

Developing Automated Design and Analysis Tools

*A Major Qualifying Project
Submitted to the Faculty of Worcester
Polytechnic Institute
In partial fulfillment of the
requirements for the degree in*

**Bachelor of Science
in Mechanical Engineering
By
Corey Alicchio, Justin Vitiello**

And

**Bachelor of Science
in Robotics Engineering
By
Callum Taylor**

**Advisor:
Pradeep Radhakrishnan**

Date: April 27th 2017

This report represents work of WPI undergraduate students submitted to the faculty as evidence of a degree requirement. WPI routinely publishes these reports on its web site without editorial or peer review. For more information about the projects program at WPI, see <http://www.wpi.edu/Academics/Projects>

Copyright Information

The work presented here is copyrighted by Corey Alicchio, Callum Taylor, Justin Vitiello and
Professor Pradeep Radhakrishnan.

ABSTRACT

Design automation is complex and work is carried out in several stages. Staged development aids in recasting the results into virtual labs that can be used to improve understanding of design and analysis topics. In this project, during the first stage, a tool was developed to generate SolidWorks assemblies from design graphs of gear trains consisting of gear and shaft information. The tool was developed using SolidWorks API. In the second stage, automated creation of bond graphs from SolidWorks assemblies was carried out. The built-in feature recognition system in SolidWorks is used to identify the geometric relationships between different components in the assembly and using grammar rules, a system graph is generated. This system graph is integrated into the automated bond graph generation tool from MQP 2015-16. The final stage involved development of grammar rules and logic to automatically extract state equations from bond graphs. The status of these tools and their testing with students in various courses will be discussed.

TABLE OF CONTENTS

Abstract.....	iii
Table of Figures	vii
Acknowledgments.....	x
Executive summary.....	xi
Introduction.....	1
Literature Review.....	3
A. Gear Train Synthesis	3
B. Bond Graph Modeling.....	4
C. Feature Recognition.....	7
Objectives	8
A. Create a Tool to Automatically Generate A 3D Model of a Gear Train	8
B. Create a Tool to Automatically Obtain a Bond Graph.....	8
C. Automatically Derive State Equations from a Causal Bond Graph	8
Automated Creation of SolidWorks Assemblies	10
A. Read Data from User Entry	11
B. Interpreting Graph XML Data.....	12
C. Generate Gears, Shafts and Bearings	15
I. Modifying Part Dimensions Automatically	15
II. Required Information For Creating A Part.....	15
III. Template Files.....	16
IV. Generate Shafts	18
V. Generate Bearings.....	18

D.	Creating an Assembly.....	19
I.	Adding mates with SolidWorks API.....	20
II.	Results.....	22
	Extracting Bond Graphs from SolidWorks Assemblies	25
A.	Reading Part Data.....	26
B.	Creating System Graphs	28
C.	SolidWorks Identification Results.....	31
	Analyzing Bond graphs.....	33
A.	State equations.....	33
B.	Results	40
C.	Direction of Bond Graphs.....	41
D.	Bond Graph Results.....	43
	Conclusions.....	44
A.	Recommendations and Future Work	45
I.	Automated Gear Generation	45
II.	Extraction of Bond Graph.....	45
III.	State Equations and System Response.....	46
	References.....	47
	Appendix A: Pseudo Code of Gear Generation.....	49
	Appendix B: Student Generated Gear Trains	51
	Appendix C: User Guide for Generating Gears	52
	Appendix D: Rules to Generate System Graphs.....	56
	Appendix E: Rules to generate State Equations	58

A. Rules to Format Graph	58
B. State Equation Ruleset 1	59
C. State Equation Ruleset 2.....	60
D. State Equations Summation at junction.....	62
Authorship.....	64

TABLE OF FIGURES

Figure 1. Overview of the graph-grammar method for generating bond graphs [3]	5
Figure 2. Flow chart of gear generation process	10
Figure 3. Screenshot of User interface for gear generation	11
Figure 4. Entry field for assembly data.....	12
Figure 5: CSV Data file generated by the tool.....	12
Figure 6. Graphical Representation of Meshed Gears with attached parameters	13
Figure 7. XML representation of a node point	13
Figure 8: Example output of reading an XML file	14
Figure 9. Code segment to open a SolidWorks File, modify dimensions and save the part...	15
Figure 10. Base sketch of gear template	16
Figure 11. Feature Manager of gear template	16
Figure 12. Planes created for gear tooth alignment	17
Figure 13. SolidWorks Gears Generated by the tool	18
Figure 14. Shaft template base sketch.....	18
Figure 15. Basic shaft component fully completed.	18
Figure 16. Bearing template base sketch (Sketch is revolved around x-axis)	19
Figure 17. Bearing component completed	19
Figure 18. Process to create a SolidWorks Assembly	20
Figure 19. Code Segment for creating mates within SolidWorks	21
Figure 20. Complete Gear Assembly.....	22
Figure 21. Example of Student generated assemblies	24
Figure 22. Class Survey Results	24

Figure 23: Flow Chart to Create Bond Graphs from SolidWorks Assemblies.....	26
Figure 24: Code segment to define an arc connecting two nodes.....	27
Figure 25: SolidWorks Identification Graph	28
Figure 26: LabelGears Rule	29
Figure 27: SolidWorks Assembly System Graph	30
Figure 28: SolidWorks Assembly Bond Graph	32
Figure 29. Flow chart for generating state equations.....	35
Figure 30. Bond graph rule to add intermediate node	37
Figure 31. Identifying a source of flow in the bond graph	38
Figure 32.Rule to identify Derivative Causality for an I element.....	38
Figure 33. Equal flows at a one junction	39
Figure 34. Equal Efforts at a 0 junction	39
Figure 35. Causality bond graph of mass spring damper system	40
Figure 36. Direction assignment to the nodes.....	42
Figure 37. Rule to apply start direction to system	42
Figure 38. Rule to propagate positive direction through the system	42
Figure 39. Rule to flip arc direction.....	43
Figure 40: LabelGears Rule	56
Figure 41: LabelShafts Rule	56
Figure 42: RemoveCoincident Rule.....	57
Figure 43: RemoveGearmate Rule.....	57
Figure 44: RemoveGear0DOF Rule	57
Figure 45. Rule to add intermediate nodes, configuration 1	58

Figure 46. Rule to add intermediate nodes, configuration 2.....	58
Figure 47. Rule to add intermediate nodes, configuration 3.....	58
Figure 48. Rule to add intermediate nodes, configuration 4.....	58
Figure 49. Rule to identify integral causality.....	59
Figure 50. Rule to identify integral causality of C element.....	59
Figure 51. Rule to identify source of flow	59
Figure 52. Rule to identify a source of effort.....	59
Figure 53. Rule to identify derivative causality of C element	60
Figure 54. Rule to identify derivative causality of I element	60
Figure 55. Rule to identify derivative causality of R element	60
Figure 56. Rule to identify integral causality of R element.....	60
Figure 57. Rule to identify gyrator node with known effort.....	60
Figure 58. Rule to identify gyrator node with known flow	61
Figure 59. Rule to identify transformer node with known effort.....	61
Figure 60. Rule to identify 0 junction with a known source of effort	61
Figure 61. Rule to identify 1 junction with a known source of flow	61
Figure 62. Rule to indicate if a node has been used in a summation equation of a 1 junction	62
Figure 63. Rule to indicate if a node has been used in a summation equation of a 0 junction	62
Figure 64. Rule to mark if node can be summed at 0 junction	62
Figure 65. Rule to mark if node can be summed at 0 junction	62
Figure 66. Rule to apply the summation equation to at node at a 0 junction.....	62
Figure 67. Rule to apply the summation equation to at node at a 1 junction.....	63

ACKNOWLEDGMENTS

We would like to acknowledge the students in D2017 ME4320 Advanced Engineering Design course at Worcester Polytechnic Institute for testing and providing feedback on the gear generation program. We would also like to thank Professor Pradeep Radhakrishnan for his support, and guidance on this project. Finally, we would like to acknowledge Worcester Polytechnic Institute for providing us the opportunity to pursue this project.

EXECUTIVE SUMMARY

This project researched and developed solutions related to the automated design of gear-trains. Investigations were conducted into the creation of 3D models of gear-train parts and assemblies, generation of dynamic models and the extraction of differential equations from those dynamic models. Automated solutions were developed from scratch to more accurately create systems in a shorter amount of time and with less human input when compared to manual iterative design. The project was carried out in three stages.

The first stage involved developing a tool that automatically generated a 3D model of a gear train. This was accomplished by using the SolidWorks API to script the creation of gears, shafts, bearings, and the total assembly. The required input parameters for the creation of a gears in a gear train are Gear Type, Number of Teeth, Pitch, Face Width, Bore Diameter, Pitch Cone Angle, Helix Angle, and Material. For shafts, diameter, length, and material are required. To create the assembly, 3D coordinates, axis of rotation of each component, meshing information, and shaft placement are required to properly place each part. Bearings can be chosen to be automatically created, and are placed at both ends of each shaft in the assembly. The completed assembly accurately represents the change in rotational velocity from the input to the output of the system. The input parameters can be entered into a GUI by the user. As the actual gear-train design solution would exist as a graph in an XML format in the overall automation scheme, there is also an option in the GUI to import this data from an XML file.

Students in an Advanced Engineering Design course were asked to use this tool to complete a gear train assignment. They were tasked with creating gear trains given certain constraints, and then to generate the 3D model using this tool to verify its

accuracy. Students provided feedback on the program as well as suggesting areas of improvement.

