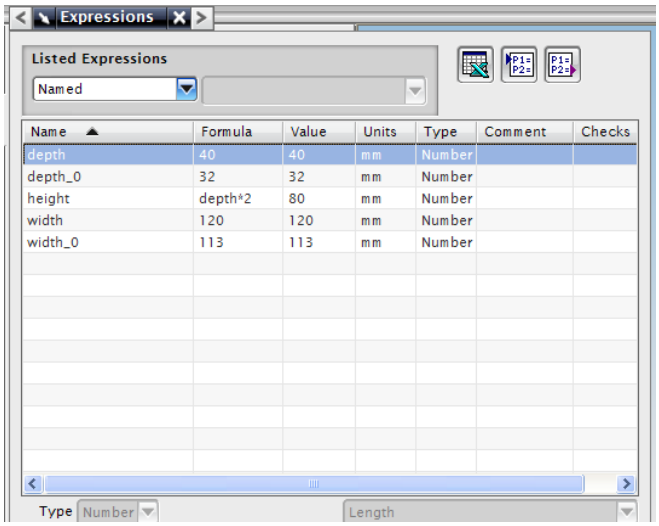

Figure 2.7: Dimensional Constraints that Describe the Solid Model in Fig 2.1

Dimensioning and constraint-based geometry help the designer in the definition of the object and the use of geometrical dimensions as part of design specifications. There is a direct relation between the constraints and the dimensions of the object. Once geometry is defined by a set of constraints, these may be manipulated to obtain valid dimensioning schemes. An example of constraint application is setting a desired dimension in the geometry as fixed; this will make it possible to change other parts of the geometry while retaining the desired fixed dimension. This is another tool used in the design characterization process and a part of the software tool developed for this work as it provides a simple selection method for application of constraints and complements other available options for the description of a design.

2.4 Dimensioning

We use dimensioning as a tool to help the designer understand the spatial arrangement of the objects being drawn. When the geometry is dimensioned, parameters of the explicit constraints are displayed in the drawing. These include horizontal, vertical, parallel dimensions, radius, diameters, angles and other engineering dimensions defined by the designer, as shown in Figure 2.8.

Dimensions for size and position must be complete so that the part is defined for manufacturing. There is more than one way to dimension single parts thus the dimensioning process must be performed individually for each component in a design (Dones, 1991).

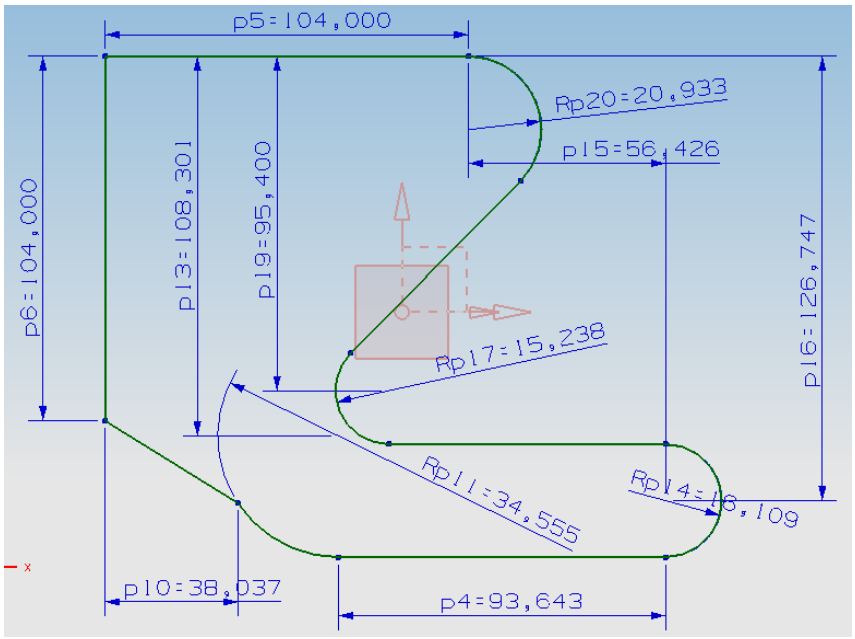

Figure 2.8: Example Dimensioning Scheme

A consistent set of constraints should be supplied and requires that every characteristic point in the geometry be uniquely constrained. Dimensions are used to constrain the object's geometry. Dimensioning schemes are not unique and it may be possible to define invalid schemes. For an effective use of a constraint-based system a valid set of constraints must be defined (Pérez, 1993). Dimensions and dimensional constraints are necessary in the design process and are integrated into the geometry description and applied in this work through the use of our software tool.

Chapter 3: APPLICATIONS

In this chapter, we will discuss the development of the C++ software tool and its application within the UGS-NX5 environment. UGS-NX5 is an open architecture CAD program that allows the application of an external code. For the purpose of this work, we developed the external code using C++. With the use of available libraries and functions targeted specifically for the interface between C++ and UGS-NX5, we were able to achieve the required interactions between the two software applications. The result of this development was a software tool capable of analyzing and aiding in the characterization of bidimensional geometry within the UGS-NX5 modeling environment.

3.1 C++ as the Programming Language

The C++ programming language provides a combination of both high-level and low-level language features. It is widely used in the software industry; its standard was ratified in 1998, the current version of which is the 2003 version (ISO/IEC JTC1/SC22/WG21 - The C++ Standards Committee, 2008). Many C++ libraries exist which are not part of the standard, including *uf* libraries specifically designed to work with Unigraphics Software, now called UGS-NX5.

For the purpose of this work, we used the *uf* libraries, shown in Table A.1 in Appendix A. By applying these libraries within the C++ code, as presented in Appendix B, we enabled the interaction between our software tool and UGS-NX5. Figure 3.1 provides an

overview of the main program. The program starts by initializing the "Open C API Environment" a command that enables the UGS-NX5 capability of being run by an external C++ code. Then it runs the six specific functions in the program, which will be explained in the next section. The functions read the geometry's characteristic points in the plane and with that information they calculate which objects are parts of the desired geometry and which ones need to be removed. The functions will also create dimensioning and constraining information for the remaining objects that are selected by the user. The output of this algorithm will be a geometry free of unwanted objects and with the desired parts dimensioned and constrained. This is the major contribution of this work.

Figure 3.1: Main Program Flow Chart

3.2 UGS-NX5 Solid Modeling Package

UGS-NX5, originally called Unigraphics, is a commercial Computer Aided Design software suite developed by Siemens PLM Software. UGS-NX5 serves the basic design tasks by providing different environments. An environment is defined as a specified environment, consisting of a set of tools, which allow us to perform specific design tasks in a particular area. Figure 3.2 shows an example of the Modeling Environment.

The Modeling environment that we are using for this work is a parametric and feature-based environment. The parametric nature of the software package allows us to use parameters in defining the shape and size of a geometry.

Here, we used the geometric and dimensional constraints available within UGS-NX5. The geometric and dimensional constraints in the Sketcher environment are used to precisely define the size and position of the sketched elements with respect to the surroundings. UGS-NX5 also provides us with various types of dimensions such as: Horizontal, Vertical, and Radius which are also used with our C++ software tool.

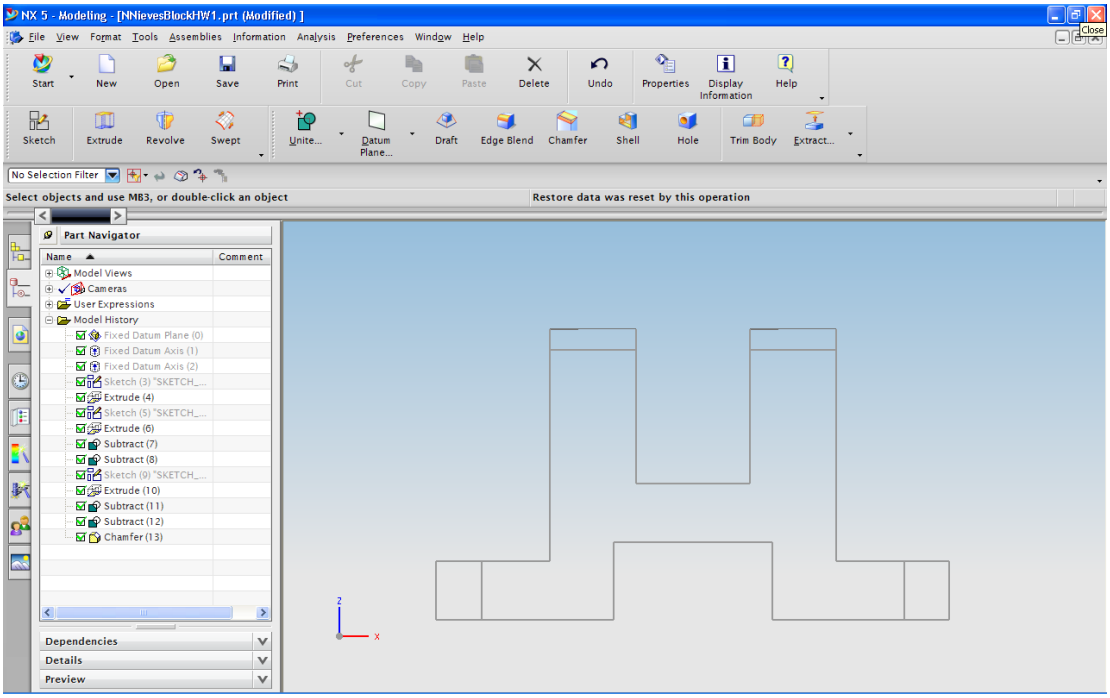

Figure 3.2: Example UGS-NX5 Modeling Environment

3.2 Application Description

UGS-NX5 has an "open architecture" design that allows us to externally access the program functionality. We achieved C++ interaction with UGS-NX5 using the available C++ *uf* libraries and the set of functions specific to UGS-NX5 shown Table A.2. With this functionality we developed a UGS-NX5 external C++ program that analyzes the objects that form the bidimensional geometry within the solid modeler's environment.

The code was divided into several functions called from the main program. It prompts the user to select the geometry to be analyzed and then records the coordinates of the points that make up the geometry. Using this information the tool determines which objects are not part of the geometry and removes them. The tool will then prompt the user to select

the lines or arcs to be dimensioned and constrained and displays the dimensions on the drawing. The third step shown in the flowchart in Figure 3.1 can be expanded as shown in Figure 3.3. It consists of six specific functions that we created for this work and will be explained in the following paragraphs.

Function 1, *layerSelectable*, shown in Figure 3.6, enables the 256 available work layers within NX to be user-selectable and makes *layerZ* the active work layer. It starts by assigning the arbitrary values 250 and 251 to the variables *layerY and layerZ*, respectively. It then runs a *for* loop with a counter variable *i* from 1 to 256 to enable each of the 256 work layers within UGS-NX5 to be selectable. After the loop is complete, it will set *layerZ* to be the work layer.

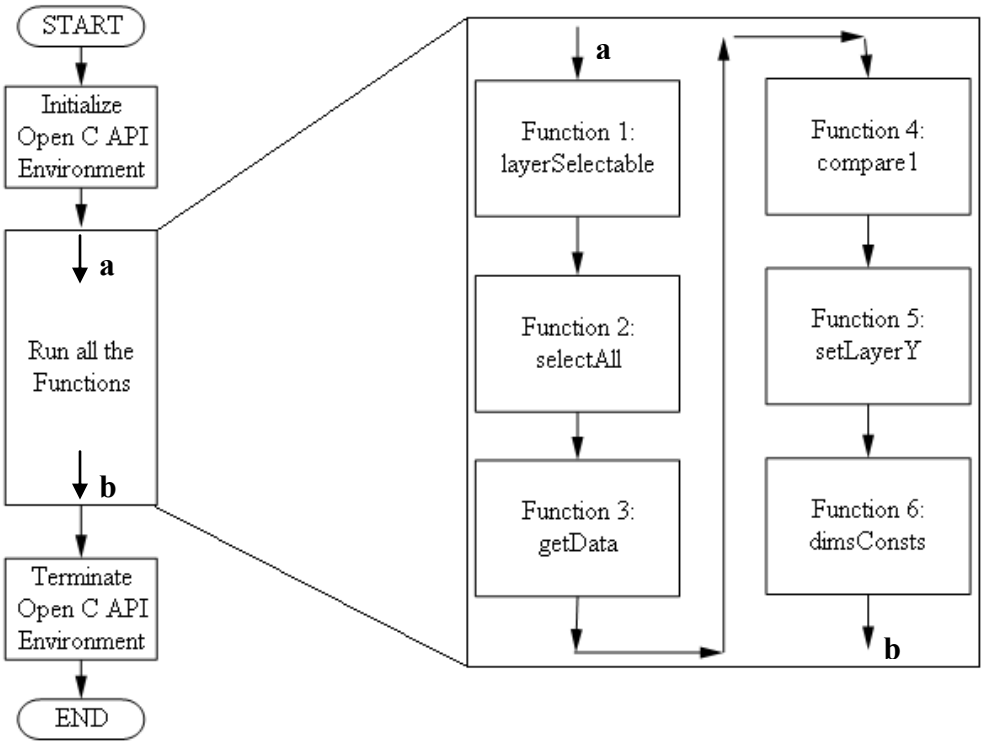

Figure 3.3: Functions within our C++ Code

Function 2, *selectAll*, shown in Figure 3.7, displays the object selection dialog, highlights the objects in the geometry and records the ID of objects into an array. It then sets the objects to *layerZ* and makes it and *layerY* the only active work layers. It first displays the object selection dialog shown in Figure 3.4 and awaits the user input to be able to define the variables *response* and *count*. When the user presses *Select All* in the dialog option the ID of the objects in the geometry will be recorded and the variable *count* will contain the total number of objects. Pressing *OK,* along with a positive count of objects, will satisfy the *if-else* statement and continue on. The function will then run a *for loop* with counter variable *i* from 0 to the value in the variable *count.* There are two commands in the loop, the first one highlights each object and the second one sets the object to *layerZ*. At the end of the *for loop* all the objects will be highlighted and set in *layerZ*. The next routine is a *for loop* and an *if-else* statement. The loop will run with counter variable *i* from 1 to 256. Inside the loop is the *if-else* statement which is satisfied for every value of *i* different from *layerY or layerZ.* With the statement satisfied, a command is run that disables the layer or sets it to be inactive. This effectively makes *layerY* and *layerZ* the only active work layers.

Function 3, *getData*, in Figure 3.8, records the curve properties of the objects in the array. These properties are point, tangent, unit principal normal, unit binormal, torsion, and radius of curvature. It also records the x, y and z coordinates of the start and end points of the lines and arcs that form the geometry. It runs a *for loop* with counter variable *i* from 1 to the value in the variable *count* and will read the information from every single object recorded. The commands in the loop are one for retrieving the curve properties at the start

and end points of the object, and the second one that will retrieve the x, y and z coordinates of the start and end points and set these values to their corresponding variables.

Function 4, *compare1*, in Figure 3.9, compares the start and end points of all lines and arcs to determine whether they are part of the enclosed area that defines the cross section of interest or excess lines that have to be removed. It then separates the good lines from the excess lines and sets them into different work layers, excess lines to *LayerZ* and good lines to *LayerY*. It uses a *for loop* with counter variable *i* from 1 to the value in variable *count.* The first action within the loop is saving all the data from previous functions into local variables. It then goes into another *for loop* that uses counter variable *j* from 0 to the value in the variable *count.* For every iteration of the loop it will retrieve the ID of each object and compare counters *i* and *j* with an *if-else* statement. The statement will be satisfied as long as the counter variables are different from each other. This will start the analysis of every individual object to determine if they are part of the cross section of interest.

The analysis consists of a comparison of the coordinates of the start points of every object in each of the x, y, and z axes against an internally programmed tolerance. The lines that are found to have matching start point coordinates will be sent to *layerY*. After this, the comparison starts again but this time for each of the end points of each object in the geometry in every x, y, and z axes. The lines found to have matching end point coordinates will be sent to *layerY*. This completes the function which purpose is to distribute the good lines and the excess lines into different work layers. Excess lines will be set to *LayerZ* and good lines to *LayerY*. Currently, the tolerance used for coordinate comparison is internally

programmed within the algorithm; a future improvement may be an option for the user to set a different tolerance.

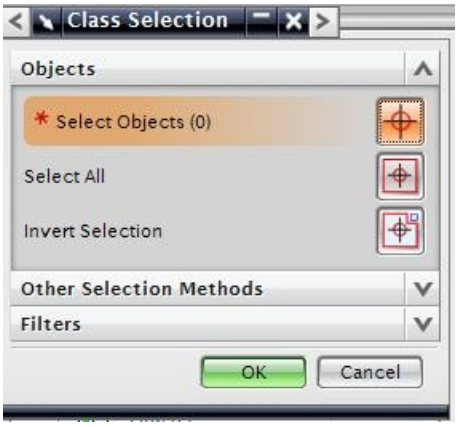

Figure 3.4: Object Selection Dialog as Displayed from Function 2: *selecAll*

The next flowchart, shown in Figure 3.10, corresponds to Function 5, *setLayerY*. It will set *LayerY,* which contains the good lines previously calculated, as the only active and shown work layer effectively removing excess objects from the display. It completes the task by using a *for loop* with counter variable *i* ranging from 1 to 256 it will run an *if-else* statement which is satisfied for every value of *i* different from *layerY*. With each iteration one layer will be disabled or set to be inactive. When counter variable *i* equals the value of *layerY* the *if-else* statement is no longer satisfied and the function goes to the next command which sets *layerY* as the only active layer. Since *LayerZ* contains the excess lines calculated from the previous function, they will be effectively removed from the geometry and no longer shown on the display.

The last function, Function 6, *dimsConsts,* shown in Figure 3.11 and 3.12, displays the line selection dialog, records the curve properties and highlights the lines or arcs selected by the user and also saves the absolute spatial coordinates of the start and end points. With this information it creates dimensional information to be displayed on the screen and also creates dimensional and geometric constraints and applies them to the selected objects. It first displays the line selection dialog once each for *Horizontal, Vertical, Diagonal Lines* and *Arcs* as shown in Figure 3.5 and awaits user input to continue. It will highlight every individual object selected by the user using a *while* statement that is satisfied as long as the variable *response* is different than *OK*. It will follow with an *if-else* statement that verifies the value of variable *response* to be different than *OK* and if satisfied will run the commands for saving the curve properties for the start and end points of the object as well as the points' coordinates in the x, y and z axes. Following this the function runs a command that creates

the dimension and dimensional constraint of the object and it will also create a geometric constraint for it and save it into variables.

When the user selects *OK* in the line selection dialog box the function will not satisfy the *while* statement and thus will continue to the next routine. The routines all follow the same syntaxis as the first one for horizontal lines with difference being that it will run once for each type of line. The subroutine is repeated for horizontal lines, vertical lines, diagonal lines, and arcs. For the arc subroutine the dimensions and constraints are defined as radial, horizontal and vertical; this is the only difference compared to previous subroutines. After completing all the line selection subroutines, the last task is to check the constrained status and degrees of freedom of the geometry.

The resulting display within the UGS-NX5 environment is a geometry free of excess lines and with the user selected objects dimensioned and constrained and a message displayed to the user with information about constrained status, over constrained, fully constrained, or how many constraints are required..

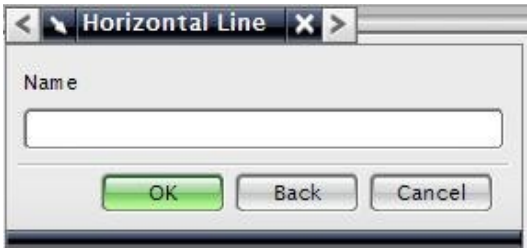

Figure 3.5: Line Selection Dialog for *Horizontal* Lines. *Vertical, Diagonal* and *Arcs* are similar.