The second stage of the project involved the automatic extraction of bond graphs from a SolidWorks assembly. A bond graph representation of the system provides a standard way of analyzing the dynamic response of a system. Using graph grammar rules to identify parts and mates within a SolidWorks assembly the system's identity graph is generated. The identity graph is then saved to a GraphSynth-compatible XML file. Various geometric and material details are also saved in the process.

The final stage of the project involved enhancing the existing Automated Virtual Lab for bond graphs to extract state equations from causal graphs. State equations are generated by following standard causality and state variable procedures as defined by bond graph methodology. The program starts with a bond graph and assigns causality to each bond. State variables are identified, and state equations are generated by finding all the efforts and flows for each bond. The resulting equations are then displayed to the user, who can then solve and graph these equations to analyze the dynamic response of a system.

Each of these stages can be used to aid students in the process of mechanical design and analysis. By being able to generate 3D models, the accompanying bond graphs and state equations, students can better understand the process of gear train design and bond graph analysis.

INTRODUCTION

Gear trains are used in many mechanical systems, generally comprised of multiple meshing gears, shafts, and bearings, to transfer, increase, or reduce rotational velocity and torque. When designing these systems, the designer must take into account the stresses and dynamic responses of the system and each individual part. This is an iterative process, and currently has to be done manually, which is very time consuming.

This process is being automated through the creation of software tools to automatically generate and analyze gear trains. These tools, being created by this team as well as others, would allow users to quickly create and analyze gear train systems. Users would input the design requirements of the system, and the program would find the optimal solution and then generate the resulting static and dynamic responses. A 3D CAD model would be created to visualize the system, a bond graph would be created from the 3D CAD model, and then bond graph techniques would be used to find and solve for the state variables within the system. By showing the process of gear generation from start to finish, this tool can be used as an educational aid to easily generate and verify system designs.

These tools are useful to students when designing systems as not every student has the same background in gear design. A tool that assisted in the generation of gear trains would allow those students to include useful gear systems into larger projects without having to know the specific mechanics of the system.

Additionally, the individual components of the tool can be extracted and used as teaching aids. The portion which generates the gear parameters can be used to show students how to generate gears given input and output speeds. The 3D model generation portion can be used by

students to verify gear designs they have created by hand, speeding up the process of iterative gear train design. The bond graph portion can show what a proper bond graph of a gear system would look like, as well as demonstrate how state equations are formed. All of these systems can be used as a way to teach the process as well as a tool to verify and correct student's own solutions.

This project is specifically focusing on the design activities of generating 3D models of gear trains, generating bond graphs from the assemblies, and generating the corresponding state equations for the bond graphs. Generating a 3D model is done by taking gear data as input and using the SolidWorks API to create a corresponding assembly. Creating bond graphs is done by using the SolidWorks API to generate an identification graph of the parts and mates within a system. Graph grammar rules are then used to identify the parts within the system and generate a system graph of the assembly. An existing program then imports this system graph to create the bond graph of the gear train. A combination of grammar rules and a C# program generates state equations to assign causality to each element and use bond graph techniques to generate state equations.

The following section will provide an overview of the related literature on this project. Following that section, we explain the three objectives created for this project. We will then go into detail for how each objective was achieved, explaining the development process and the final results. Then we discuss our final conclusions and recommendations for future work for our project.

LITERATURE REVIEW

This section presents articles related to the main design activities of this project. The articles focus on various aspects of the project as well as previous work that has been done with gear train automation.

A. Gear Train Synthesis

The design of gears and related components such as shafts bearings and housings, are tasks that rely on very well defined design procedures [3]. These methods are in accordance to many American Gear Manufacturers Association (AGMA) standards and many design equations can be determined automatically. Typical design flow of a gear may involve many iterations, first determining loading conditions and then calculating internal stresses and ultimately arriving at a safety factor for a component. This can take a significant amount of time, depending on how accurate the analysis is. It would be ideal if this process could be at least partially automated, leading to the ability to find the optimal gear design much faster.

One example of automated design was the synthesis of a two-stage helical gear reducer researched by Tudose et al [1]. Using evolutionary based algorithms, the optimal configurations for gears shafts and bearings were achieved. These algorithms generated many possible solutions that were then analyzed based on how well it fit within the given parameters, attempting to minimize the mass of the overall gear system. The most successful solutions could then be modified and reevaluated for the next iteration.

The article "A Method and Software Tool for Automated Gearbox Synthesis" by Lin et al. [4] presents another method for computer based gearbox synthesis given certain design specifications [4]. The program being used is an experimental add-on to RomaxDesigner, a

software program from Romax Technology for driveline design [5]. The gears are modeled without teeth and having an assumed synchronizer. The paper outlines nine rules for gearbox creation. The program can create multiple gearboxes which satisfy the prerequisite conditions, so additional rules are applied to minimize mass and maximize gear spacing. This helps to prevent overly complex designs. The program also identifies and prevents unwanted collisions between gears, shafts, and the bounding box. The paper concludes that roughly half of the generated gearboxes successfully fit all of the conditions presented. Viable designs varied in the number of stages as well as in the placement of gears. Limitations listed in the paper are that the program cannot create shafts of varying lengths, gear widths are chosen at random, and it is inaccurate for large complex gearboxes. The design does not include any bearings or synchronizers which are usually present in real gearboxes.

Another technique that is well suited for gear design is using graph grammars to design a system. This method allows for large, high level systems to be manipulated with generic and concise rules [1]. A seed graph was used to create an expanding tree of different component combinations. The grammar rules then are applied to create topology and create a figure from user input.

B. Bond Graph Modeling

Grammar based tools have also been applied to aid in the creation of functional schematics [2]. Numerous possible design schematics for epicyclical gear trains are automatically generated allowing the user to view many possibilities at once rather than spending excessive time generating these concepts manually. Grammar rules provide a rigorous method for the creation of design schematics that is consistent and capable of handling complex systems [2].

The design work in the studies mentioned above optimize static systems using stress relationships and analysis. In order to obtain the full response of a system it is also necessary to look at its dynamic response. Analysis of non-steady state motion reveals more details about the system. Bond graph modeling is one common technique to accomplish this task. One process for bond graph synthesis is detailed in the paper 2016 paper by Grande et al., a previous MQP at WPI [3]. This paper presents a graph grammar based approach to creating bond graphs. Each component of a system is represented as a graph node and nodes are connected by arcs as shown in Figure 1. Each node is given standard labels which identify how each part relates to bond graph notation, such as "Include_Stiffness", "Include_Friction" or "Torque_Input". This process uses a program called GraphSynth to create bond graphs through the use of graph grammar rules. These rules were written to interpret the labels applied to nodes, such as creating a C element for "Included_Stiffness" or an "I" element for the label "Include_Inertia". These rule sets are able to create the bond graph, simplify flow through junctions, and apply directions to bonds.

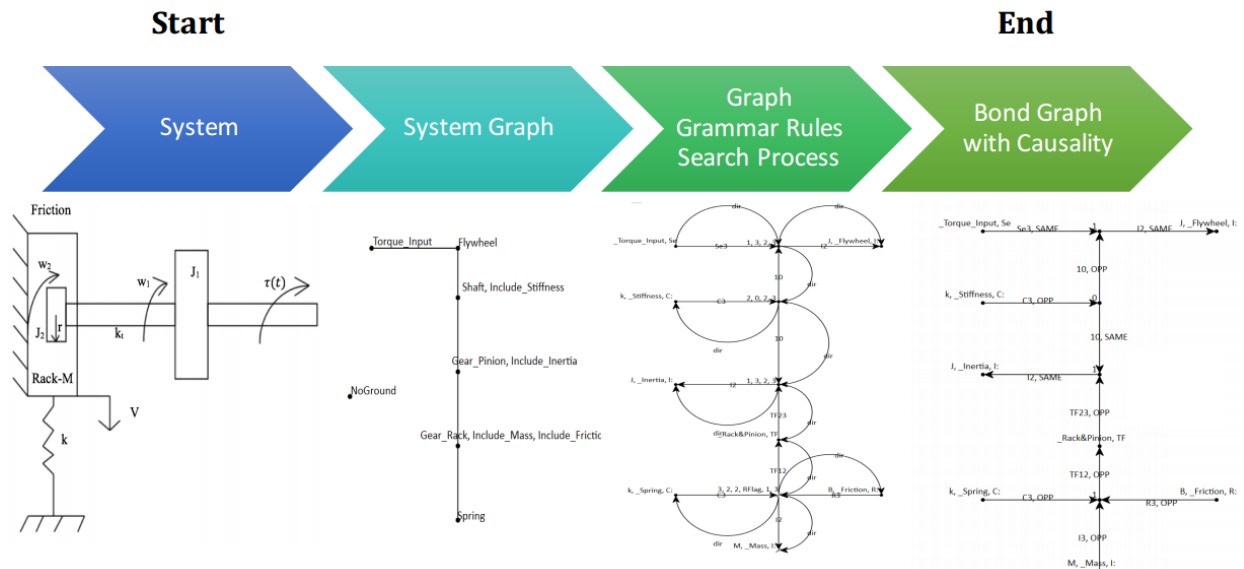


Figure 1. Overview of the graph-grammar method for generating bond graphs [3]

A second paper, "*An Automated Virtual Lab for Bond Graph Based Dynamics Modeling Using Graph Grammars and Tree Search*" [4], builds off of the previous to create a program in which bond graphs can be created. By integrating the graph-grammar based bond graph creation into the program, students can draw a system graph which will be automatically converted to an un-simplified and then a simplified bond graph. The user created bond graph is compared to the automated solution and then discrepancies are displayed to the user. The motivation behind this work was to provide immediate feedback to students learning bond graph modeling techniques.