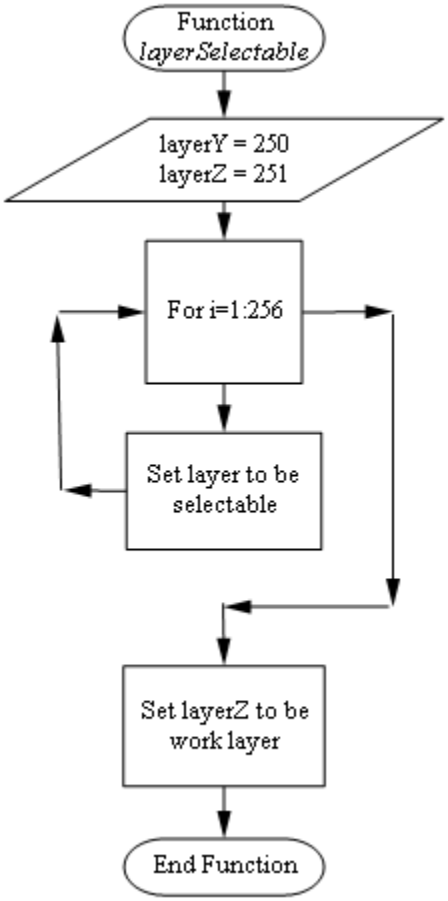

Figure 3.6: Function 1: layerSelectable

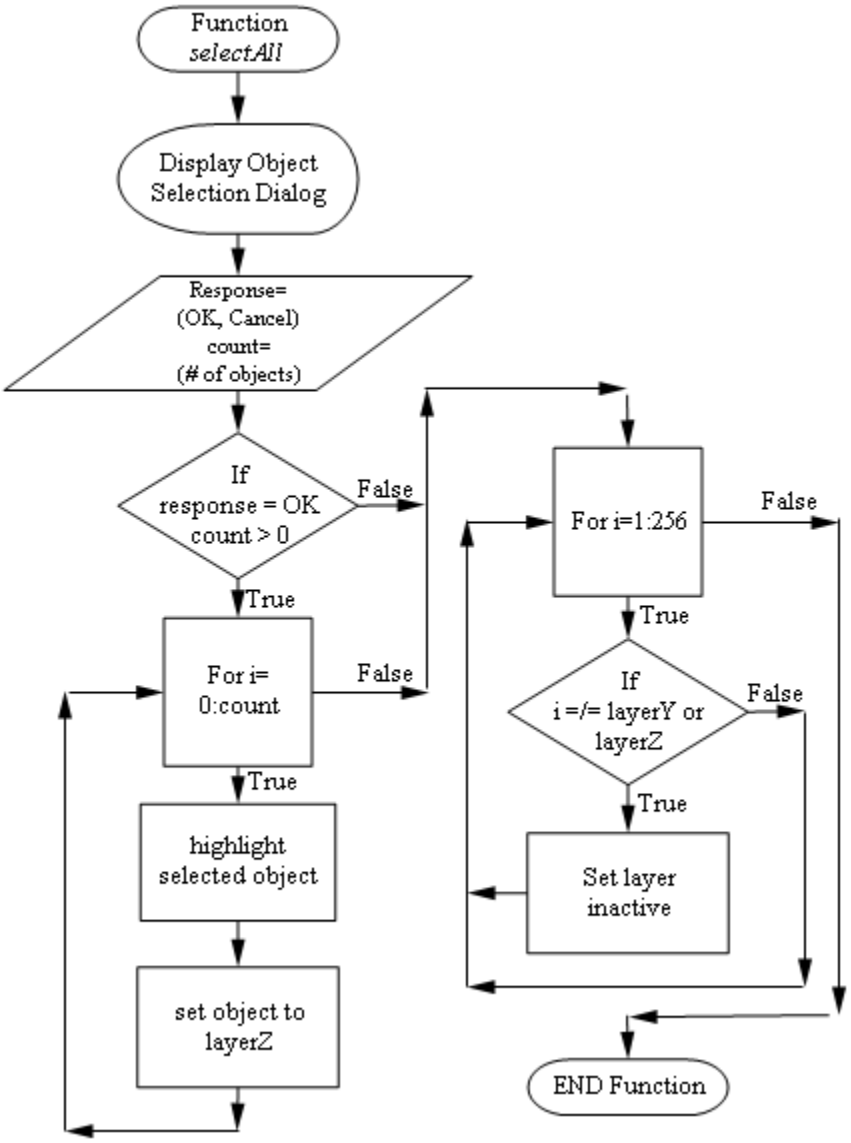

Figure 3.7: Function 2: selectAll

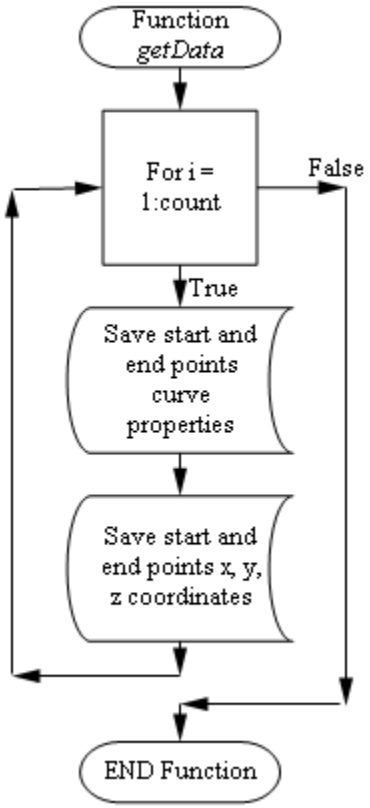

Figure 3.8: Function 3: getData

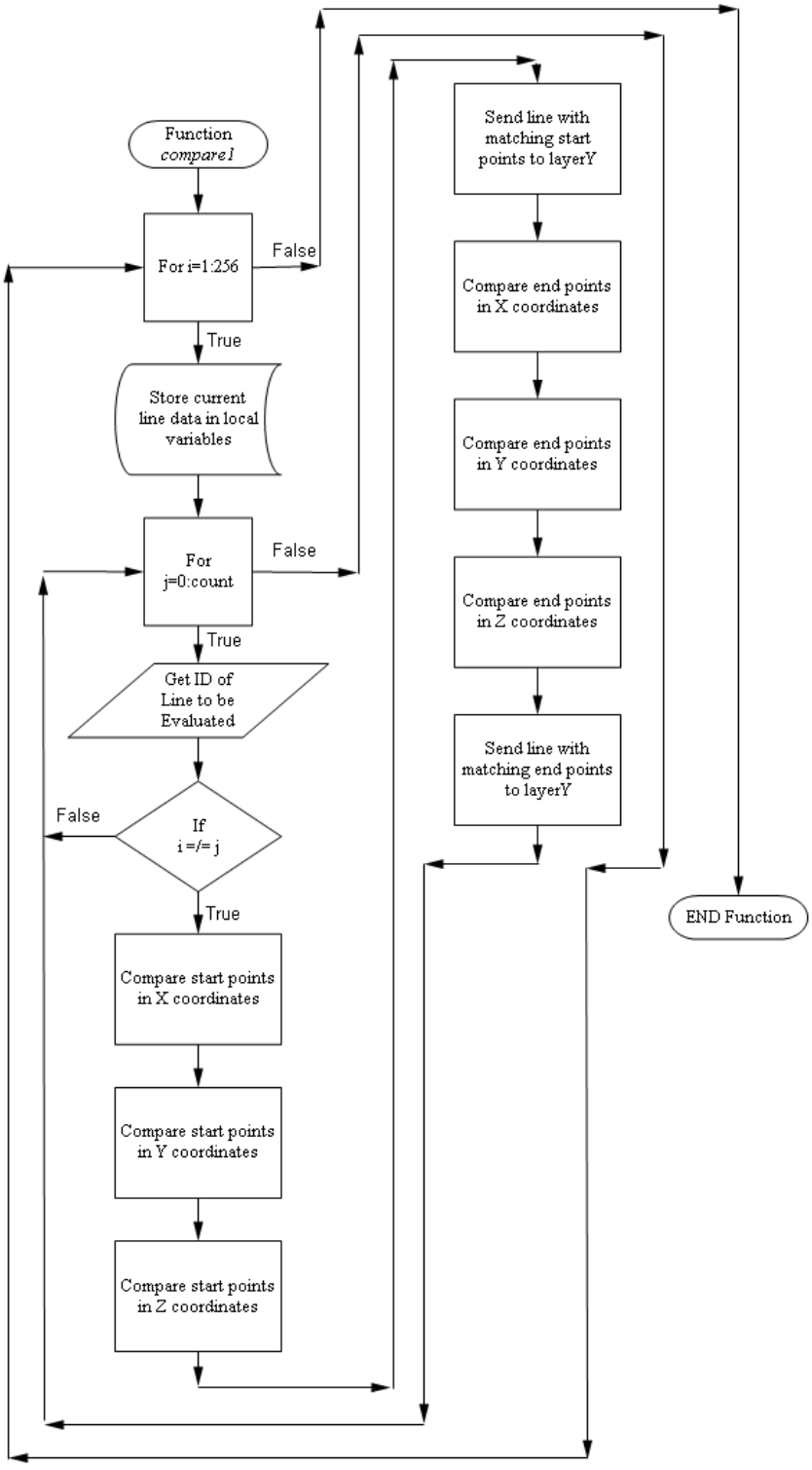

Figure 3.9: Function 4: compare1

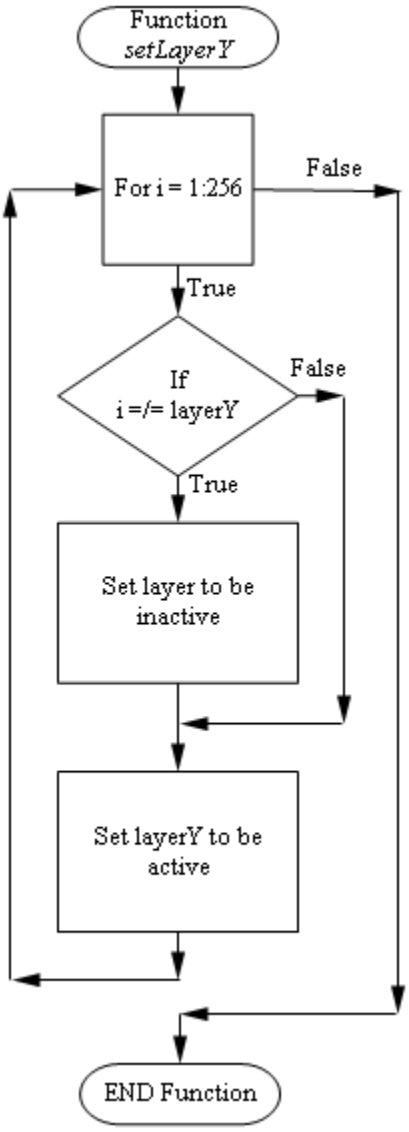

Figure 3.10: Function 5: setLayerY

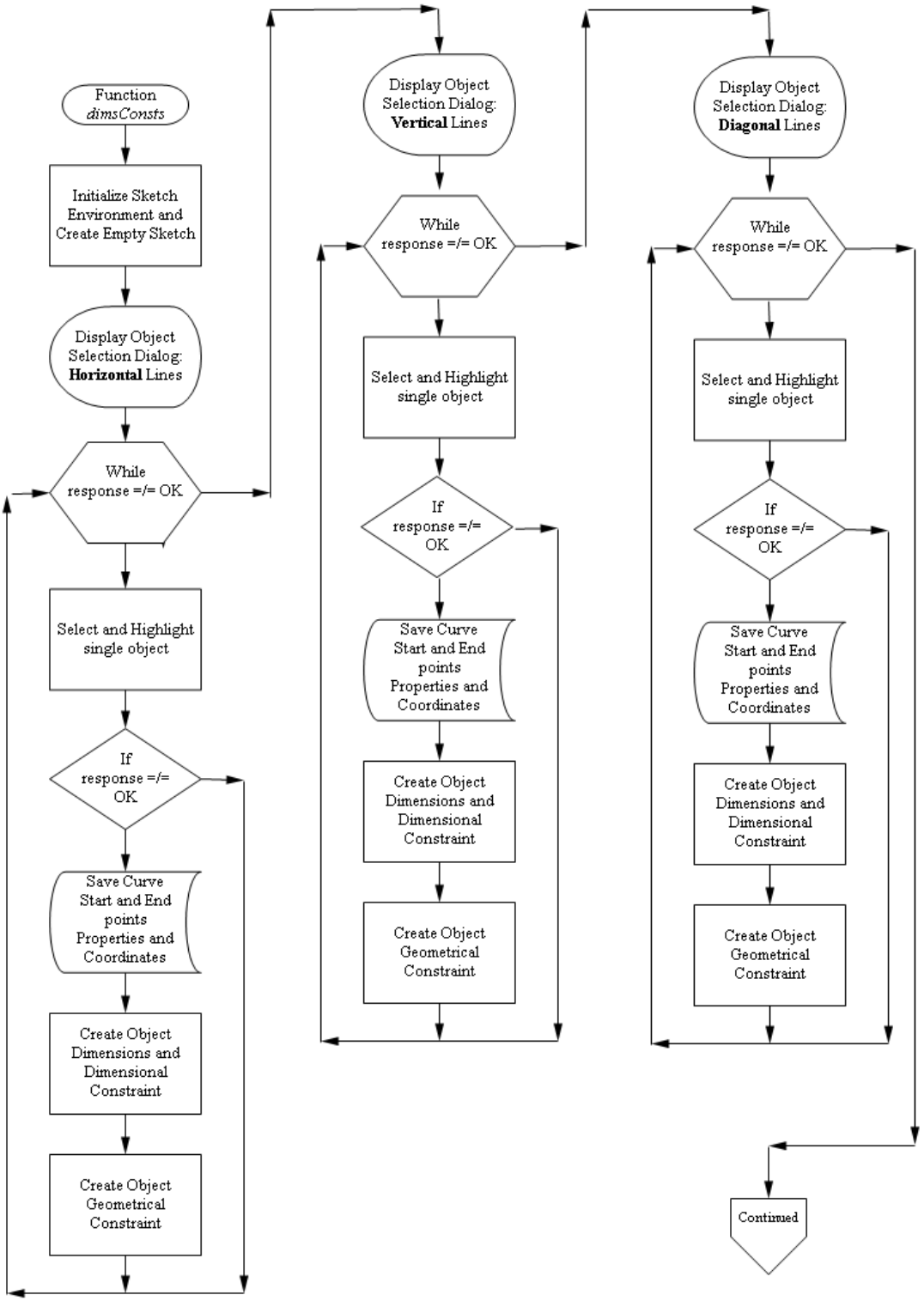

Figure 3.11: Function 6: dimsConsts, continued on next page

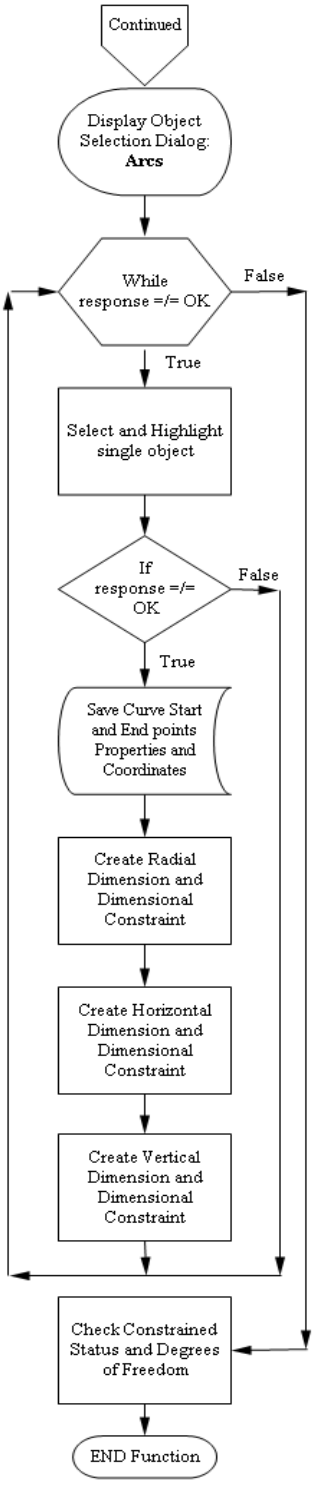

Figure 3.12: Function 6: dimsConsts, continued from previous page

3.3 Results

Now we apply the program to various examples, shown in Figures 3.13 to 3.22, of bidimensional geometries subject to unwanted *items*. We applied our software tool on these geometries and the results show complete removal of unwanted items and the dimensioning and constraining of the objects within the geometry selected by the user. The examples in this section are presented with the original geometry and the final result. A complete step by step walkthrough of Example 1 is presented in Appendix C.

3.3.1 Example 1 –Bidimensional Sketch of the Word "section"

The first example shown in Figures 3.13 and 3.14 consists of an arbitrary bidimensional sketch that contains the word *section* and is covered by a multitude of unnecessary lines and arcs that cover the geometry. This example represents an exaggerated case unlikely to occur in actual application but shown here as a demonstration of the usefulness of the software tool. After running the characterization tool, the intended geometry is displayed and easily recognized and the selected parts of the drawing have been dimensioned and constrained.

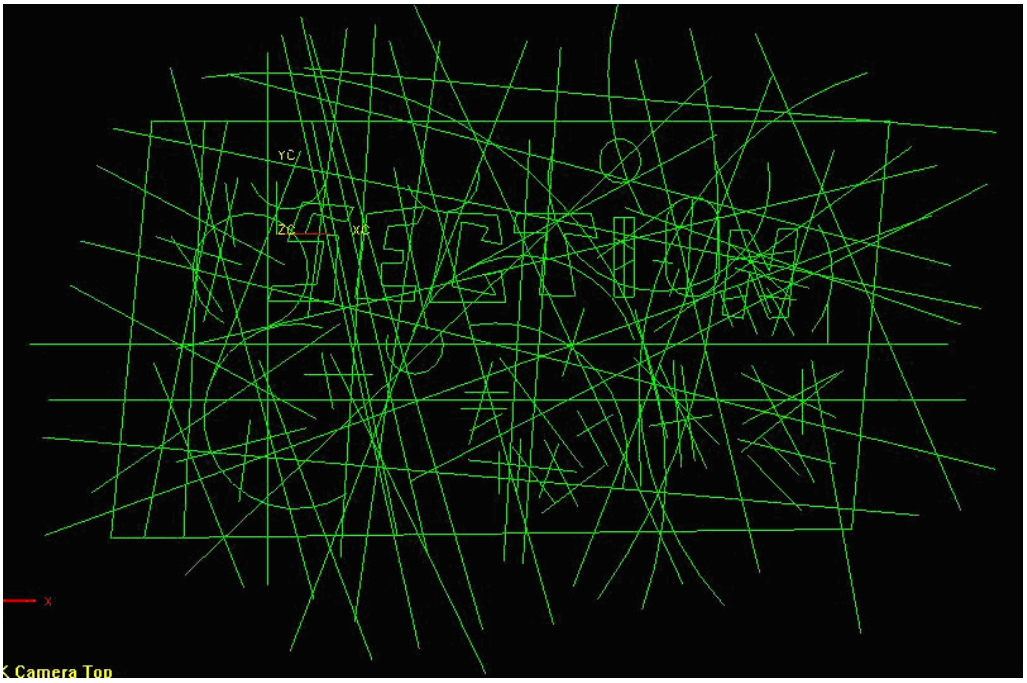

Figure 3.13: Arbitrary Geometry with Unwanted Objects

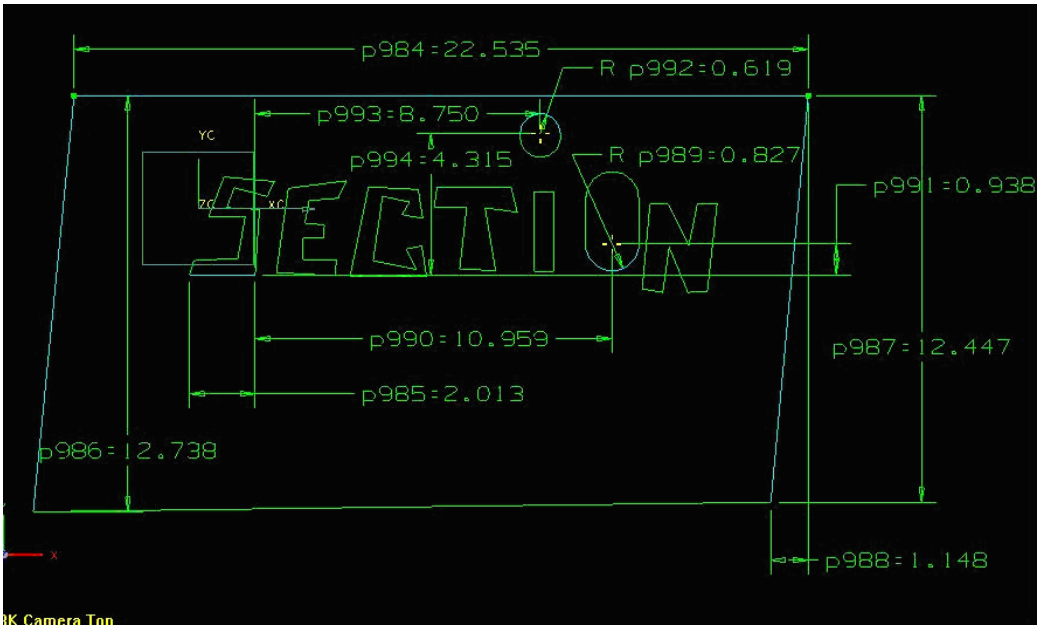

Figure 3.14: Resulting Display After the C++ Software Tool was Applied to Figure 3.13

3.3.2 *Example 2 –Representation of a turbine airfoil*

Figure 3.15 is a representation of a turbine airfoil solid model and its top bidimensional view. Although there are no unwanted objects in this example, the software developed in this work remains useful in the application of dimensions and constraints to the selected parts of a geometry as shown in Figure 3.16.

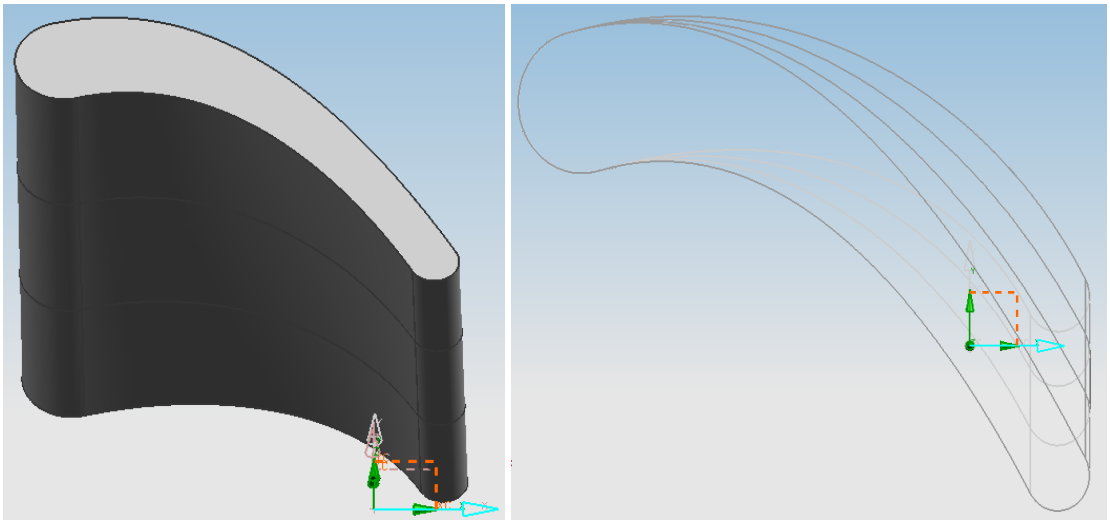

Figure 3.15: Example Turbine Airfoil Solid Model and Bidimensional Top View

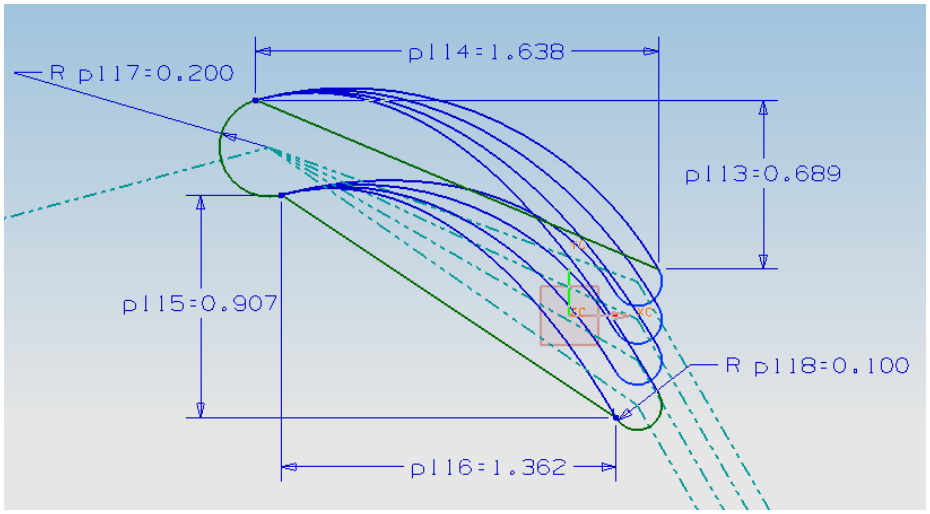

Figure 3.16: Example Turbine Airfoil Dimensioned and Constrained.

3.3.3 Example 3 –Representation of a stamping or machined component

The geometry shown in Figures 3.17 and 3.18 is a sample geometry that could be representative of a metal stamping or machined component. Notice the removal of unwanted objects and the dimensions displayed in the final geometry in Figure 3.18; the dimensioned objects are constrained as well.