Other software packages are available that generate bond graphs directly. 20Sim is a commercially available package that is all inclusive in terms of modeling [5]. It allows a system to be fully defined, with all conditions a user requires. It will then create and present the finalized bond graph, in addition to forming the state equations. The main drawback to such a service is the lack of transparency. This software lacks the ability to illustrate the steps taken to arrive at the final graph. While it is possible to compare a written solution to the end result, it becomes difficult to identify errors. The simplification rules of bond graphs make it possible to have two systems that are fundamentally equivalent, but contain different elements. This is where error checking becomes most difficult and software like 20Sim becomes less helpful. Automated Virtual Lab [4] was developed to solve this problem as it would allow for more dynamic comparisons, showing step by step solutions if required. The final step in bond graph modeling, generating state equations, is the most difficult for both students and computers. This is due to the complex relationships throughout the system. A way to easily solve and explain the equation generation process is not currently available, and should be addressed.

C. Feature Recognition

Most engineering students are familiar with solid modeling and creating 3D parts, making it a good starting point for modeling dynamic systems. Letting a student visualize a fully functional 3D assembly and its corresponding bond graph gives much more clarity than starting a problem with a bond graph. With the tools mentioned above [3], [4], it is only possible to apply bond graph techniques and analyze a system once that system has been expressed the specific format required by each tool [6].

A modeled part may be broken into sub-features through the use of convex decomposition, creating a seed graph of the part [7]. This process uses grammar rules to identify regions of a part that may be machined in a single operation. This method is limited to geometrical identification only, and does not allow for the estimation of mechanical properties such as stiffness or damping properties, which is required for a bond graph. A higher level of recognition, which would group parts by a generalized classification system, such as shafts, gears or bearings, may be sufficient to derive a bond graph model of the part. In addition to faster analysis, this may avoid the need to have a user verify the decomposition of the part which is required if analyzing each part for geometrical features [7].

The next section introduces the objectives that we developed based on limitations we found in previous literature.

OBJECTIVES

This section explains the three objectives that were determined and the requirements that the objectives must meet to be considered successful.

A. Create a Tool to Automatically Generate A 3D Model of a Gear Train

The tool should operate significantly faster than manually creating an assembly and be easy for a user just learning about gear design to use. It should be able to create multiple types of gears and generate shafts and bearings. The modeling of all parts should be highly accurate and generate fully functional assemblies.

B. Create a Tool to Automatically Obtain a Bond Graph

The tool should only require the user to select a single assembly file, then automatically gather all information from the file. This data should include mass and material properties. It must also be able to analyze mate configurations to determine how components are connected. The final output of the tool must match the format of the input for the bond graph tool developed in last year's MQP [3], [4].

C. Automatically Derive State Equations from a Causal Bond Graph

This should build upon the program previously developed by the preceding MQP [3], [4]. The starting point for this is a causal bond graph. State variables must be identified and then the equations for each of these are to be derived. It should also utilize graph grammar rules to aid in solving the equations. The final output should be the completed state equations that can be directly solved with a program such as MATLAB.

The next chapter will discuss how objective A was accomplished, describing the process and results of our work.

AUTOMATED CREATION OF SOLIDWORKS ASSEMBLIES

A tool to automatically create gear assemblies is important in automating the overall design process, as it will allow a user to easily generate gears with very little knowledge or experience

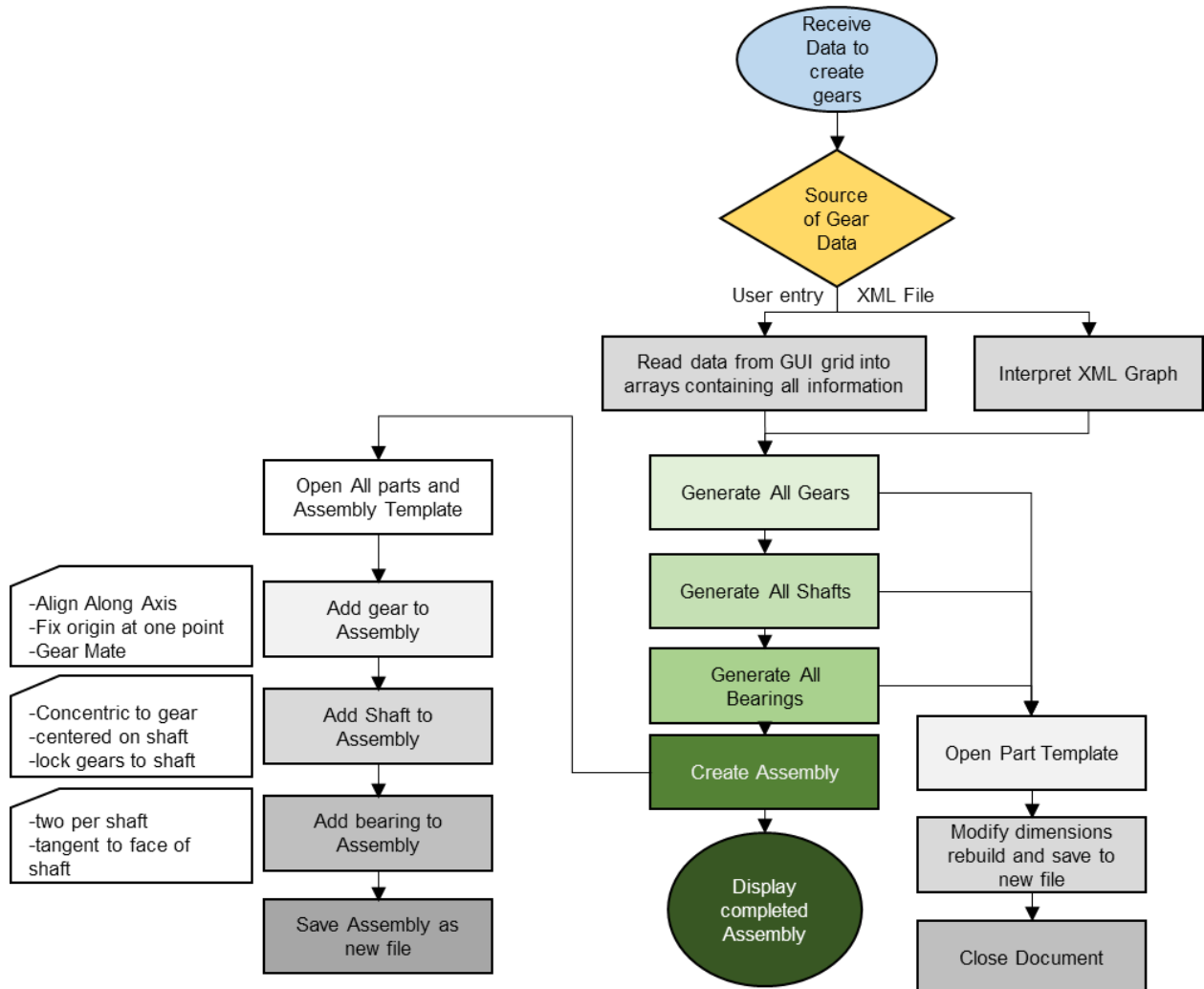


Figure 2. Flow chart of gear generation process

with gear design. Many iterations of designs can be created and visualized with minimal input from a user. The flow chart in Figure 2 describes the process this tool follows to gather data and turn that into a complete assembly model.

A. Read Data from User Entry

To generate gears from a user’s specifications, a graphical user interface (GUI) was developed to allow all of the data to be entered at once, and then generate the gear assembly by pressing a single button. Shown in Figure 3 is the main screen of the program. The data regarding each gear is entered into a data grid and then the data for each gear is stored in a struct type. The first tab in the GUI is used to specify part details, and the second tab (Figure 4. Entry field for assembly data allows a user to specify assembly details, including coordinates, orientation of parts and which gears and shafts are connected.

The side panel on the left allows the user to change several settings for gear generation. The Output folder specifies the directory that all parts and files should be stored. Units may be switched between imperial (Inches) and metric (millimeters). It is also possible to prevent the generation of either parts or the assembly, which can reduce waiting times if an assembly is not required, or the parts already exist in the directory. Once all data has been entered the user presses “Create” to automatically open SolidWorks to create the specified parts and assembly. A data file is created that stores the data into a comma separated value (CSV) file, which can be loaded by the GUI program later.

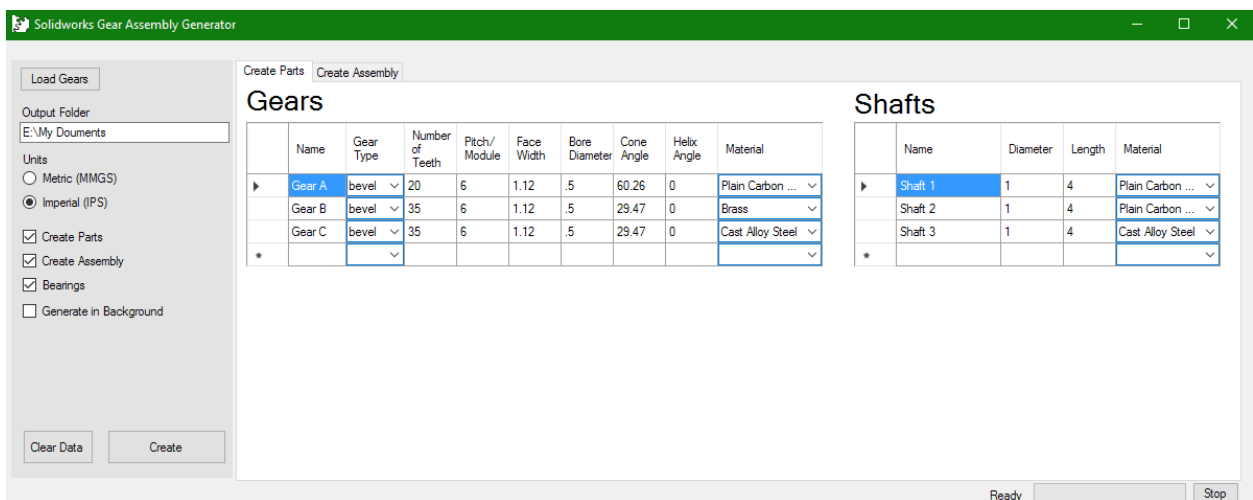


Figure 3. Screenshot of User interface for gear generation

Create Parts		Create Assembly						
Assembly								
	Gear	X coord	Y coord	Z coord	Axis of Rotation	Idler	Connected Gear	Connected Shaft
▶	Gear A	0	0	0	X	<input type="checkbox"/>	Gear A	Shaft 1
	Gear B	-1.65	0	2.92	Z	<input type="checkbox"/>	Gear B	Shaft 2
	Gear C	1.65	0	2.92	Z	<input type="checkbox"/>	Gear C	Shaft 3