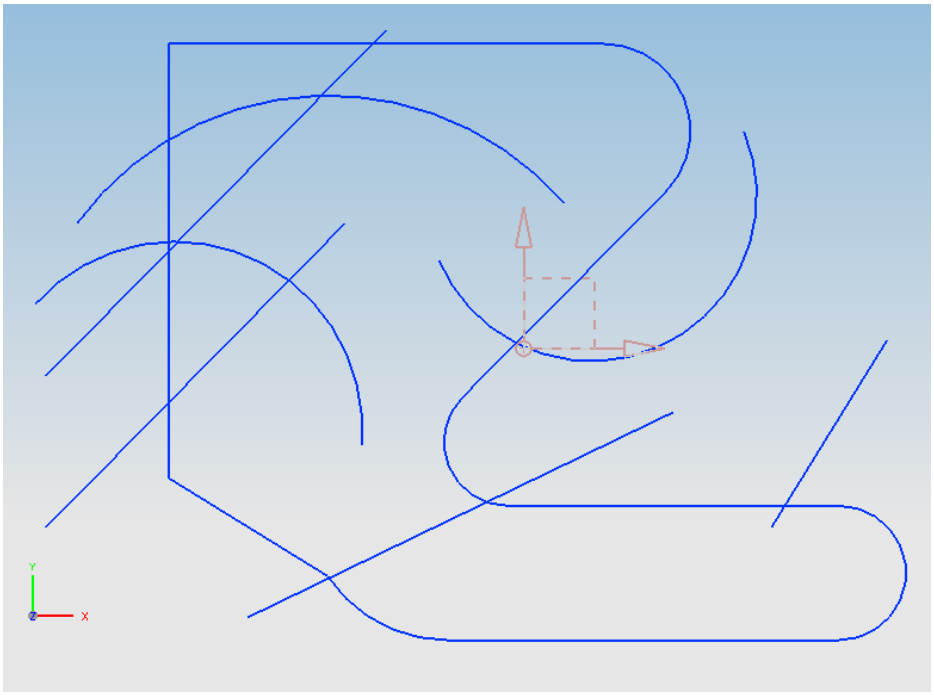

Figure 3.17: Example Geometry with Unwanted Objects

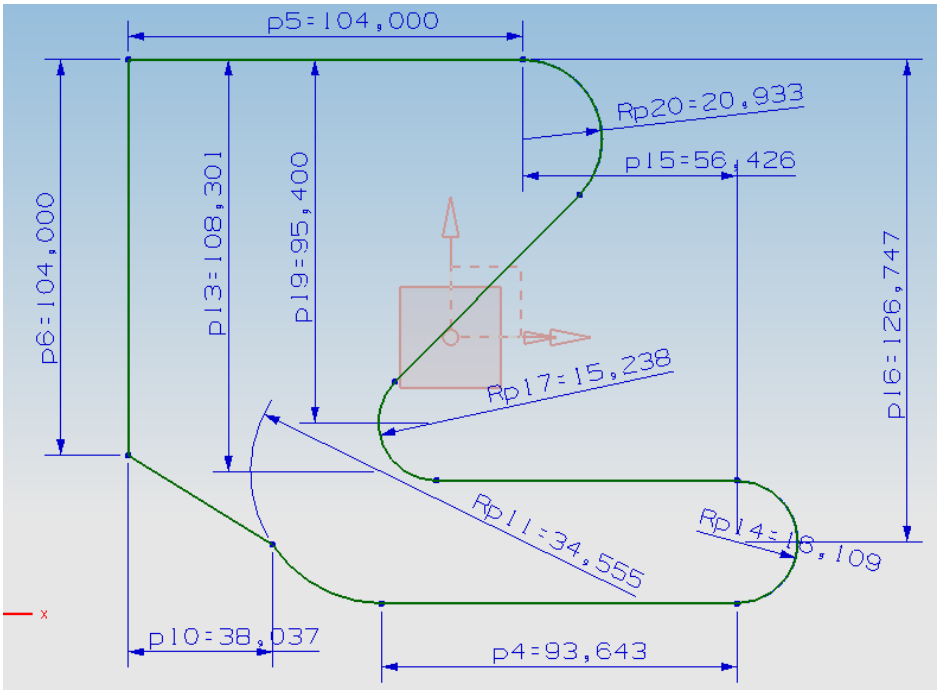

Figure 3.18: Resulting Geometry Dimensioned and Constrained

3.3.4 Example 3 – Wankel-Type Rotor from an automotive engine

In Figures 3.19 and 3.20 we present a side view of a Wankel-type rotor from an automotive internal combustion engine. Figure 3.19 shows the rotor's side view geometry with unwanted objects that disrupt the intended view and make it more challenging to accurately characterize the geometry. Figure 3.20 shows the geometry free of these objects and with the selected parts dimensioned and constrained.

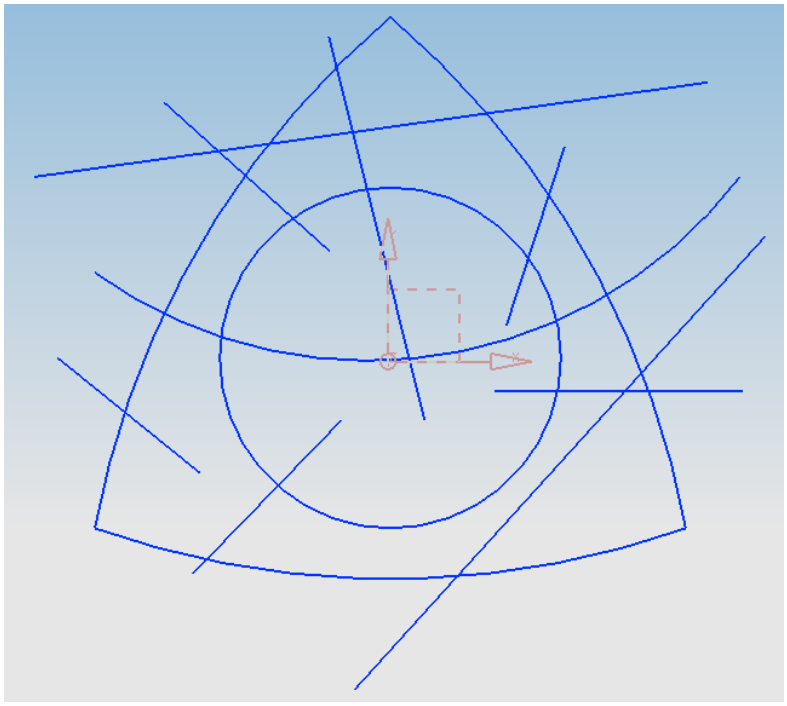

Figure 3.19: Example Wankel Rotor Profile with Unwanted Objects

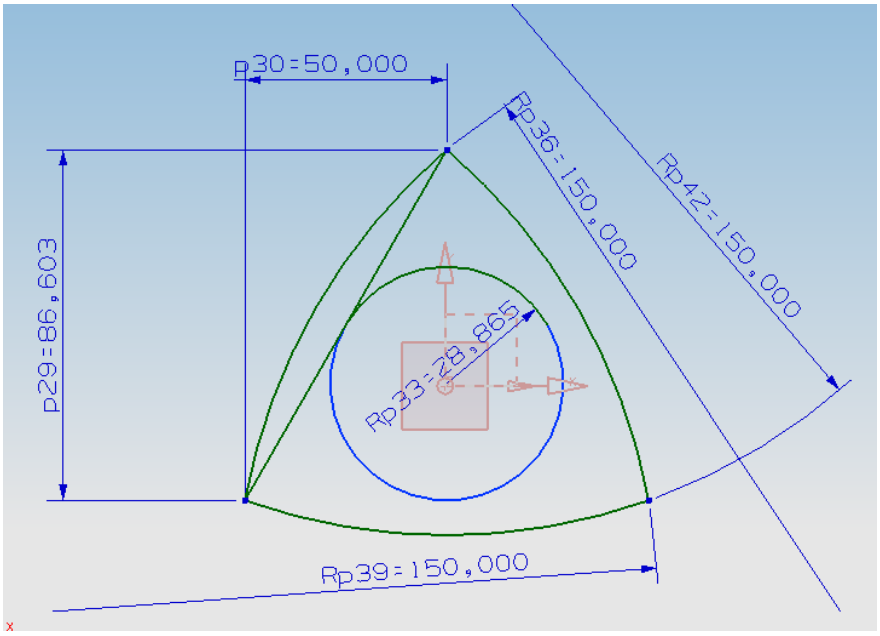

Figure 3.20: Resulting Rotor Profile Dimensioned and Constrained

3.3.5 Example 4 – Piston and rod representation

In Figures 3.21 and 3.22 we present a side view geometry representing a piston and rod from an automotive internal combustion engine. Figure 3.21 shows the geometry with unwanted objects and Figure 3.22 shows the final geometry free of these objects and with the selected parts dimensioned and constrained.

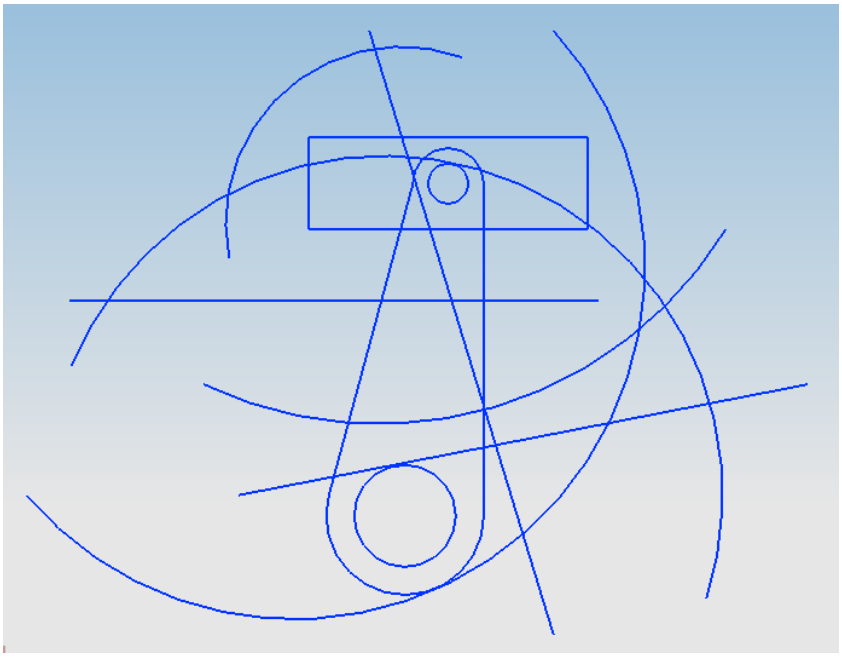

Figure 3.21: Example Piston Assembly Profile with Unwanted Objects

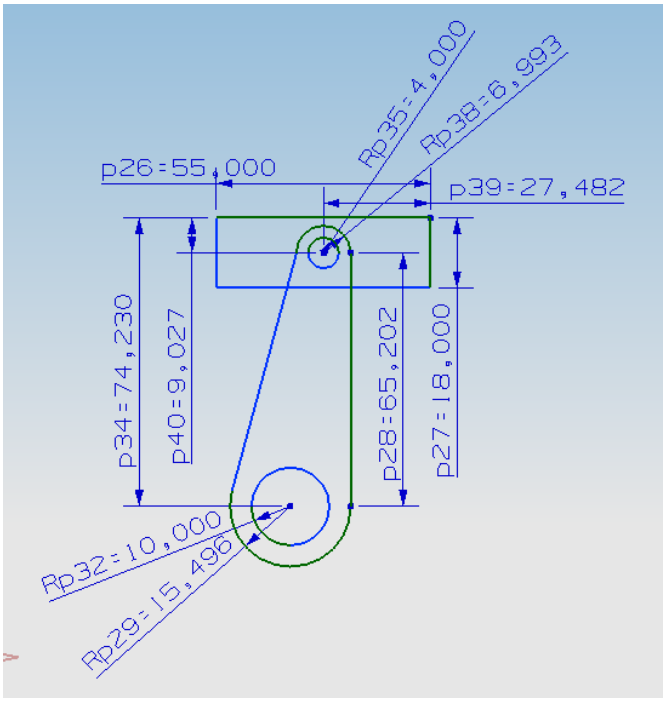

Figure 3.22: Resulting Piston Profile Dimensioned and Constrained

Chapter 4: FINAL REMARKS

4.1 Conclusions

The main objectives of this project were to improve how current modeling software, such as UGS-NX5, handle the dimensioning and constraining processes for bidimensional geometries as well as the removal of unnecessary objects by developing a software tool to help make these processes more intuitive and user friendly.

Using C++ programming language with specific software libraries and functions to interface with UGS-NX5, a software tool was developed to analyze the bidimensional objects within the modeling environment. The tool identifies the coordinates of the points that describe the geometry and determines if there are any unnecessary or remainder lines or other objects from previous manipulation of the drawing and removes them to let the designer work with the intended cross sectional area; then it allows the designer to manually choose and select objects to be dimensioned and geometrically constrained and will display the dimensions and a message for the designer to know if the geometry is under, over, or fully constrained.

The result from using the C++ software tool is a bidimensional geometry within the UGS-NX5 modeling environment that contains no unnecessary or remainder waste objects visible and the user selected objects will be dimensioned and geometrically constrained, making it parameterized. This helps the designer visualize and understand the spatial relationships of the different objects within the geometry.

4.2 Recommendations and Future Work

The main scope of this project was met with the development of a C++ software tool that interfaces with UGS-NX5 and facilitates the removal of unnecessary objects and the application of dimensions and geometric constraints to the selected objects within the geometry. The layout of the dimensions was not considered within the scope of the project and remains as an open issue for future work. The automation of the different processes within the tool, such as dimensioning and constraint application, constraint management and optimization techniques; also remain as open issues for future developments.

Although the resulting output from manual selection is enough for documenting the functionality of this software tool, developing algorithms to automate and optimize the dimensioning and constraining processes and for obtaining a better layout for dimensioning geometry is recommended.

Another limitation of this work is that it was developed specifically to interface only C++ with UGS-NX5 and bidimensional geometries. Although UGS-NX5 is an excellent solid modeling software package, there are many other excellent modeling software packages available from other companies. Broadening the use of the tool to these other solid modeling software families and expanding the analysis capabilities of the tool to three-dimensional objects can also be considered an area of further development. An idealized characterization tool would be able to recognize the interfacing solid modeling software, load the appropriate libraries and seamlessly start the automatic characterization process regardless of the solid modeler in use.

REFERENCES

Aldefield, B., (1988). *Variation of Geometries Based on a Geometric-Reasoning Method*, Computer Aided Design, 20(3), 117-126.

Borduin, S. M., Autodesk, Inc. (2000) *Modeling System Having Constraint Solvers*, U.S. Pat. 6,063,126

Dones Pérez, P. M. (1991) *Automatic Dimensioning in Constraint-based Geometry*, Thesis (M.S.), University of Puerto Rico Mayagüez Campus.

Goyal, V. K., R. Valentin, J. F. Betts, and V. K. Goyal, *Finite Element Analysis and Optimization for Engineering Design Using MATLAB Programming*, Book proposal submitted, 2008.

Jaramillo, H. (1993) *Automatic Dimensioning and Tolerances*, Thesis (M.S.), University of Puerto Rico Mayagüez Campus.

Liberty, J., Horvath, D. B. (2005) *C++*, Indianapolis, Sam's Publishing

Light, R. and Gossard, D., (1982). *Modification of Geometric Models through Variational Geometry*, Computer Aided Design, 14(4), 209-214.

Maarten, J. and Van, E., (1989). *Creation and Modification of Parameterized Solid Models by Graphical Interaction*, Computer & Graphics, 13(1), 71-76.

Pabón Irizarry, I. U. (1996) *Artificial Intelligence in Automatic Dimensioning Layout*, Thesis (M.S.), University of Puerto Rico Mayagüez Campus.

Pérez Jiménez, A. (1993) *Finite Element Modeling and Optimization in a Constraint-based Environment*, Thesis (M.S.), University of Puerto Rico Mayagüez Campus.

Serrano, D. (1984) *MATHPAK: An Interactive Preliminary Design Package*, Thesis (M.S.), Massachusetts Institute of Technology.

Serrano, D. (1987) *Constraint Management in Conceptual Design*, Thesis (Sc.D.), Massachusetts Institute of Technology.

Serrano, D. (1991) *Automatic Dimensioning in Design for Manufacturing*, ACM OB9791-427-9/91, pp. 379-386.

Shigley, J.E., (1989). *Mechanical Engineering Design* 5th ed., New York, NY: McGraw-Hill.

Stroustrup, Bjarne, (2000). *The C++ Programming Language*, Special Edition, Addison-Wesley, ISBN 0-201-70073-5.

Suzuki, H., Ando, H. & Kimura, F., (1990). *Geometric Constraints and Reasoning for Geometrical CAD Systems*. , Computer & Graphics, 14(2), 211-224.

Tickoo, S., Kanthe, A. P. (2007). *NX 5 for designers*. New York: CADCIM Technologies. ISBN: 978-1-932709-40-7

UGS Corp. (2007) *'Intermediate NX Design and Assemblies with Teamcenter Integration – Student Guide'*, Publication Number: MT10056-TC-S – NX 5.

Ullman, L., Signer, A., (2006) *C++ Programming*, Berkeley: Peachpit Press.

WEB: http://design.osu.edu/carlson/history/lesson10.html . Carlson, W (2003). *A Critical History of Computer Graphics and Animation, Section 10: CAD/CAM/CADD/CAE.* The Ohio State University

WEB: http://www.me.mtu.edu/~bettig/MEEM5408/ (2008) *Design Automation: Theories and Implementation*.