Figure 4. Entry field for assembly data

Gear Data		4/22/2017 23:10						
Name	Gear Type	Number of Teeth	Pitch/ Module	Face Width	Bore Diameter	Cone Angle	Helix Angle	Material
pinion	spur	17	39	0.1968503	0.125	0	0	Plain Carbon Steel
Gear	spur	64	39	0.2362204	0.19685039	0	0	Plain Carbon Steel
Shaft Data								
Name	Diameter	Length	Material					
Assembly Data								
Gear	X coord	Y coord	Z coord	Axis of Rotation	Idler	Connected Gear	Connected Shaft	
pinion	1	3	5	x	FALSE	Gear		
Gear	12	0.4	2	y	FALSE	Pinion		

Figure 5: CSV Data file generated by the tool

B. Interpreting Graph XML Data

This MQP project was done in conjunction with another project that aimed at automating the generation of parameters for a gear train. That project optimizes the selection of gears based on stresses, operating conditions and lifecycle analysis and the resulting parameters are saved into a specially formatted XML file. This file contains all relevant information about the gears and how they are connected and assembled. This project takes these XML files and interprets them to create CAD files.

A program was developed to take a formatted XML file, read it and then generate the SolidWorks model based on the read data. The XML file contains a special header generated by GraphSynth, a list of arcs and a list of nodes. Each node refers to either a part or a single characteristic of a part, and arcs connect these nodes together to form a graph. As shown below in the graphical representation (Figure 6), Gear C is connected to 9 different nodes specifying its characteristics, (such as safety factor, gear type, pitch diameter, etc..), one of the connecting

nodes is a “connection” node that is shared with Gear D to indicate that they are meshed with each other.

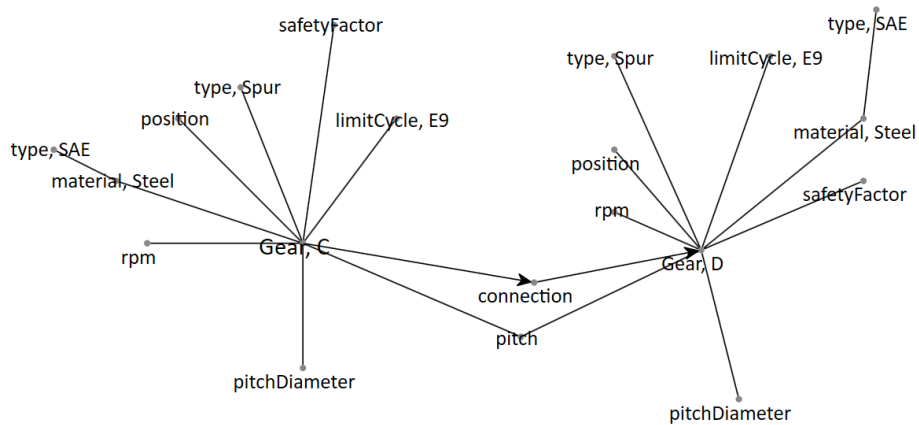


Figure 6. Graphical Representation of Meshed Gears with attached parameters

```

</node>
<node
  xsi:type="ruleNode">
  <name>n43</name>
  <localLabels>
    <string>limitCycle</string>
  </localLabels>
  <localVariables>
    <double>1000000000</double>
  </localVariables>
  <X>1608</X>
  <Y>336</Y>
  <Z>0</Z>
  <negateLabels />
  <NotExist>>false</NotExist>
  <containsAllLocalLabels>>false</containsAllLocalLabels>
  <TargetType />
  <strictDegreeMatch>>false</strictDegreeMatch>
</node>

```

Figure 7. XML representation of a node point

The program extracts and stores all the useful information in various lists for generating the assembled. Then the characteristics of the gear train are displayed in an easily readable format to an output window. It also displays a list of errors and or warnings present in the XML file. There are three types of error message, a warning where the program changes nothing (ex: Detecting that two gears are too close together), a warning where the program changes a value to a default

(ex: Detecting that there is no set shaft length), and a warning message that a key piece of information is missing and the program will not be able to build a SolidWorks model correctly (ex: Detecting that a gear does not have a specified pitch value). Figure 8 below displays an example of the output window after reading an XML file. If desired, the user can hit enter in this window to automatically open SolidWorks which will then generate the gear train specified by the XML.

```
-----  
Z Rotation  
-----  
No Connection Shaft  
Gear F is located at X: 2.99, Y: 3.14, Z: 3.7  
Material: 1018, Pitch:8, Pitch Diameter:5, Type: spur  
F is connected to Shaft -1  
F is connected to Gear  
  
No Connection Shaft  
Gear E is located at X: 3.65, Y: 0, Z: 3.7  
Material: 1018, Pitch:8, Pitch Diameter:2, Type: spur  
E is connected to Shaft -1  
E is connected to Gear  
  
-----  
Z Rotation  
-----  
Gear C is too close to Gear D  
Distance: 3.15158690186388, Combined Radius: 3.2  
Gear D is too close to Gear C  
Distance: 3.15158690186388, Combined Radius: 3.2  
Gear F is too close to Gear E  
Distance: 3.20861340768875, Combined Radius: 3.5  
Gear E is too close to Gear F  
Distance: 3.20861340768875, Combined Radius: 3.5
```

Figure 8: Example output of reading an XML file

C. Generate Gears, Shafts and Bearings

There are many steps involved in creating each part for the assembly. The process is broken down into single one CAD part at a time, generating all gears shafts and bearings sequentially.

I. **Modifying Part Dimensions Automatically**

With all the gear data extracted in the previous section this data may now be sent into SolidWorks. An application programming interface (API) built into SolidWorks is utilized via a visual studio C# program to manipulate parts in SolidWorks. The snippet of code shown below (Figure 9) is an example of how the API automatically opens a SolidWorks part, and sets sketch dimensions within the part (the Bore diameter, Pitch and the number of teeth). The rebuild command is then sent, and the part is saved with a filename given by the user.

```
SwDoc = ((ModelDoc2)(swApp.OpenDoc6(FileLocation + Gear_Template, 1, 0, "", ref longstatus, ref longwarnings)));  
  
((Dimension)(swDoc.Parameter("Bore@HoldingSke"))).SystemValue = gear.ShaftDia;  
((Dimension)(swDoc.Parameter("Pitch@HoldingSke"))).SystemValue = gear.Pd;  
((Dimension)(swDoc.Parameter("Num_teeth@HoldingSke"))).SystemValue = gear.Numteeth * (3.14159 / 180);  
  
swDoc.Rebuild(2);  
Longstatus = swDoc.SaveAs3(userLocation + gear.Name + ".sldprt", 0, 2);
```

Figure 9. Code segment to open a SolidWorks File, modify dimensions and save the part

II. **Required Information For Creating A Part**

There is a minimum amount of information that must be known about each gear prior to generation. The required information includes: Pitch, Number of teeth, face width, bore diameter, cone angle and helix angle. If the part is not a bevel or helical gear the cone and helix angles are ignored. There are several other parameters that can be set according to gear design standards which include the addendum, dedendum and pressure angle [8]. The pressure angle is

set to 20 degrees for all gears, the addendum is set equal to the reciprocal of the diametral pitch and the dedendum is equal to 1.157 times the reciprocal of the diametral pitch. These are values are in accordance to gear design standards [8], [9].

III. Template Files

To generate a gear, a SolidWorks part has been created to act as a template file. These files contain predefined dimensions within the template file can be changed to update the model according to the user's specifications. These dimensions are held in a base sketch. All other features inserted into the part such as revolves, cuts, extrusions and patterns are dependent on the base sketch dimensions. This allows the creation of any gear arrangement from a single template. Figure 10 show the base sketch for the template, and Figure 11 shows the feature manager for each feature used to create the gear.

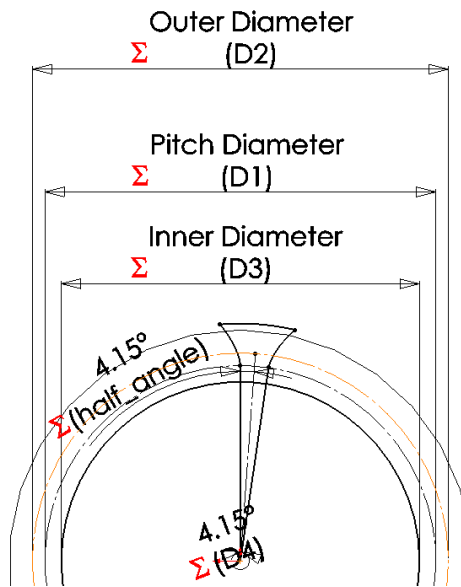


Figure 10. Base sketch of gear template

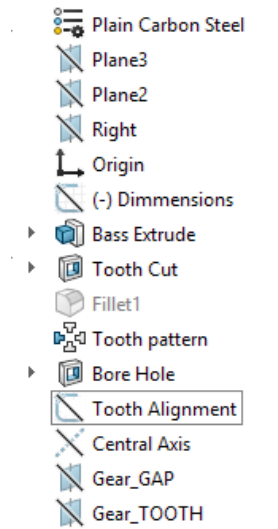


Figure 11. Feature Manager of gear template

Features used in creating the gears:

1. *Base extrusion*: A cylinder the size of the outer diameter + the dedendum length

2. *Single tooth cut*: The first tooth cut is modeled using parametric equations for an involute profile. This ensure that the gear mesh will be extremely accurate
3. *Circular pattern*: The tooth cut is then patterned in a circle equal to the number of teeth
4. *Bore diameter cut*: An extruded cut that is the diameter of the bore
5. *Reference geometry*: To assist in assembling the gears, three reference geometry features are inserted; a central axis of the gear, a plane passing through the center of a gear tooth and a plane passing through the space between two teeth (shown in Figure 12).

Each type of gear (spur, helical, bevel, worm, worm wheel) has a separate template file. They are modified in the same fashion as spur gears. The difference is the additional parameters; for bevel gears: cone angle, helical gears: helix angle, worm/worm wheel: number of threads on worm wheel. Examples of different types of gears are shown in Figure 13.

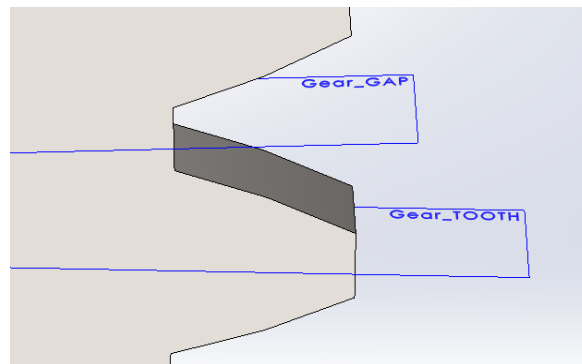


Figure 12. Planes created for gear tooth alignment



Figure 13. SolidWorks Gears Generated by the tool. (Helical, Bevel, spur, Worm, Worm wheel)

IV. Generate Shafts

Simple cylindrical shafts are utilized in this program, requiring information only about its length, diameter and material. A template file is also used in a similar fashion to the spur gear. The template file (shown in Figure 14 and Figure 15) is opened, the length and diameter dimensions are modified and then the part is saved as a new part.

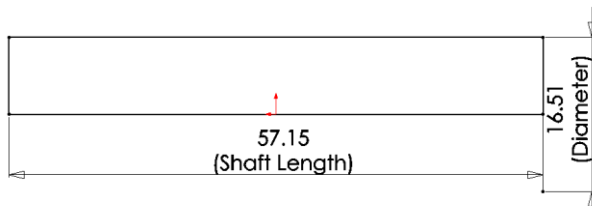


Figure 14. Shaft template base sketch

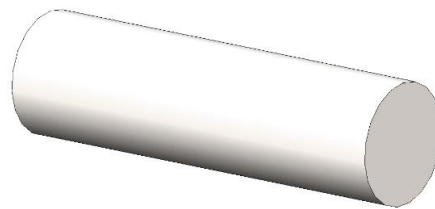


Figure 15. Basic shaft component fully completed.

V. Generate Bearings

If bearings have been chosen for assembly, they are generated next. For this project, a universal ball bearing part is used. Its dimensions, shown in Figure 16, are scaled according to

the size of the shaft it attaches to. All other dimensions (outer diameter, ball size, thickness) are proportional to the shaft diameter it is attaches to. The template for the part is opened, then the dimensions are set, shown in Figure 16, and the part is saved as a new part.

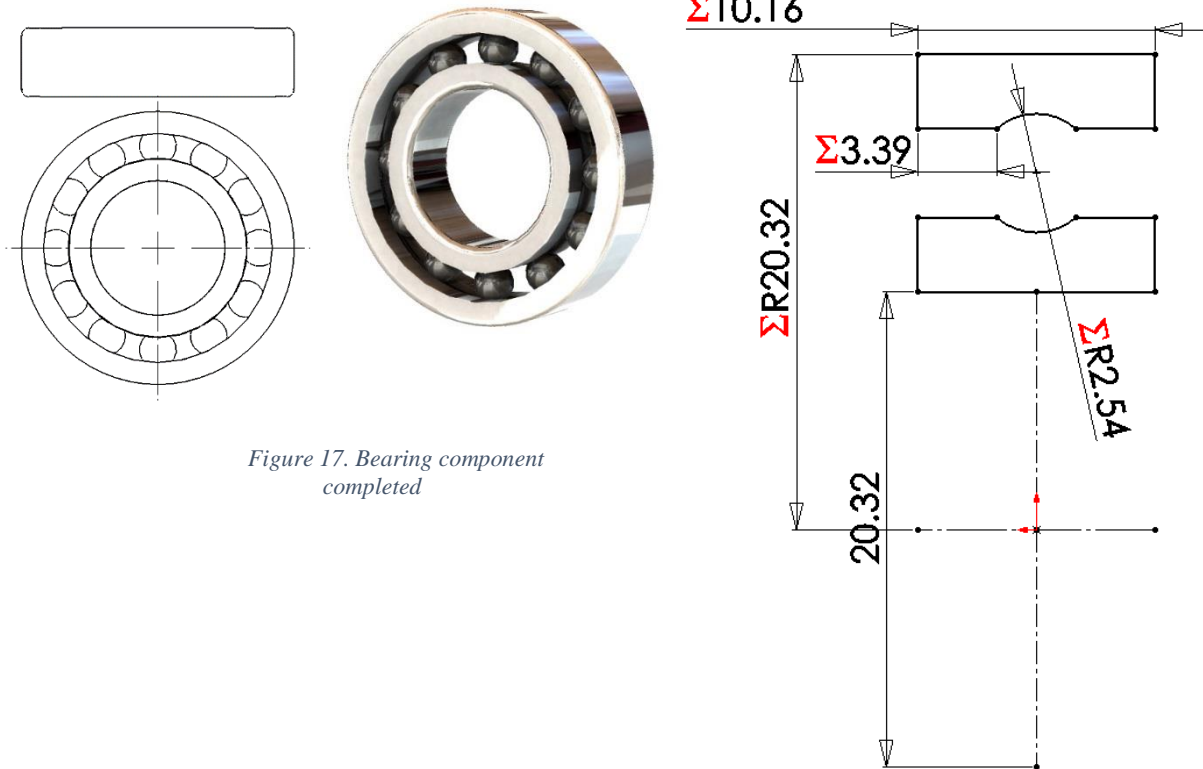


Figure 17. Bearing component completed

Figure 16. Bearing template base sketch (Sketch is revolved around x-axis)

D. Creating an Assembly

When all parts are generated and saved, they are then added together into a new Assembly file. To add a part into an assembly using a C# macro in SolidWorks, the part file must be open. To minimize the processing power required, each part is opened, added to the assembly and closed. The flow chart in Figure 18 shows the process to add each gear, shaft and bearing into the assembly. First a point is added to locate the gear within the assembly, then the gear is added and

fixed in the assembly. Shafts and bearings are then added afterwards mating them to the gears they attach to.

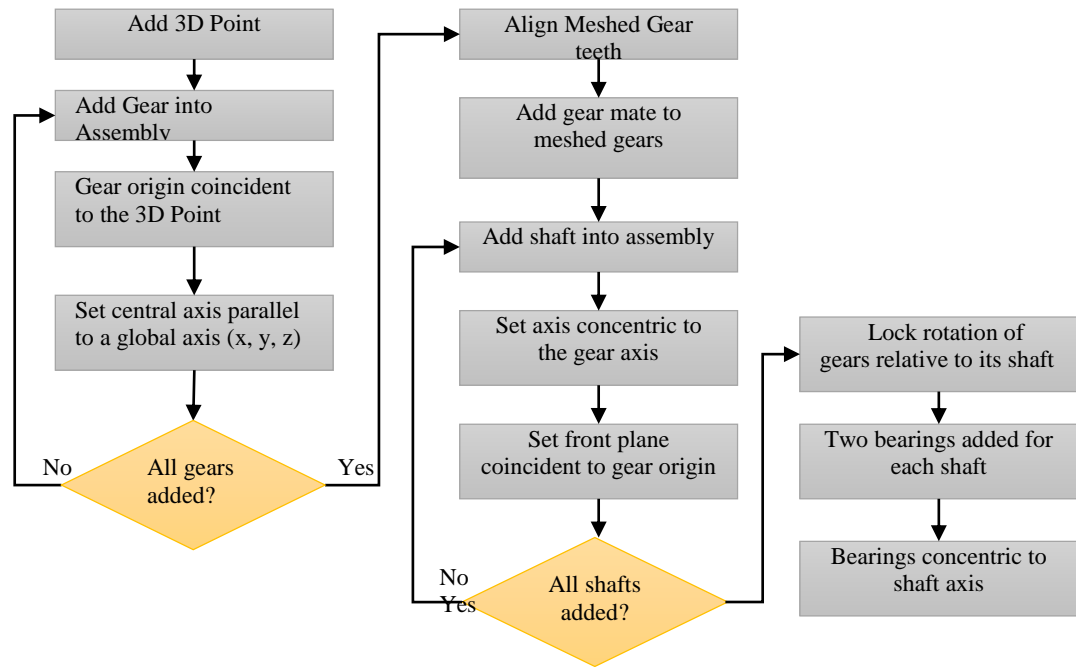


Figure 18. Process to create a SolidWorks Assembly

I. Adding mates with SolidWorks API

Many mates must be created in the program while defining the assembly. Mates define how parts will move relative to each other in the assembly, essentially limiting degrees of freedom. To add a mate, a face, line, plane or point of two parts are selected and then the mate will limit the motion of the selected features. This process is automated by selecting the faces to be mated through the API and applying the proper SolidWorks mate as required by the system. The code segment in Figure 19 demonstrates how the API creates the mates, by selecting two points (selectByID2) then adding the mate (AddMate5). All parameters for a mate are specified within this line, including any distances or flipped configurations.


```

boolstatus = swDoc.Extension.SelectByID2("Point"+(i+23)+"@Shaft coords sketch", "EXTSKETCHPOINT",
Gear[i].x, Gear[i].y, Gear[i].z, true, 1, null, 0);

boolstatus = swDoc.Extension.SelectByID2("Point1@Origin@" + Gear[i].Name + "-1@Assembly_Template",
"EXTSKETCHPOINT", 0, 0, 0, true, 1, null, 0);

myMate = null;
swAssembly = ((AssemblyDoc)(swDoc));
myMate = ((Mate2)(swAssembly.AddMate5(0, 0, false, 0, 0, 0, 0, 0, 0, 0, false, false, 0, out
longstatus)));