WEB: http://www.open-std.org/jtc1/sc22/wg21/. *ISO/IEC JTC1/SC22/WG21 - The C++ Standards Committee* (2008).

Yuen, M. M., Tan, S. T. and Yu, K. M., (1988). *Scheme for Automatic Dimensioning of CGS Defined Parts*, Computer Aided Design, 20(3), 151-159.

APPENDIX A: C++ Functions and Libraries used with UGS-NX5

Table A.1: C++ Libraries used to Interface with UGS-NX5

uf_disp.h	Contains the prototypes and descriptions of the display functions.
uf.h	Open C API general interface.
uf_ui.h	This is the Open C API interface to the NX user interface.
uf_exit.h	A user exit is an optional feature that allows running Open C API programs automatically at certain predefined locations (or exits) in NX.
uf_object_types.h	Define names of all NX object types and subtypes.
uf_modl.h	Perform various modeling operations which include functions for: creation of primitives and features; querying modeling objects; creating, deleting, and editing expressions.
uf_part.h	Perform interactive functions that can be performed within NX plus query routines.
uf_obj.h	Get the version of the part in which the given object was created and the version in which it was last modified.
uf_curve.h	Provides routines that enable manipulation of points and curves.
uf_sket.h	Open C API interface to the NX sketcher.
uf_layer.h	Access to the NX database.

Table A.2: C++ Functions within UGS-NX5

UF_CALL	Calls other UF Functions
UF_LAYER_set_status	Sets layer status
UF_UI_select_with_class_dialog	Used to select multiple objects with the class selection dialog.
UF_UI_SEL_SCOPE_WORK_PART	Allows selecting only objects which belong to the work part.
UF_DISP_set_highlight	Highlights all the selected objects
UF_OBJ_set_layer	Set all the objects to a work layer
UF_MODL_ask_curve_props	Returns the point, tangent, unit principal normal, unit binormal, torsion, and radius of curvature on a curve at a given parameter.
UF_SKET_initialize_sketch	Initializes the NX sketch environment
UF_SKET_create_sketch	Creates empty NX sketch
UF_UI_select_with_single_dialog	Selects a single object with the single selection dialog

Table A.3: Additional C++ Functions within UGS-NX5

UF_CURVE_line_t	Input Coordinates of line in absolute space
UF_CURVE_create_line	Creates a line
UF_SKET_add_objects	Adds geometric objects to the current sketch
UF_SKET_dim_object_t	Create object dimensions
UF_SKET_create_dimensional_constraint	Creates dimensional constraints
UF_SKET_con_geom_s	Create geometric information of object to be constrained
UF_SKET_create_geometric_constraint	Create geometric constraints
UF_CURVE_create_arc_thru_3pts	Create an arc from given parameters
UF_SKET_ask_sketch_status	Returns the sketch status

APPENDIX B: C++ Code

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// CHARACTERIZATION OF BIDIMENSIONAL GEOMETRY THROUGH
// SOLID MODELING SOFTWARE USING A C++ INTERFACE
//
// Program by:
// Faculty Advisor: Dr. Vijay K. Goyal
// Graduate Student: Neit J. Nieves-Flores
// January 2009
//
// Project Submitted in Partial Fulfillment of the Requirements for the Degree of
// Master of Engineering in Mechanical Engineering
//
// University of Puerto Rico at Mayaguez
// Department of Mechanical Engineering
// Mayaguez, Puerto Rico 00680
// vgoyal@uprm.edu
// neit.nieves@gmail.com
//
// GLOBAL VARIABLES
//          layerY          //work layer
//          layerZ          //work layer
//          objectsID       //array of selected objects
//          startPoint      //point from UF Function
//          endPoint        //point from UF Function
//          startX          //Array of start points in X direction
//          startY          //Array of start points in Y direction
//          startZ          //Array of start points in Z direction
//          endX            //Array of end points in X direction
//          endY            //Array of end points in Y direction
//          endZ            //Array of end points in Z direction
//
// LOCAL VARIABLES
//          cue             //text variable for selection gui
//          title           //text variable for selection gui
//          response        //user selection (OK, BACK or CANCEL)
//          tangent         //tangent output from UF Function
//          p_norm          //unit principal normal output from UF Function
//          b_norm          //unit binormal output from UF Function
//          torsion         //torsion output from UF Function
//          rad_of_cur      //radius of curvature output from UF Function
//          currentLine     //current line being
//          evalLine        //line being compared to current
//          currentStartX   //start point in X direction
//          currentStartY   //start point in Y direction
//          currentStartZ   //start point in Z direction
//          currentEndX     //end point in X direction
//          currentEndY     //end point in Y direction
//          currentEndZ     //end point in Z direction
//          goodLine        //used to move the current line if another line conected or close to it is found
//          tol             //lines start and end points comparison tolerance
//          name            //character variable
```

```
//          sketch_tag          //parameter used for UF sketch functions
//          outward             //parameter used for UF sketch functions
//          option              //parameter used for UF sketch functions
//          opt_mat             //Option Matrix used for UF sketch functions
//          h                   //flag variable for horizontal constraints
//          v                   //flag variable for vertical constraints
//          cursor              //parameter used for UF functions
//          startpoint          //point output from UF Function
//          endpoint            //point output from UF Function
//          object              //parameter used for UF functions
//          view                //parameter used for UF functions
//          objarray1           //parameter used for UF functions
//          line                //coordinates of selected line in absolute space
//          dim_objs            //object dimensions
//          con_tag             //parameter used for constraint UF functions
//          con_geom            //parameter used for constraint UF functions
//          create_flag1        //parameter used for arc UF functions
//          first_point1        //parameter used for arc UF functions
//          second_point1       //parameter used for arc UF functions
//          third_point1        //parameter used for arc UF functions
//          arc_tag1            //parameter used for arc UF functions
//          dim_objsarc         //parameter used for arc UF functions
//
// OUTPUT VARIABLES
//          count               //count of selected objects
//          goodLineArr         //Array of Good Lines
//          sket_status         //current sketch status
//          dof_needed          //degrees of freedom needed for fully constraint if under-constrained
//
// TEMPORARY VARIABLES
//          i                   //counter
//          j                   //counter
//
// FUNCTIONS:
//          layerSelectable     //Makes all the layers Selectable and Sets layerZ to be the Work Layer
//          selectAll           //Gets all the IDs of all the objects and moves them to layerZ
//          getData             //Gets all the objects start and end points
//          compare1            //Compares the start and end points of all lines and arcs
//          setLayerY           //Sets layerY to be the only active layer
//          dimsConsts          //Select Objects and Apply Dimensions and Geometric Constraints
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// INITIAL DEFINITIONS
//
// UGS LIBRARIES
//
#include <uf_disp.h>            //Contains the prototypes and descriptions of the display functions.
#include <uf.h>                 //Open C API general interface.
#include <uf_ui.h>              //This is the Open C API interface to the NX user interface.
#include <uf_exit.h>            /*A user exit is an optional feature that allows to run Open C API programs
                                automatically at certain predefined locations (or exits) in NX.*/
#include <uf_object_types.h>            //Define names of all NX object types and subtypes.
#include <uf_modl.h>            /*Perform various modeling operations which include functions for: creation of primitives
                                and features; querying modeling objects; creating, deleting, and editing expressions*/
#include <uf_part.h>            //Perform interactive functions that can be performed within NX plus query routines.
#include <uf_obj.h>             /*Get the version of the part in which the given object was created and the version
```

```
                        in which it was last modified.*/
#include <uf_curve.h>       //Provides routines that enable manipulation of points and curves.
#include <uf_sket.h>        //Open C API interface to the NX sketcher.
#include <uf_layer.h>       //Access to the NX database.
////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Definition UF_CALL
#define UF_CALL(X) (report( __FILE__, __LINE__, #X, (X)))          //used to enable calling of other UF
Functions
static int report( char *file, int line, char *call, int irc)      //parameters for UF_CALL
{return(irc);}
//
////GLOBAL VARIABLES/////
int layerZ=251,layerY=250;          //work layers
int     count=1;                    //count of selected objects
tag_p_t objectsID;                  //Array of selected objects
double startPoint[ 3 ];             //Point from UF Function
double endPoint[ 3 ];               //Point from UF Function
double startX[1000];                //Array of start points in X direction
double startY[1000];                //Array of start points in Y direction
double startZ[1000];                //Array of start points in Z direction
double endX[1000];                  //Array of end points in X direction
double endY[1000];                  //Array of end points in Y direction
double endZ[1000];                  //Array of end points in Z direction
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// FUNCTION 1:  layerSelectable
// Purpose:  Makes all the layers selectable
//
// Local Variables:
//      i               //counter
//      layerZ          //work layer
//
// Output variables:
//      None
//
// UF Functions called:
//      UF_LAYER_set_status         //sets layer status
//
// Called from:
//      Main Program
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////
void layerSelectable(void)
 {
 int i;
      for (i=1;i<257;i++)
      {
      UF_LAYER_set_status(i,2);       //This will set all the layers to be selectable
      }
      UF_LAYER_set_status(layerZ,1);  // This will set the layer Z to be the work Layer
 }
//
//END layerSelectable
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
//  FUNCTION 2:  selectAll
```

```
//  Purpose:  Display Object Selection Dialog
//
//  Local Variables
//          cue                 //text variable for selection gui
//          title               //text variable for selection gui
//          response            //user selection (OK, BACK or CANCEL)
//          i                   //counter
//          layerY              //work layer
//          layerZ              //work layer
//
//  Output variables
//          count               //count of selected objects
//           objectsID          //array of selected objects
//
//  Local UF Functions
//     UF_UI_select_with_class_dialog          //used to select multiple objects with the class selection dialog
//     UF_UI_SEL_SCOPE_WORK_PART               //Allows to select only objects which belong to the work part
//     UF_DISP_set_highlight                   //Highlights all the selected objects
//     UF_OBJ_set_layer                        //Set all the objects to a work layer
//     UF_LAYER_set_status                     //Sets layer status
//
//  Called from
//     Main Program
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
static void selectAll(void)
   {
          char cue[] = "Select all the Objects that you want to Analyze";
   char title[] = "Line and Arc Selection Dialog";
   int i, response;
//
   if((UF_CALL(UF_UI_select_with_class_dialog(
       cue,                                //Message for cue
                    title,                 //Dialog Title
                    UF_UI_SEL_SCOPE_WORK_PART,
                                           //Allows to select only objects which belong to the work part.
       NULL,                               //Selection Initialization procedure
                    NULL,                        //User data for intitialization
                    &response,                   //OUTPUT - UF_UI_BACK, UF_UI_CANCEL, UF_UI_OK
                    &count,                      //OUTPUT - Count of selected objects
                    &objectsID))) == 0)          //OUTPUT - Array of objects selected
             {
       if (response == UF_UI_OK && count > 0)
          {
          for (i=0; i<count; i++)
            {
            UF_DISP_set_highlight(objectsID[i], 0);     //Highlights all the selected objects
                                           UF_OBJ_set_layer(objectsID[i],layerZ);      //Set all the objects to layerZ
            }
          }
               }
            for (i=1;i<257;i++)
               {
                       if (i!=layerZ && i!=layerY)
                         {
                         UF_LAYER_set_status(i,4);     //This will set all the layers to be inactive exept layer Y and layer Z
                         }
               }
```

```
   }
//
//END selectAll
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
//  FUNCTION 3: getData
//  Purpose:
//                      Gets all the start points and end points for all the lines and arcs,
//     and stores them in arrays
//
//  Local Variables
//          i                   //counter
//          tangent             //tangent output from UF Function
//          p_norm              //unit principal normal output from UF Function
//          b_norm              //unit binormal output from UF Function
//          torsion             //torsion output from UF Function
//          rad_of_cur          //radius of curvature output from UF Function
//          startPoint          //point output from UF Function
//          endPoint            //point output from UF Function
//
//  Output Variables
//          startX              //Array of start points in X direction
//          startY              //Array of start points in Y direction
//          startZ              //Array of start points in Z direction
//          endX                //Array of end points in X direction
//          endY                //Array of end points in Y direction
//          endZ                //Array of end points in Z direction
//
//  UF Functions called
//          UF_MODL_ask_curve_props
                                //Returns the point, tangent, unit principal normal, unit binormal,
//                              //torsion, and radius of curvature on a curve at a given parameter.
//
//  Called from
//     Main Program
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void getData(void)
   {
          int i;
          double tangent[ 3 ];
          double p_norm[ 3 ];
          double b_norm[ 3 ];
          double torsion;
          double rad_of_cur;

          for(i=0;i<count;i++)
     {
////////Save Start and End points of the Objects
                    /*Returns the point, tangent, unit principal normal, unit binormal,
                      torsion, and radius of curvature on a curve at a given parameter.*/

UF_MODL_ask_curve_props (objectsID[i], 0, startPoint, tangent, p_norm, b_norm, &torsion, &rad_of_cur );
UF_MODL_ask_curve_props (objectsID[i], 1, endPoint, tangent, p_norm, b_norm, &torsion, &rad_of_cur );
                    startX[i]=startPoint[0];  //Saves startpoint X in a in an Array
                    startY[i]=startPoint[1];  //Saves startpoint Y in a in an Array
                    startZ[i]=startPoint[2];  //Saves startpoint Z in a in an Array
```

```
                    endX[i]=endPoint[0];     //Saves endpoint X in a in an Array
                    endY[i]=endPoint[1];     //Saves endpoint Y in a in an Array
                    endZ[i]=endPoint[2];     //Saves endpoint Z in a in an Array
               }
    }
//
//End getData
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
//  FUNCTION 4: compare1
//  Purpose:
//                    Compares the start and end points of all lines and arcs to determine whether they are part
//                    of the cross section of interest or excess lines that have to be removed.
//
//  Local Variables
//          i                   //counter
//          j                   //counter
//          startX              //Array of start points in X direction
//          startY              //Array of start points in Y direction
//          startZ              //Array of start points in Z direction
//          endX                //Array of end points in X direction
//          endY                //Array of end points in Y direction
//          endZ                //Array of end points in Z direction
//          currentLine         //current line
//          evalLine            //line being compared to current
//          currentStartX       //start point in X direction
//          currentStartY       //start point in Y direction
//          currentStartZ       //start point in Z direction
//          currentEndX         //end point in X direction
//          currentEndY         //end point in Y direction
//          currentEndZ         //end point in Z direction
//          goodLine            //used to move the current line if another line conected or close to it is found
//          tol                 //lines start and end points comparison tolerance
//
//  Output variables
//          goodLineArr         //Array of Good Lines
//
//  UF Functions called
//          UF_OBJ_set_layer            //Set all the objects to a work layer
//
//  Called from
//          Main Program
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int goodLineArr[1000];
int currentLine,evalLine;
double currentStartX,currentStartY,currentStartZ,currentEndX,currentEndY,currentEndZ;
void compare1(void)
{
int i,j;
int goodLine;   /*This variable is used to move the current line if another
                                        line conected to it or close to it is found.
                                        If another line is found it will be set to 1.*/

double tol=.01;   //This is the tolerance that the program will use to evaluate the lines start and end points.

for(i=0;i<count;i++)  //Controls the line being evaluated.
```

```
   {
////Stores the current line data in local variables
            currentLine=objectsID[i];
            currentStartX=startX[i];
            currentStartY=startY[i];
            currentStartZ=startZ[i];
            currentEndX=endX[i];
            currentEndY=endY[i];
            currentEndZ=endZ[i];

////Searches for lines having the same start and end points.

            for(j=0;j<count;j++)            // This for will cycle thru all the lines to compare all of them with the current line.
                        {
      goodLine=0;                           //Set variable goodline equal to zero
                        evalLine=0;                 //resets evalLine
                        evalLine=objectsID[j];   //Gets the ID of line being evaluated
                                    if (i != j)
                                    {
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///////////////Searches for lines having the same Start Point