```

Figure 19. Code Segment for creating mates within SolidWorks

a. *Add all Gears into an Assembly*

The positions and orientation of gears are specified in x, y, z coordinates and the orientation is set along one global axis (x, y, z). To locate the parts within the assembly, a 3D sketch is created with a point at the coordinate of the origin of the gear. A single gear is then added into the assembly, then a *coincident* mate between its origin and the corresponding point in the 3D sketch. Orientation is then set by aligning the gears central axis parallel to the global axis in the assembly.

b. *Add Gear Mates*

Gear mates are then added between the meshed gears. An attempt to correctly align the teeth is first made by selecting the planes running through the gear tooth of one gear and the gap between the tooth of the mating gear and setting them *Coincident*. This step is important in making the gears align properly, as SolidWorks does not automatically orient parts to avoid collision. This coincident mate is then suppressed to allow a gear mate to be inserted. The number of teeth of meshed gears is already known, so this value is used to set the gear ratio.

c. *Add Shafts*

Shafts are inserted after all of the gears are positioned and aligned properly. A concentric mate is created between the central axis of the shaft and the central axis of the gear that is on that shaft. For any gears that are meant to be locked to the shaft (i.e. not an idler gear) a plane of the

gear and a plane of the shaft are set to be *coincident*, coupling the motion of the gear and shaft. The shafts are then centered on the gear.

d. Add Bearings

Two instances of each bearing are inserted into the assembly. The center of the two bearings are made *concentric* to the shaft they connect to, and then set *coincident* to either end of the shaft. The bearings for all other shafts are then added. Once bearings are fully defined, the assembly is complete and the program finishes execution zooming to fit the entire assembly in frame. An example of a completed two stage gear train can be seen below (Figure 20).

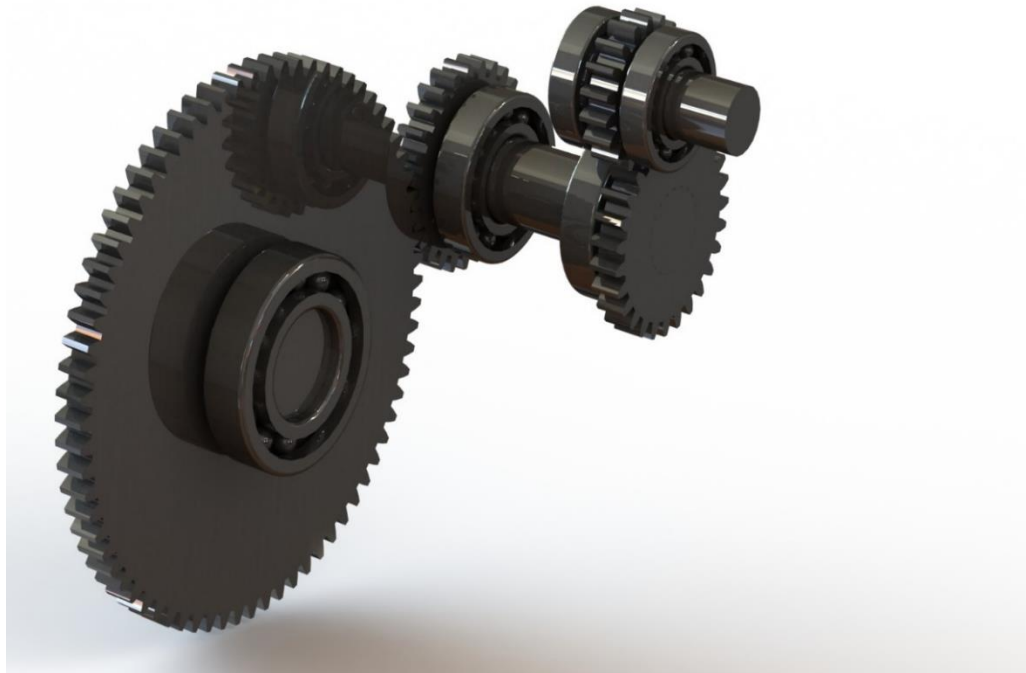


Figure 20. Complete Gear Assembly

II. Results

Many configurations and types of gear trains were tested to ensure proper generation. The program has no built-in limit to the size or number of parts that may be added into an assembly,

however the limiting factor tended to be the computer running SolidWorks. Limitations in C# scripting required the parts to be opened before adding into an assembly. The sheer number of files being opened may occasionally cause SolidWorks to crash depending on the memory capacity of the computer. Time to complete a full assembly ranged from around 30 seconds, for an assembly with fewer than 4 gears, to around 3 minutes to create a system with 19 gears. These times change depending on the computer you are using. Further optimization may be possible to cut down loading times, but it is still many times faster than any designer could ever create manually. The system currently only creates the assembly, and assumes the user has entered the dimensions correctly. It does not verify if the data is correct. This is a feature that would be helpful in the future to prevent invalid parts from being generated.

a. Testing in Advanced Engineering Design Course (ME4320)

To assess the usefulness of such a tool, it was given to students enrolled in an Advanced Engineering Design Course (ME4320) in D-term 2017 to use for a gear-train design project. The project stated requirements of an input and output locations as well as input and output speeds. The students then had to come up with designs for gear trains and perform stress analysis on the system. The students were required to use the tool to generate the gear-trains that they designed for this project. Students were asked to provide feedback as to how the tool helped and how easy it was to use. The survey showed 90% of students successfully generated their designs using the tool (Figure 22 below). Initially there were some issues with sharing the source code with students, which was the main cause for why 43% of students experienced issues (Figure 22 below). Nearly every student successfully generated a gear train with very little help outside of an instructional document reproduced in Appendix C. When asked what they liked about the program, by far the most common response was on the time it saved, and how much simpler it

made the design process compared to having to find gears online or having to design each gear individually (Figure 22 Right). Most student also found the user interface very simple and easy to understand.

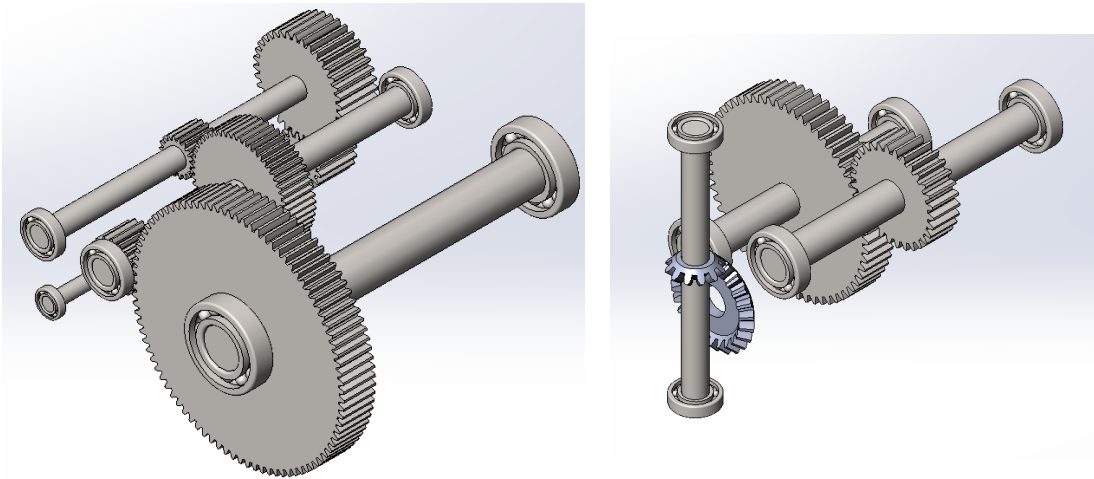


Figure 21. Example of Student generated assemblies

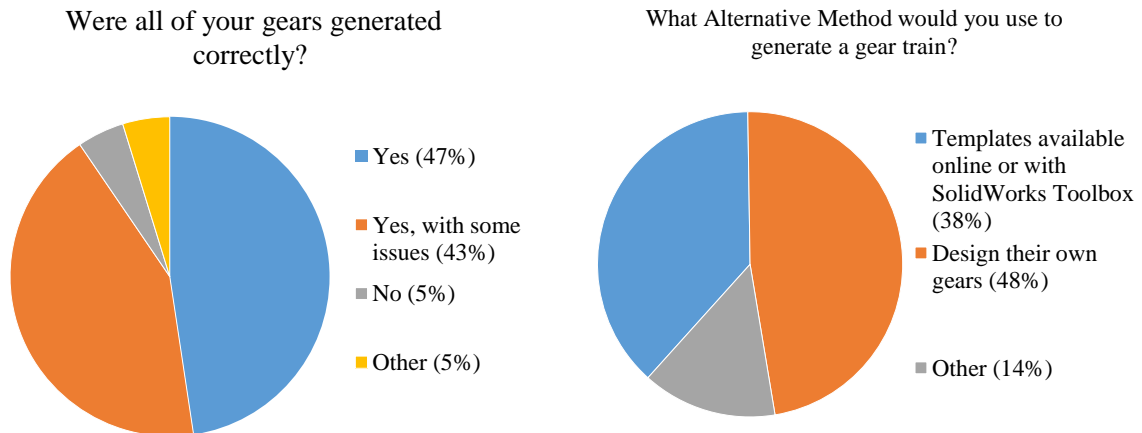


Figure 22. Class Survey Results

The next section will describe how a bond graph is formed given a solidworks assembly file, through the use of graph grammar. This is to accomplish our second objective.