//////////////////Compares start point in the X Coordinate
                                                            if ((currentStartX<=startX[j]+tol && currentStartX>=startX[j]-tol) ||
(currentStartX<=endX[j]+tol
                                                                        && currentStartX>=endX[j]-tol))
                        {
                                                            if (currentStartX==startX[j])
                                                                        {

                                                                        }
                                                                        else if (currentStartX==endX[j])
                          {
                                                                        }

//////////////////////Compares start point in the Y Coordinate
                                                            if ((currentStartY<=startY[j]+tol && currentStartY>=startY[j]-
tol) || (currentStartY<=endY[j]+tol
                                                                        && currentStartY>=endY[j]-tol))
                                    {
                                                                                    if (currentStartY==startY[j])
                                    {
                                                                                    }
                                                                                    else if (currentStartY==endY[j])
                                    {
                                                                                    }

//////////////////////Compares start point in the Z Coordinate
                                                            if ((currentStartZ<=startZ[j]+tol && currentStartZ>=startZ[j]-
tol) || (currentStartZ<=endZ[j]+tol
                                                                        && currentStartZ>=endZ[j]-tol))
                                    {
                                                                                    if (currentStartZ==startZ[j])
                                    {
                                                                                    }
                                                                                    else if (currentStartZ==endZ[j])
                                    {
                                                                                    }
```

```
//
////////////////////////////If the start point matches it will send the currentLine and evalLine to layerY
                                                                UF_OBJ_set_layer(evalLine, layerY);  //Sets the
evaluated line to layer Y
//
                                                                goodLine=1;                     //Set goodline equal to
1 so that it will be sent to layer Y
                                                        }}}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///////////////Searches for lines having the same End Point//////////////////////////////////

////////////////////Compares End Point in the X Coordinate
                                                if ((currentEndX<=startX[j]+tol && currentEndX>=startX[j]-tol) ||
(currentEndX<=endX[j]+tol
                                                        && currentEndX>=endX[j]-tol))
                        {
                                                                if (currentEndX==startX[j])
                        {
                                                                }
                                                                if (currentStartZ==endZ[j])
                        {
                                                                }

////////////////////Compares End Point in the Y Coordinate
                                                if ((currentEndY<=startY[j]+tol && currentEndY>=startY[j]-tol) ||
(currentEndY<=endY[j]+tol
                                                        && currentEndY>=endY[j]-tol))
                        {
                                                                if (currentEndY==startY[j])
                        {
                                                                }
                                                                if (currentStartZ==endZ[j])
                        {
                                                                }

////////////////////Compares End Point in the Z Coordinate
                                                if ((currentEndZ<=startZ[j]+tol && currentEndZ>=startZ[j]-tol) ||
(currentEndZ<=endZ[j]+tol
                                                        && currentEndZ>=endZ[j]-tol))
                        {
                                                                if (currentEndZ==startZ[j])
                        {
                                                                }
                                                                if (currentStartZ==endZ[j])
                        {
                                                                }

////////////////////////////If the End Point matches it will send the currentLine and evalLine to layerY

                        UF_OBJ_set_layer(evalLine, layerY);   //Sets the evaluated line to layer Y
                                                                goodLine=1;                     //Set goodline equal to
1 so that it wil be sent to layer Y
                                                        }}}

                                        if(goodLine==1)   // If good line is equal to 1, it will send current line to layer Y
                {
                                        UF_OBJ_set_layer(currentLine,layerY);  //Function sets line to layerY
                                        goodLineArr[i]=1;
```

```
                                        }
                                        goodLineArr[i]=0;
                                }
                        }
                }
        }
}
//
//END compare1
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
//  FUNCTION 5: setLayerY
//  Purpose:  Sets layerY to be the only active and shown layer
//
//  Local Variables
//          i                               //counter
//
//  Output variables
//          None
//
//  UF Functions called
//          UF_LAYER_set_status         //Sets work layer status
//
//  Called from
//          Main Program
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void setLayerY(void)
   {
        int i;
        for (i=1;i<257;i++)
           {
                if (i!=layerY)
                {
                UF_LAYER_set_status(i,4);  //Set all layers to be inactive
                }
        UF_LAYER_set_status(layerY,1);       //Set layerY to be the Active and only shown layer
           }
   }
//
//END setLayerY
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
//  FUNCTION 6: dimsConsts
//  Purpose:  Select Lines or Arcs and Apply Dimensions and Geometric Constraints
//
//  Local Variables
//          name                //character variable
//          sketch_tag          //parameter used for UF sketch functions
//          outward             //parameter used for UF sketch functions
//          option              //parameter used for UF sketch functions
//          opt_mat             //Option Matrix used for UF sketch functions
//          cue                 //character variable for gui
//          title               //character variable for gui
//          response            //user selection (OK, BACK or CANCEL)
//          h                   //flag variable set to 1 for horizontal constraints
//          v                   //flag variable set to 1 for vertical constraints
```

```
//          i                   //counter
//          j                   //counter
//          cursor              //parameter used for UF functions
//          tangent             //tangent output from UF Function
//          p_norm              //unit principal normal output from UF Function
//          b_norm              //unit binormal output from UF Function
//          torsion             //torsion output from UF Function
//          rad_of_cur          //radius of curvature output from UF Function
//          startpoint          //point output from UF Function
//          endpoint            //point output from UF Function
//          object              //parameter used for UF functions
//          view                //parameter used for UF functions
//          objarray1           //parameter used for UF functions
//          line                //coordinates of selected line in absolute space
//          dim_objs            //object dimensions
//          con_tag             //parameter used for constraint UF functions
//          con_geom            //parameter used for constraint UF functions
//          create_flag1        //parameter used for arc UF functions
//          first_point1        //parameter used for arc UF functions
//          second_point1       //parameter used for arc UF functions
//          third_point1        //parameter used for arc UF functions
//          arc_tag1            //parameter used for arc UF functions
//          dim_objsarc         //parameter used for arc UF functions
//
// Output variables
//          sket_status         //current sketch status
//          dof_needed          //degrees of freedom needed for fully constraint if under-constrained
//
//  UF Functions called
//          UF_SKET_initialize_sketch           //Initializes the NX sketch environment
//          UF_SKET_create_sketch               //Creates empty NX sketch
//          UF_UI_select_with_single_dialog     //Selects a single object with the single selection dialog
//          UF_DISP_set_highlight               //highlight or unhighlight the selected object
//          UF_MODL_ask_curve_props             //Returns the point, tangent, unit principal normal, unit binormal,
//                                              //torsion, and radius of curvature on a curve at a given parameter.
//          UF_CURVE_line_t                     //Input     Coordinates of line in absolute space
//          UF_CURVE_create_line                //Creates a line
//          UF_SKET_add_objects                 //Adds geometric objects to the current sketch
//          UF_SKET_dim_object_t                //Create object dimensions
//          UF_SKET_create_dimensional_constraint       //Creates dimensional constraints
//          UF_SKET_con_geom_s                  //Create geometric information of object to be constrained
//          UF_SKET_create_geometric_constraint         //Create geometric constraints
//          UF_CURVE_create_arc_thru_3pts       //Create an arc from given parameters
//          UF_SKET_ask_sketch_status           //Returns the sketch status
//
//  Called from
//          Main Program
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
void dimsConsts(void)
{
//Initialize the Sketch
        char name [30] = "Parametric Sketch";  //Character variable
        tag_t sketch_tag = NULL_TAG;
        UF_CALL(UF_SKET_initialize_sketch (name, &sketch_tag));  //Initializes the NX sketch environment
//
//////Create Sketch 1
```

```
            //Sketch Variables
            int outward =1;
            int option = 2;
            double opt_mat[9];
//
            //Option Matrix
            // X unit directions
            opt_mat[0] = 1;
            opt_mat[1] = 0;
            opt_mat[2] = 0;
            // Y unit directions
            opt_mat[3] = 0;
            opt_mat[4] = 1;
            opt_mat[5] = 0;
            // CSYS origin point
            opt_mat[6] = 0;
            opt_mat[7] = 0;
            opt_mat[8] = 0;
//
////Create Empty Sketch
            UF_SKET_create_sketch (name, option, opt_mat, NULL, NULL, outward, &sketch_tag );
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////Prompts the User to Select HORIZONTAL LINES
//
            char cue[] = "Select a Horizontal Line";
   char title[] = "Horizontal Line";
//
//*****LOCAL VARIABLES*****//
            int response=UF_UI_CANCEL;
            int v,h;
            int i=1;
            int j=1;
   double cursor[3];
            double startpoint [ 3 ];
            double endpoint [ 3 ];
            double tangent [ 3 ];
            double p_norm [ 3 ];
            double b_norm [ 3 ];
            double * torsion=0;
            double * rad_of_cur=0;
            tag_t object, view, objarray1[4];
//
            while (response!=UF_UI_OK)
            {
            UF_CALL(UF_UI_select_with_single_dialog(cue,title, UF_UI_SEL_SCOPE_NO_CHANGE, NULL, NULL,
&response,
                                                                                                    &object,
cursor, &view));
            /*Selects a single object with the single selection dialog. The object can
              be selected with the cursor or by entering a name. The object is
              highlighted*/
//
                        UF_DISP_set_highlight(object,0);  //unhighlight selected object
                        if(response!=UF_UI_OK)
                        {
////////Save Start and End points of the Selected Line
                        /*Returns the point, tangent, unit principal normal, unit binormal,
```

```
                    torsion, and radius of curvature on a curve at a given parameter.*/
                    UF_MODL_ask_curve_props (object, 0, startpoint, tangent, p_norm, b_norm, torsion, rad_of_cur );
                    UF_MODL_ask_curve_props (object, 1, endpoint, tangent, p_norm, b_norm, torsion, rad_of_cur );
//
////////Redraw the line
                    UF_CURVE_line_t line1;  //Input        Coordinates of line in absolute space
                    line1.start_point[0] = startpoint[0];
                    line1.start_point[1] = startpoint[1];
                    line1.start_point[2] = startpoint[2];
                    line1.end_point[0] = endpoint[0];
                    line1.end_point[1] = endpoint[1];
                    line1.end_point[2] = endpoint[2];
//
                    UF_CURVE_create_line(&line1, &objarray1[0]);                    //Creates the Line
//
                    UF_SKET_add_objects (sketch_tag,1, &objarray1[0]);  //Adds geometric objects to the current sketch.
//
////////Applying Dimensional Constraint to the Lines
//
    UF_SKET_dim_object_t dim_objs[2];  //Creates Object Dimensions
        tag_t con_tag;
        dim_objs[0].object_tag = objarray1[0];
        dim_objs[0].object_assoc_type = UF_SKET_end_point;
        dim_objs[0].object_assoc_mod_value = UF_SKET_first_end_point;
        dim_objs[1].object_tag = objarray1[0];
        dim_objs[1].object_assoc_type = UF_SKET_end_point;
        dim_objs[1].object_assoc_mod_value = UF_SKET_last_end_point;
//
                //Creates Dimensional Constraint
        UF_SKET_create_dimensional_constraint (sketch_tag, UF_SKET_horizontal_dim, 2, dim_objs, cursor, &con_tag
);
//
                if (i==1)
                {
                i=i+1;
                tag_t con_tag12;
//
/////////Applying Geometric Constraint to the Lines
                UF_SKET_con_geom_s con_geom2[1];
                con_geom2[0].geom_tag=objarray1[0];
                con_geom2[0].vertex_type=UF_SKET_no_vertex;
                con_geom2[0].vertex_index=NULL;
                con_geom2[0].use_help=UF_SKET_no_help_data;
//
                                //Creates Geometric Constraints
                UF_SKET_create_geometric_constraint(sketch_tag,UF_SKET_fixed,1,con_geom2,&con_tag12);
                h=1;
                }
        tag_t con_tag11;
//
        UF_SKET_con_geom_s con_geom1[1];
        con_geom1[0].geom_tag=objarray1[0];
        con_geom1[0].vertex_type=UF_SKET_no_vertex;
        con_geom1[0].vertex_index=NULL;
        con_geom1[0].use_help=UF_SKET_no_help_data;
//
                //Creates Geometric Constraints
        UF_SKET_create_geometric_constraint(sketch_tag,UF_SKET_horizontal,1,con_geom1,&con_tag11);
```

```
                }
        }
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////Prompts the User to Select VERTICAL LINES
//
        char cue2[] = "Select a Vertical Line ";
   char title2[] = "Vertical Line";
   response=UF_UI_CANCEL;
//
        while (response!=UF_UI_OK)
        {
        UF_CALL(UF_UI_select_with_single_dialog(cue2,title2, UF_UI_SEL_SCOPE_NO_CHANGE, NULL, NULL,
                                                                                        &response,
&object, cursor, &view));
        /*Selects a single object with the single selection dialog. The object can
          be selected with the cursor or by entering a name. The object is
          highlighted*/
//
        UF_DISP_set_highlight(object,0);  //unhighlight selected object
//
                if(response!=UF_UI_OK)
                {
////////Save Start and End points of the Selected Line
                /*Returns the point, tangent, unit principal normal, unit binormal,
                  torsion, and radius of curvature on a curve at a given parameter.*/
                UF_MODL_ask_curve_props (object, 0, startpoint, tangent, p_norm, b_norm, torsion, rad_of_cur );
                UF_MODL_ask_curve_props (object, 1, endpoint, tangent, p_norm, b_norm, torsion, rad_of_cur );
//
////////Redraw the line
                UF_CURVE_line_t line2;  //Input        Coordinates of line in absolute space
                line2.start_point[0] = startpoint[0];
                line2.start_point[1] = startpoint[1];
                line2.start_point[2] = startpoint[2];
                line2.end_point[0] = endpoint[0];
                line2.end_point[1] = endpoint[1];
                line2.end_point[2] = endpoint[2];
//
                UF_CURVE_create_line(&line2, &objarray1[1]);  //Creates the Line
//
                UF_SKET_add_objects (sketch_tag,1, &objarray1[1]);//Adds geometric objects to the current sketch.
//
////Applying Dimensional Constraint to the Lines
        tag_t con_tag;
        UF_SKET_dim_object_t dim_objs2[2];  //Creates Object Dimensions
        dim_objs2[0].object_tag = objarray1[1];
        dim_objs2[0].object_assoc_type = UF_SKET_end_point;
        dim_objs2[0].object_assoc_mod_value = UF_SKET_first_end_point;
        dim_objs2[1].object_tag = objarray1[1];
        dim_objs2[1].object_assoc_type = UF_SKET_end_point;
        dim_objs2[1].object_assoc_mod_value = UF_SKET_last_end_point;
//
////////Create Dimensional Constraints
        UF_SKET_create_dimensional_constraint (sketch_tag, UF_SKET_vertical_dim, 2, dim_objs2, cursor, &con_tag
);
                if (j==1)
                {
           j=j+1;
//
```

```
//////////Applying Geometric Constraint to the Lines
            tag_t con_tag4;
          UF_SKET_con_geom_s con_geom4[1];
          con_geom4[0].geom_tag=objarray1[1];
          con_geom4[0].vertex_type=UF_SKET_no_vertex;
          con_geom4[0].vertex_index=NULL;
          con_geom4[0].use_help=UF_SKET_no_help_data;
//
//////////Creates Geometric Constraints
          UF_SKET_create_geometric_constraint(sketch_tag,UF_SKET_fixed,1,con_geom4,&con_tag4);
                    v=1;
                    }
          tag_t con_tag1;
          UF_SKET_con_geom_s con_geom[1];
          con_geom[0].geom_tag=objarray1[1];
          con_geom[0].vertex_type=UF_SKET_no_vertex;
          con_geom[0].vertex_index=NULL;
          con_geom[0].use_help=UF_SKET_no_help_data;
//
//////////Creates Geometric Constraints
          UF_SKET_create_geometric_constraint(sketch_tag,UF_SKET_vertical,1,con_geom,&con_tag1);
                    }
          }
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////Prompts the User to Select DIAGONAL LINES
//
          char cue4[] = "Select a Diagonal Line ";
   char title4[] = "Diagonal Line";
   response=UF_UI_CANCEL;
//
          while (response!=UF_UI_OK)
          {
          UF_CALL(UF_UI_select_with_single_dialog(cue4,title4, UF_UI_SEL_SCOPE_NO_CHANGE, NULL, NULL,
                                                                                    &response,
&object, cursor, &view));
          /*Selects a single object with the single selection dialog. The object can
            be selected with the cursor or by entering a name. The object is
            highlighted*/
//
      UF_DISP_set_highlight(object,0);  //unhighlight selected object
//
                    if(response!=UF_UI_OK)
                    {
//////////Save Start and End points of the Selected Line
                    /*Returns the point, tangent, unit principal normal, unit binormal,
                      torsion, and radius of curvature on a curve at a given parameter.*/
//
                    UF_MODL_ask_curve_props (object, 0, startpoint, tangent, p_norm, b_norm, torsion, rad_of_cur );
                    UF_MODL_ask_curve_props (object, 1, endpoint, tangent, p_norm, b_norm, torsion, rad_of_cur );
//
//////////Redraw the line
                    UF_CURVE_line_t line3;  //Input       Coordinates of line in absolute space
                    line3.start_point[0] = startpoint[0];
                    line3.start_point[1] = startpoint[1];
                    line3.start_point[2] = startpoint[2];
                    line3.end_point[0] = endpoint[0];
                    line3.end_point[1] = endpoint[1];
                    line3.end_point[2] = endpoint[2];
```

```
//
                        UF_CURVE_create_line(&line3, &objarray1[2]);  //Creates the Line
//
                        UF_SKET_add_objects (sketch_tag,1, &objarray1[2]);  //Adds geometric object to the current sketch
//
////Applying Dimensional Constraint to the Lines
            tag_t con_tag2;
            UF_SKET_dim_object_t dim_objs2[2];//Creates Object Dimensions
            dim_objs2[0].object_tag = objarray1[2];
            dim_objs2[0].object_assoc_type = UF_SKET_end_point;
            dim_objs2[0].object_assoc_mod_value = UF_SKET_first_end_point;
            dim_objs2[1].object_tag = objarray1[2];
            dim_objs2[1].object_assoc_type = UF_SKET_end_point;
            dim_objs2[1].object_assoc_mod_value = UF_SKET_last_end_point;
//
                        //Creates Dimensional Constraint
            UF_SKET_create_dimensional_constraint (sketch_tag, UF_SKET_vertical_dim, 2, dim_objs2, cursor, &con_tag2
);
//
            if (v!=1 || h!=1)
            {
            tag_t con_tag6;
            UF_SKET_dim_object_t dim_objs6[2];  //Creates Object Dimensions
            dim_objs6[0].object_tag = objarray1[2];
            dim_objs6[0].object_assoc_type = UF_SKET_end_point;
            dim_objs6[0].object_assoc_mod_value = UF_SKET_first_end_point;
            dim_objs6[1].object_tag = objarray1[2];
            dim_objs6[1].object_assoc_type = UF_SKET_end_point;
            dim_objs6[1].object_assoc_mod_value = UF_SKET_last_end_point;
//
////Create Dimensional Constraint
            UF_SKET_create_dimensional_constraint (sketch_tag, UF_SKET_fixed, 2, dim_objs6, cursor, &con_tag6 );
            }
//
            tag_t con_tag3;
            UF_SKET_dim_object_t dim_objs3[2];  //Creates Object Dimensions
            dim_objs3[0].object_tag = objarray1[2];
            dim_objs3[0].object_assoc_type = UF_SKET_end_point;
            dim_objs3[0].object_assoc_mod_value = UF_SKET_first_end_point;
            dim_objs3[1].object_tag = objarray1[2];
            dim_objs3[1].object_assoc_type = UF_SKET_end_point;
            dim_objs3[1].object_assoc_mod_value = UF_SKET_last_end_point;
//
////Create Dimensional Constraint
            UF_SKET_create_dimensional_constraint (sketch_tag, UF_SKET_horizontal_dim, 2, dim_objs3, cursor,
&con_tag3 );
                        }
            }
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////Prompts the User to Select ARCS
//
            char cue3[] = "Select an Arc ";
   char title3[] = "Arc";
   response=UF_UI_CANCEL;
   int create_flag1 = 1;
            tag_t arc_tag1=NULL;
            double first_point1[ 3 ];
            double second_point1[ 3 ];
```

```
            double third_point1[ 3 ];
//
            while (response!=UF_UI_OK)
            {
            UF_CALL(UF_UI_select_with_single_dialog(cue3,title3, UF_UI_SEL_SCOPE_NO_CHANGE, NULL, NULL,
                                                                                                    &response,
&object, cursor, &view));
            /*Selects a single object with the single selection dialog. The object can
              be selected with the cursor or by entering a name. The object is
              highlighted*/
//
   UF_DISP_set_highlight(object,0);  //unhighlight selected object
//
                        if(response!=UF_UI_OK)
                        {
////////Save Three points of the Selected Arc Line
                        /*Returns the point, tangent, unit principal normal, unit binormal,
                          torsion, and radius of curvature on a curve at a given parameter.*/
//
                        UF_MODL_ask_curve_props (object, 0, first_point1, tangent, p_norm, b_norm, torsion, rad_of_cur );
//Start Point
                        UF_MODL_ask_curve_props (object, 0.5, second_point1, tangent, p_norm, b_norm, torsion, rad_of_cur
);  //Mid Point
                        UF_MODL_ask_curve_props (object, 1, third_point1, tangent, p_norm, b_norm, torsion, rad_of_cur );
//End Point
//
////////Redraw the line
                        UF_CURVE_create_arc_thru_3pts (create_flag1, first_point1, second_point1, third_point1, &arc_tag1);
//Creates Arc
//
                        UF_SKET_add_objects (sketch_tag,1, &arc_tag1);  //Adds geometric object to the current sketch
//
////Applying Radial Dimensional Constraint to the Arcs
            UF_SKET_dim_object_t dim_objs2[2];  //Creates Object Dimensions
            tag_t con_tag2;
            dim_objs2[0].object_tag = arc_tag1;
            dim_objs2[0].object_assoc_type = UF_SKET_arc_center;
            dim_objs2[0].object_assoc_mod_value = NULL;
//
////////Creates Dimensional Constraint
            UF_SKET_create_dimensional_constraint (sketch_tag, UF_SKET_radius_dim, 1, dim_objs2, cursor, &con_tag2
);
//
////Applying Horizontal Dimensional Constraint to the Arcs
            tag_t con_tag3;
            UF_SKET_dim_object_t dim_objsarc[2];  //Creates Object Dimensions
            dim_objsarc[0].object_tag = arc_tag1;
            dim_objsarc[0].object_assoc_type = UF_SKET_arc_center;
            dim_objsarc[0].object_assoc_mod_value = NULL;
            dim_objsarc[1].object_tag = objarray1[0];
            dim_objsarc[1].object_assoc_type = UF_SKET_end_point;
            dim_objsarc[1].object_assoc_mod_value = UF_SKET_last_end_point;
//
////////Creates Dimensional Constraint
            UF_SKET_create_dimensional_constraint (sketch_tag, UF_SKET_horizontal_dim, 1, dim_objsarc, cursor,
&con_tag3 );
//
////Applying Vertical Dimensional Constraint to the Arcs
```

```
            tag_t con_tag9;
            UF_SKET_dim_object_t dim_objsarc2[2];  //Creates Object Dimensions
            dim_objsarc2[0].object_tag = arc_tag1;
            dim_objsarc2[0].object_assoc_type = UF_SKET_arc_center;
            dim_objsarc2[0].object_assoc_mod_value = NULL;
            dim_objsarc2[1].object_tag = objarray1[0];
            dim_objsarc2[1].object_assoc_type = UF_SKET_end_point;
            dim_objsarc2[1].object_assoc_mod_value = UF_SKET_last_end_point;
//
////////Creates Dimensional Constraint
            UF_SKET_create_dimensional_constraint (sketch_tag, UF_SKET_vertical_dim, 1, dim_objsarc2, cursor,
&con_tag9 );
//
            UF_SKET_status_t * sket_status=NULL;
            int * dof_needed=NULL;
            UF_SKET_ask_sketch_status(sketch_tag,sket_status,dof_needed);
                        /*This returns the given sketch's status and the degrees of freedom needed to make the sketch
                          full-constrained if it is currently under-constrained */
                        }
             }
}
//
//END dimsConsts
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ufusr(char *param, int *retcode, int param_len)
  {
  if (!UF_CALL(UF_initialize()))         /*Initializes the Open C API environment and allocates an Open C
                                                                                                    API execute license.*/
    {
            layerSelectable();
                        //Makes all the layers Selectable
                        //Sets layerZ to be the Work Layer
//
            selectAll();
                        //Dialog to select the lines
                        //Gets all the IDs of all the objects
                        //Moves all the objects to layerZ
//
            getData();
                        //Gets all the start points and end points for all the lines and arcs,
                        //and stores them in arrays
//
            compare1();
                        //Compares the start and end points of all lines and arcs to determine whether they are part
                        //of the cross section of interest or excess lines that have to be removed.
//
            setLayerY();
                        //Sets layerY to be the only active and shown layer
//
            dimsConsts();
                        //Select Lines or Arcs and Apply Dimensions and Geometric Constraints
//
            UF_CALL(UF_terminate());
                        //Terminates the Open C API environment
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
//  PROGRAM END
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

APPENDIX C: Step by Step Program Run Example

In this section we present a step by step walkthrough of our characterization tool using Example 1 in section 3.3.1. Figure C.1 shows the UGS-NX5 opening screen. From this screen the user can start new projects or open existing ones.

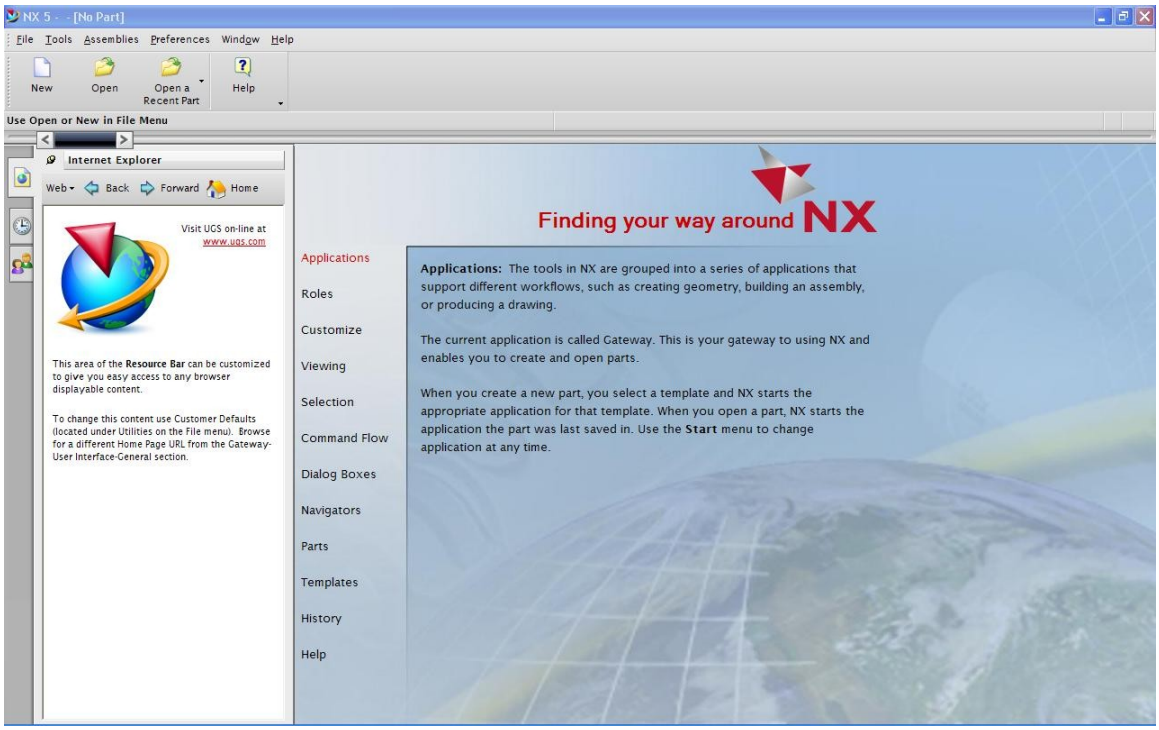

Figure C.1: UGS-NX5 Start Screen

In Figures C.2 and C.3 on the next page we show Example 1 being loaded into the UGS-NX5 environment. This example consists of a bidimensional geometry displaying the word *section* but it is covered in waste lines and arcs that make it nearly impossible to distinguish the geometry.