EXTRACTING BOND GRAPHS FROM SOLIDWORKS ASSEMBLIES

This section details the process of extracting SolidWorks data from an assembly and converting that data into a bond graph. The process is illustrated by the flow chart in Figure 23 below. First, the user opens a SolidWorks assembly of a gear train. Using the SolidWorks API, the parts and mates within the assembly are outputted to an identification graph, including mass and size of each part. Grammar rules are then applied to this graph to identify each part as a shaft or a gear, based on the mates between each part. After all parts are identified, this system graph can be opened by the Automated Virtual Lab program and be converted to a bond graph. This graph is then presented to the user.

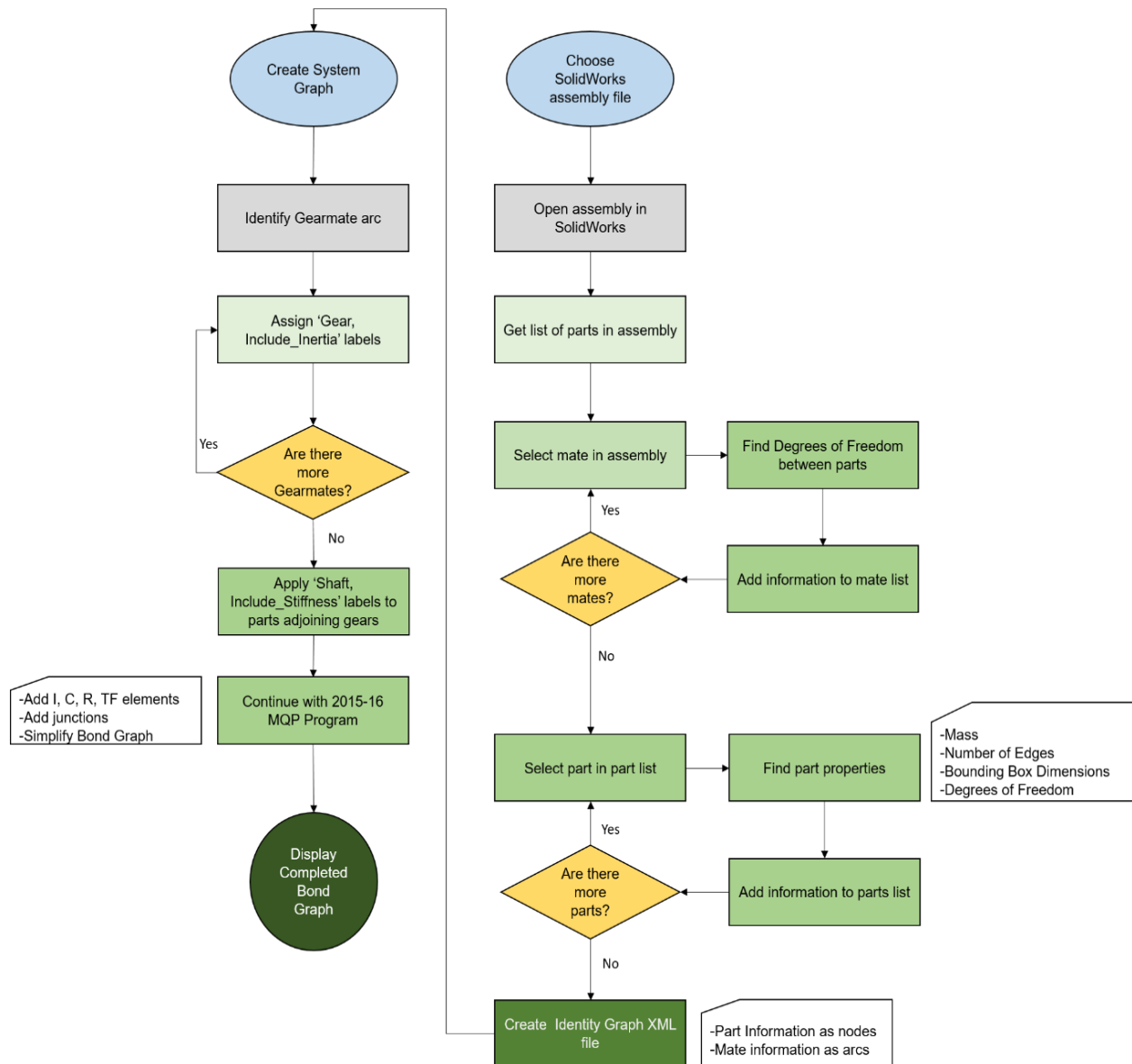


Figure 23: Flow Chart to Create Bond Graphs from SolidWorks Assemblies

A. Reading Part Data

To create bond graphs from a SolidWorks Assembly, the parts within an assembly must be identified along with the relevant part parameters. This is done using the SolidWorks API by generating lists of the parts and mates within an assembly. With these lists, each part is examined to determine the mass, degrees of freedom (DOF), number of edges, dimensions of the bounding

box, and which axis the part can translate or rotate about. This information is added to a list of labels for each part. Each mate is then examined to determine the type of mate, the parts being mated, and the relative degrees of freedom of the two parts. Degrees of freedom are calculated based on how parts can move in relation to each other, i.e. a gear and shaft have 0 degrees of freedom because a gear is fixed to a shaft, and the two cannot translate or rotate independently. All this information is then exported as a GraphSynth compatible XML file. A node is created for each part, with each identified property added as a local label. Each mate is added to the XML file as an arc with the endpoints being the two parts being mated. An arc is created for the type of mate as well as an arc for the degrees of freedom between the two parts. The code snippet below shows an example of how an arc is defined:

```
arc.Name = nameLookup[(string)swComp.Name2] + nameLookup[(string)swComp2.Name2];
arc.start = nameLookup[(string)swComp.Name2];
arc.end = nameLookup[(string)swComp2.Name2];
arc.label = DOF[2, 0] + "DOF,";
arc.label = arc.label + swMateCodes[swMate.Type];
arc.label = arc.label.TrimEnd(',');
```

Figure 24: Code segment to define an arc connecting two nodes

The name of the arc is comprised of the name of the two nodes connected by the arc. As nodes are named in nX convention (n0, n1, ...) as opposed to part names, a lookup dictionary linking part names to node names is used to find the correct node for a given part. The start and end node are identified the same way. An arc label is added for the degrees of freedom as well as the name of the mate between the two parts.

The resulting graph is called the Identity Graph of the system. A sample identification graph is shown below in Figure 25, which shows the identification of a three-stage gear train containing six gears and three shafts. The label of '0DOF' on the arc signifies that the two parts are fixed to each other, and so have no degrees of freedom relative to each other. The

‘Coincident’ label is the primary mate between the two parts, which is copied from the SolidWorks definition. In this graph, the top node is a shaft, the next two are a gear pair, next is a shaft, followed by another gear pair, another shaft, and then the final gear pair. These nodes can be identified by the connecting mates as well as the number of edges on each part (shafts have fewer edges than gears).

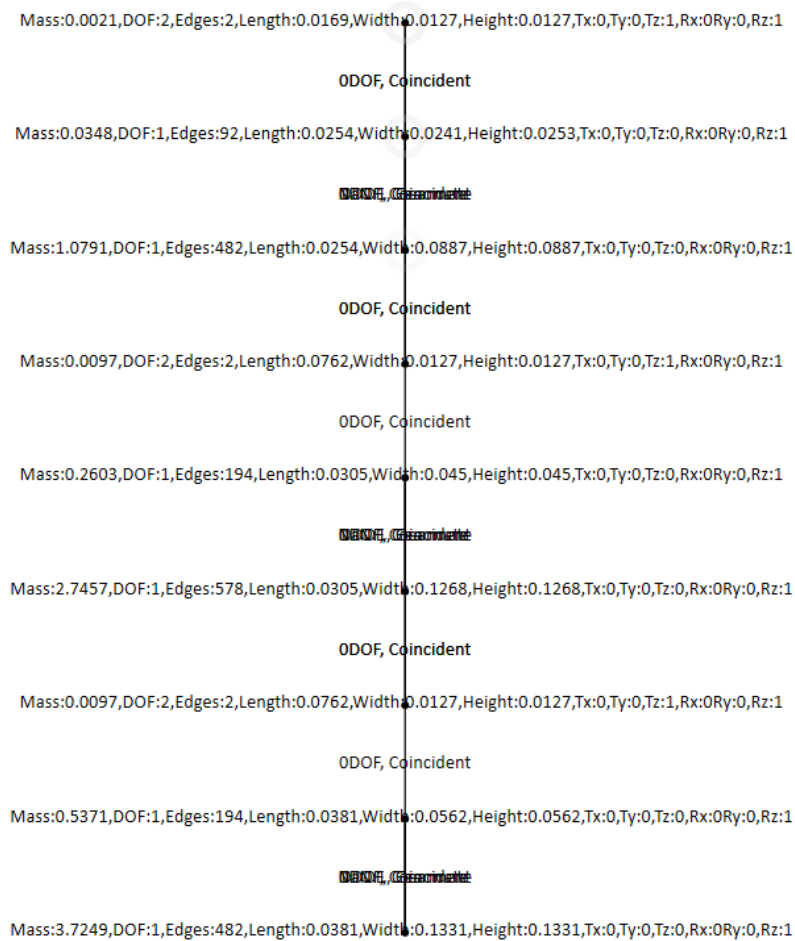


Figure 25: SolidWorks Identification Graph

B. Creating System Graphs

Once the data has been retrieved from SolidWorks and transformed into an identity graph, it must be adapted into a format that is readable by the AVL program. The identity graph is opened and a set of GraphSynth rules converts it into a system graph (Figure 27), which is the type of

graph needed to generate bond graphs. These rules are organized into a SolidWorks Simplification ruleset.