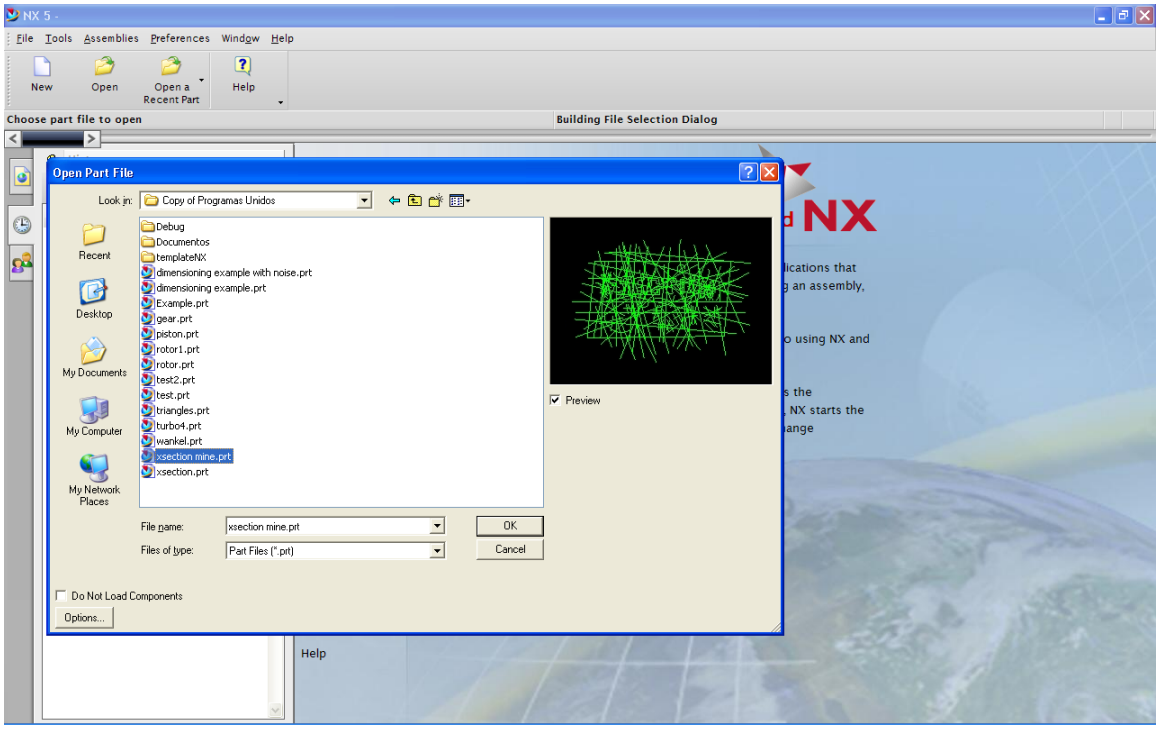

Figure C.2: Open File Dialog Box

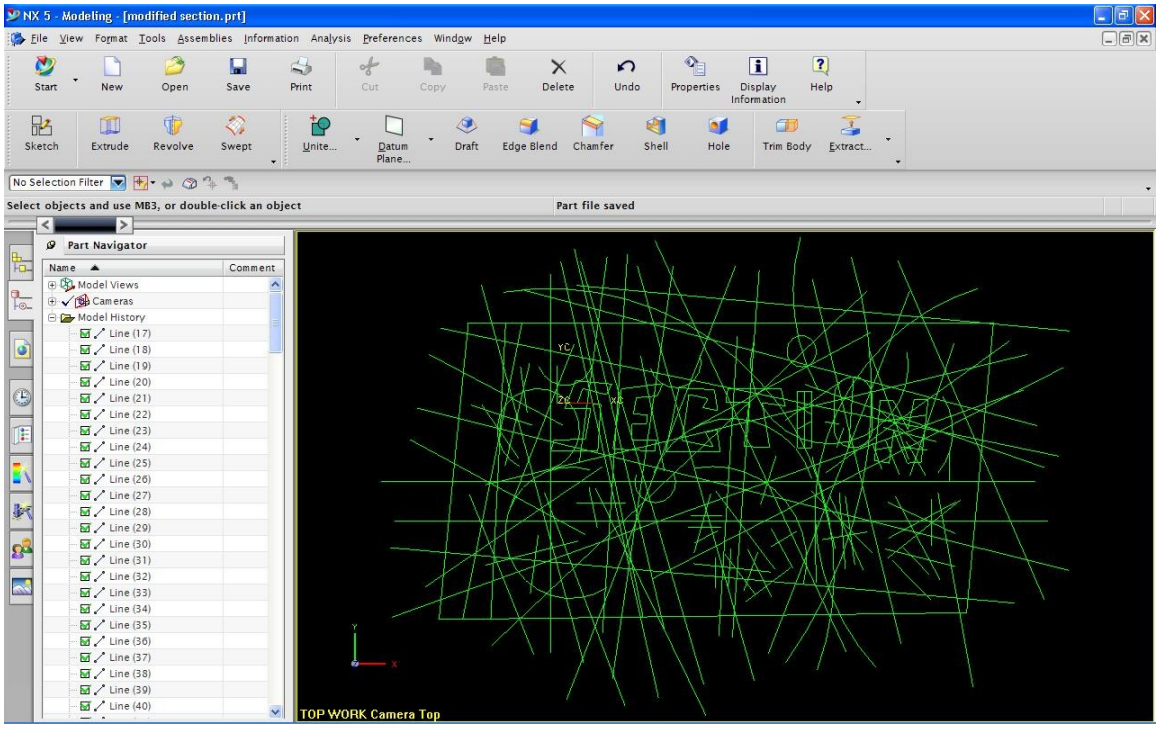

Figure C.3: Example 1: Note the Excess Objects Covering the Geometry

To call our C++ characterization tool from the UGS-NX5 environment the user presses *ctrl+u* and the *Execute User Function* dialog box will appear as shown in Figure C.4 on the next page. The user then has to choose the code from the appropriate location on the computer hard drive and click *ok* to run it. As shown in Figure C.5, the *Object Selection* dialog box will appear and the user has to manually select the objects to be analyzed or click *Select All.* The objects will be selected and highlighted as shown in the Figure.

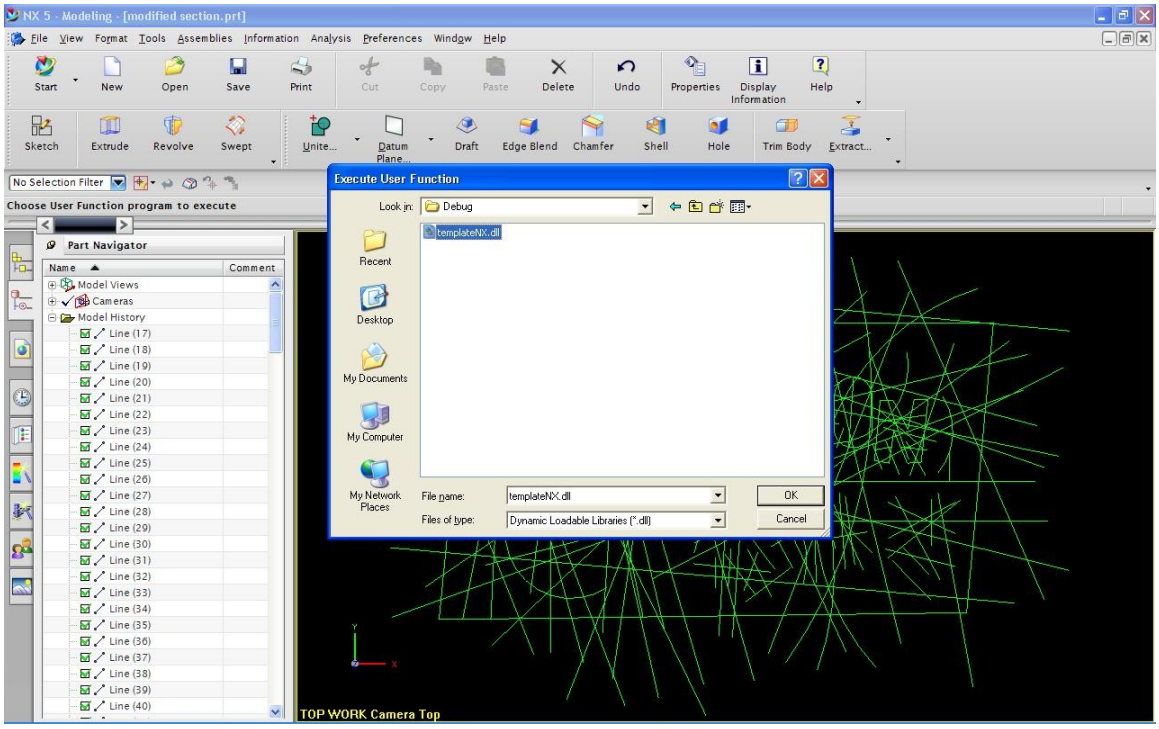

Figure C.4: Press ctrl+u to Call C++ Software Tool

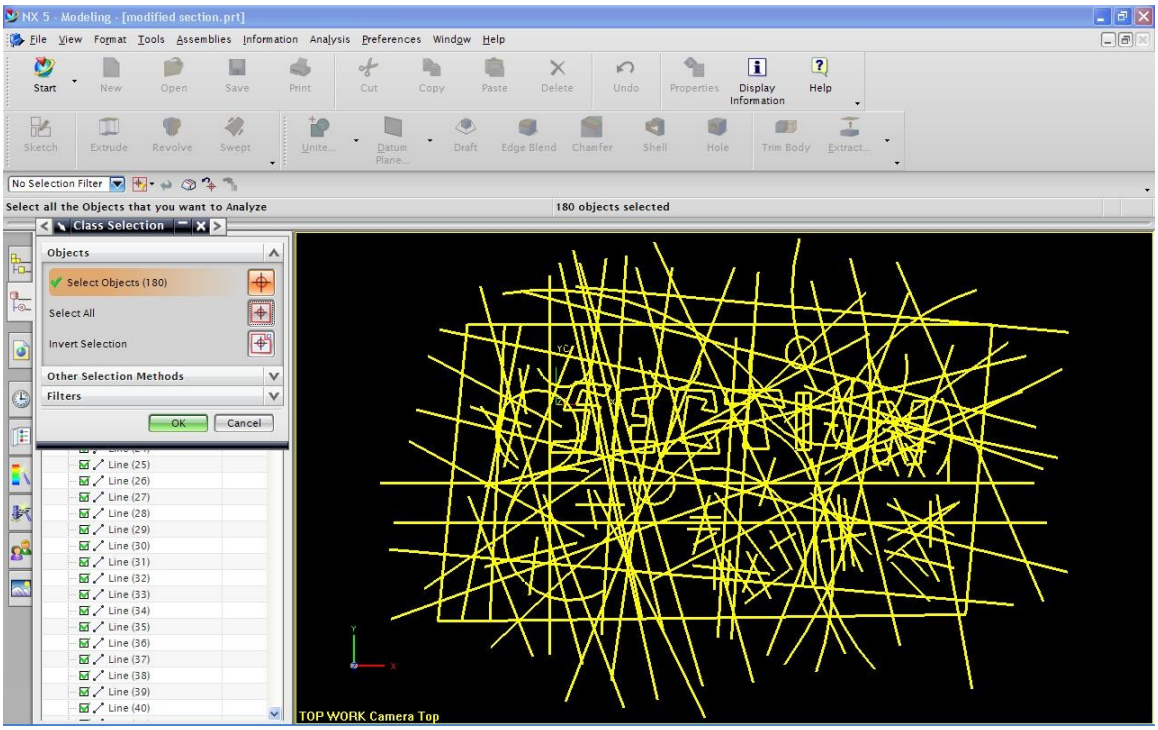

Figure C.5: Selected All Objects from Geometry

Figure C.6 shows the actual geometry after all the excess objects have been removed. The dimensioning and constraining part of the code is automatically started and the *line selection* dialog box will be displayed as also seen in the Figure. There will be independent selection dialog boxes for every type of line that the program can recognize, horizontal, vertical, diagonal, and arcs. The user will choose the appropriate objects of every type that are desired to be dimensioned and constrained and then click *ok*. After this is completed and the user clicks *ok* in the last line selection dialog box, arcs, the program will automatically display the dimensions of the selected objects and create the constraints, as shown in Figure C.7. The tool will also calculate how many constraints are still needed, if any, to make the geometry fully constrained and display a message to the user. This last step completes the use of the characterization tool.

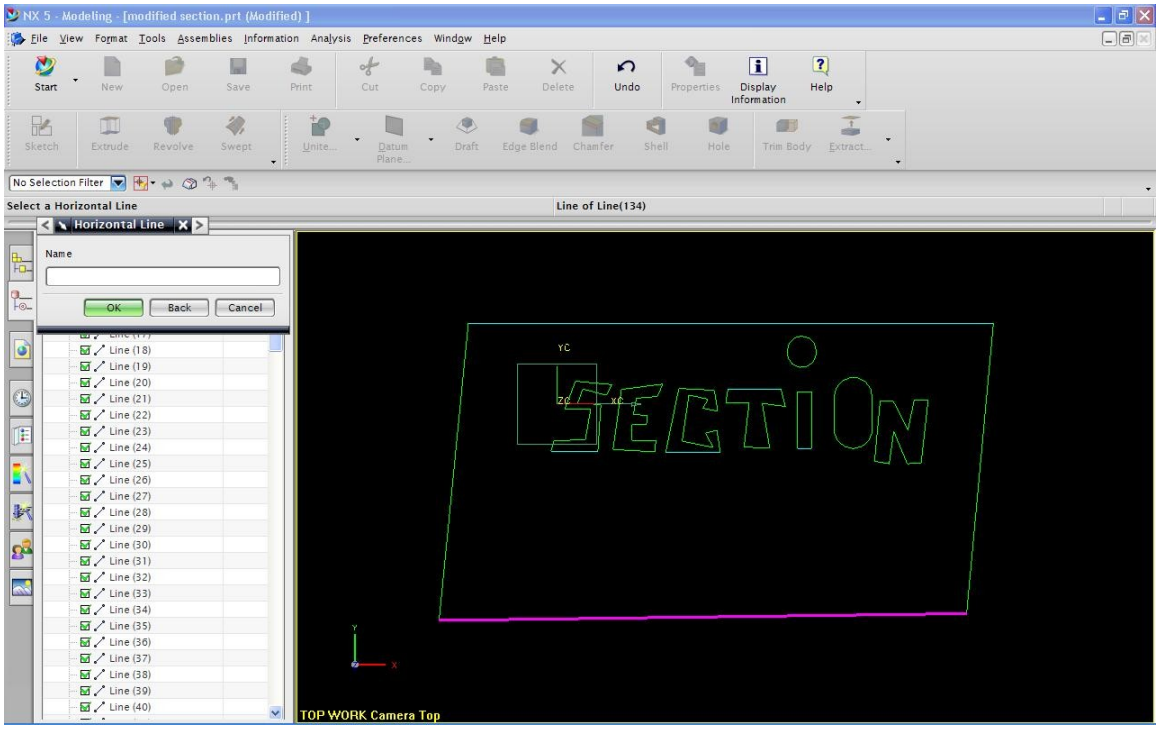

Figure C.6: Excess Objects Removed, Start Manual Selection of Objects to be Dimensioned

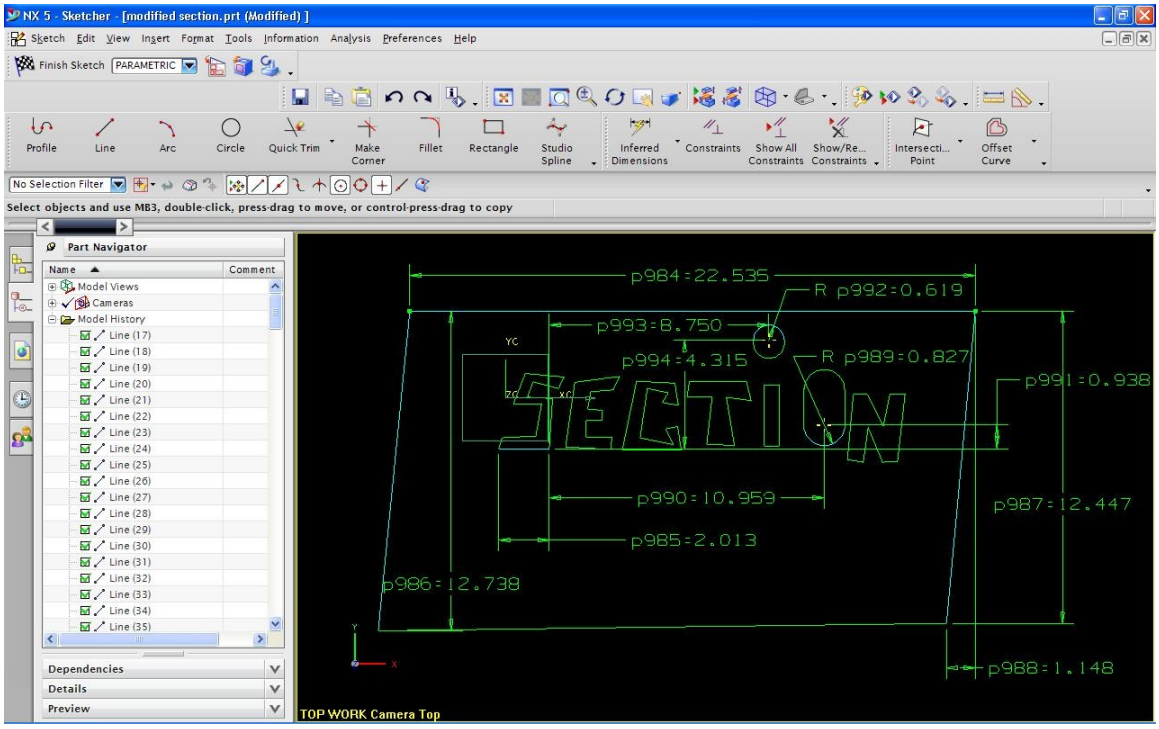

Figure C.7: Final Display with Selected Objects Dimensioned and Constrained

VITAE

Neit Josafat Nieves-Flores was born in Bayamón, Puerto Rico and completed his Bachelor of Science in Mechanical Engineering at the University of Puerto Rico at Mayagüez (UPRM) in December 2000 and in January 2009 he completed his Master of Engineering in Mechanical Engineering within the same university.

He began his professional career in the Automotive Industry as a Manufacturing Engineer for Visteon Corporation working with several automotive systems such as Radiators, Power Steering Systems, Driveshafts and other drivetrain components for Ford vehicles. He then continued on to a Corporate Manufacturing Engineering position at American Axle & Manufacturing, Inc. where he continued his work in the drivetrain area for GM and Chrysler vehicles.

Neit successfully launched the new SmartBar^{tm} Production Line at AAM de Mexico Plant, Guanajuato Gear and Axle, Inc. and then started in a new position as a Product Development Engineer overseeing the Design and Optimization of Multi-Piece Propeller Shafts for Heavy Duty Vehicles.