These rules operate by first finding a definitive mate. A definitive mate is a mate that gives insight to the type of parts being mated. For example, a gear mate signifies that the two parts being mated are each a gear. When an arc labeled 'GearMate' is found in the identity graph, the two LabelGears rules (one of which is shown in Figure 26 below) add the labels 'Gear' and 'Include_Inertia' to each node at the endpoint of the arc. The 'Gear' label indicates to the Bond Graph program that the part is a gear and that transformers should be added between the two nodes. The 'Include_Inertia' label indicates that the part has a moment of inertia and should have an 'I' element added to the 1 junction of the gear. The left portion of the rule shows what the rule is searching for, and the right section shows the output of the rule. The 'LabelShaft' rules identify shafts as parts that have zero degrees of freedom relative to a gear, so the rules look for nodes that have a 0DOF arc attached to a node with a 'Gear' label. When this condition is met the labels 'Shaft' and 'Include_Stiffness' are added to the node. The 'Include_Stiffness' rule indicates that a C element should be added off of the 0 element of the shaft. A complete step-by-step process of the rule application can be seen in Appendix D: Rules to Generate System Graphs.

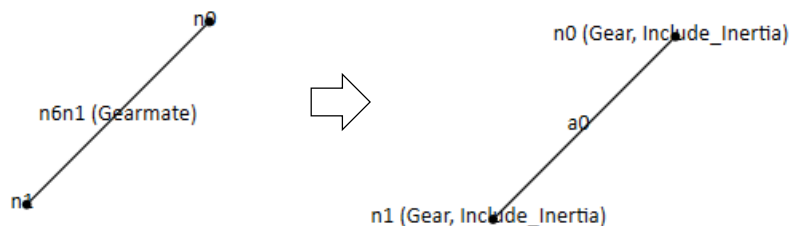


Figure 26: LabelGears Rule

After all rules have been processed and all nodes have appropriate labels applied, the resulting graph is a system graph readable by the AVL program. An example of this graph is shown below in Figure 27, the changes to the identity graph are noted in red. Each node, in addition to the part properties which existed in the previous graph, has two labels added: either ‘Include_Inertia’ and ‘Gear’, or ‘Include_Stiffness’ and ‘Shaft’. Labels on the arcs have been removed and the arcs are currently directionless.

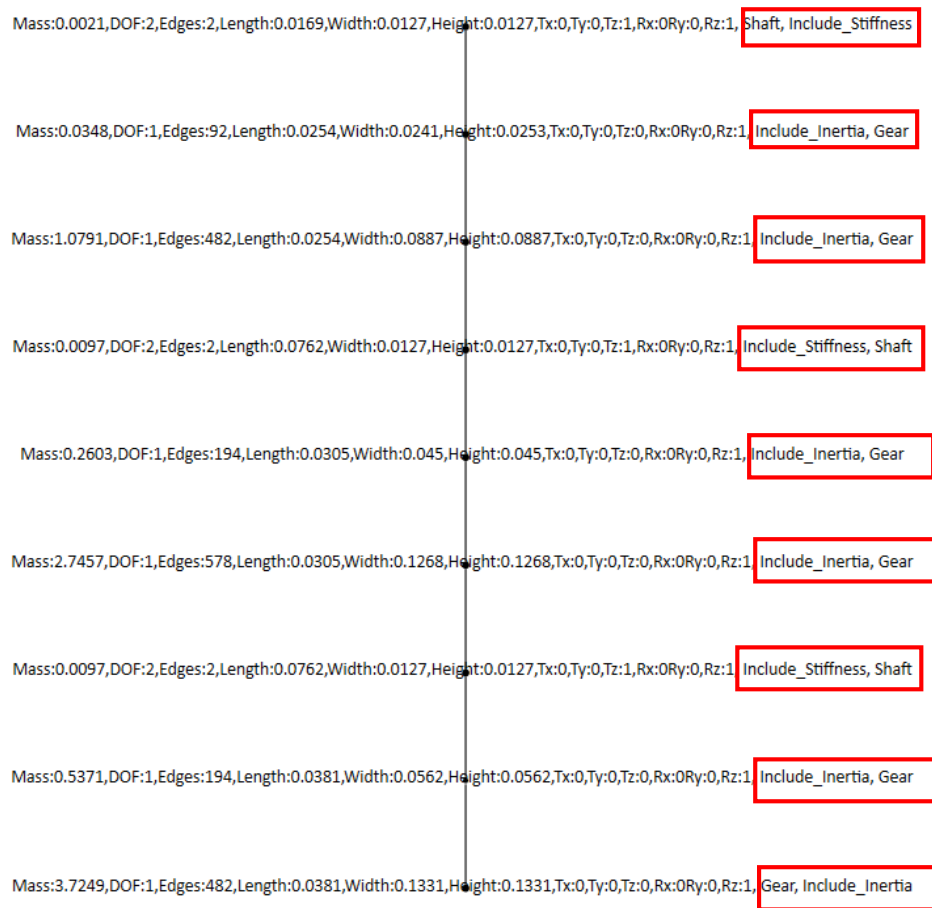


Figure 27: SolidWorks Assembly System Graph

The existing rule sets from the AVL are used to create a bond graph from this system graph, with some modifications. Rules were either added or modified to create the appropriate

gear and shaft nodes. Previously, to create a 1 junction with an I element, a node with the 'Gear' label was changed to a 1 junction, and a new node with the label 'I:J_Added_Inertia' was created. This created a problem where the part properties would be attached to the 1 junction, rather than the I element where they belonged. To solve this, the modified rules first change the node with the 'Gear' and 'Include_Inertia' labels to an I element, a new node is created for the 1 junction, and all existing arcs attached to the 'Gear' node are moved to the new 1 junction node. The same method is used to solve a similar issue with shafts and their stiffness: A new 0 junction is created, and the shaft node is changed to a C element to keep the part information with its appropriate element. It is important that these modified rules be applied after the rules which create transformers, as the transformer rules are based on finding two connected nodes which contain the 'Gear' label and adding a transformer node between them. This must happen first as after the new 1 and J elements are created, the nodes labeled 'Gear' are no longer directly connected because they have the 1 junctions between them. A completed bond graph can be seen below in Figure 28 **Error! Reference source not found.** There is an I element representing a gear's inertia off of each 1 junction, there is a C element off of each 0 junction representing the torsional shaft stiffness of each shaft. A 'TF_GearMesh' node was added to represent the change in rotation between a gear mesh.

C. SolidWorks Identification Results

The combination of the SolidWorks Identification program and the associated rules can properly create a system graph of a gear train from a SolidWorks assembly. The AVL program can interpret this generated system graph and create an accompanying bond graph. This completed bond graph can be seen below in Figure 28. Comparing this graph to the system graph

in Figure 27, the 0 and 1 junctions as well as the transformer elements between 1 junctions have been added.



Figure 28: SolidWorks Assembly Bond Graph

ANALYZING BOND GRAPHS

This section presents a method for automatically generating state equations and then an improved method for assigning bond directions while creating a bond graph.

A. State equations

The main goal of the bond graph is the derivation of dynamic state equations. It would be ideal to be able to automate this process as it allows users to easily check their work, or avoid having to do manual calculations altogether.

State equations are used to determine variation between the displacement and momentum variables of energy storing elements in a system. Each element in a system has two variables associated with it; a “flow” and an “effort” variable. The “flow” variable corresponds to the element’s velocity, current, or air flow in the system. The “effort” variable corresponds to its force, voltage or pressure exerted. In any system, a few of these variables will be assigned as state variables, determined by assigning causality and finding the integral I and C elements. A state variable assignment indicates that the element is in energy storing mode.

The Automated Virtual Lab is able to take the system graph, generate causal bond graphs and identify the state variables. From these causal graphs, grammar rules are used to generate the state equations. However, this is not simply accomplished with just grammar rules. The main issue is the equations are unique for each element in the system, and so cannot be handled by rules alone. To keep track of the equations as they are formed, a C# program runs in the background in between rule-checks and rule-applications.

The flow chart given in Figure 29 shows the overall process of deriving state equations. Two sets of graph grammar rules are used to identify all flows and efforts in a system. Each time

a flow or variable equation is derived, that corresponding node is marked with the label “knowneffort” or “knownflow” to signify that the equation has been created. An equation, or partial equation can then be generated any time another variable is known. Once all nodes have one of these labels, the graph is solved and the equations being stored in the background can be formatted and displayed.



Figure 29. Flow chart for generating state equations

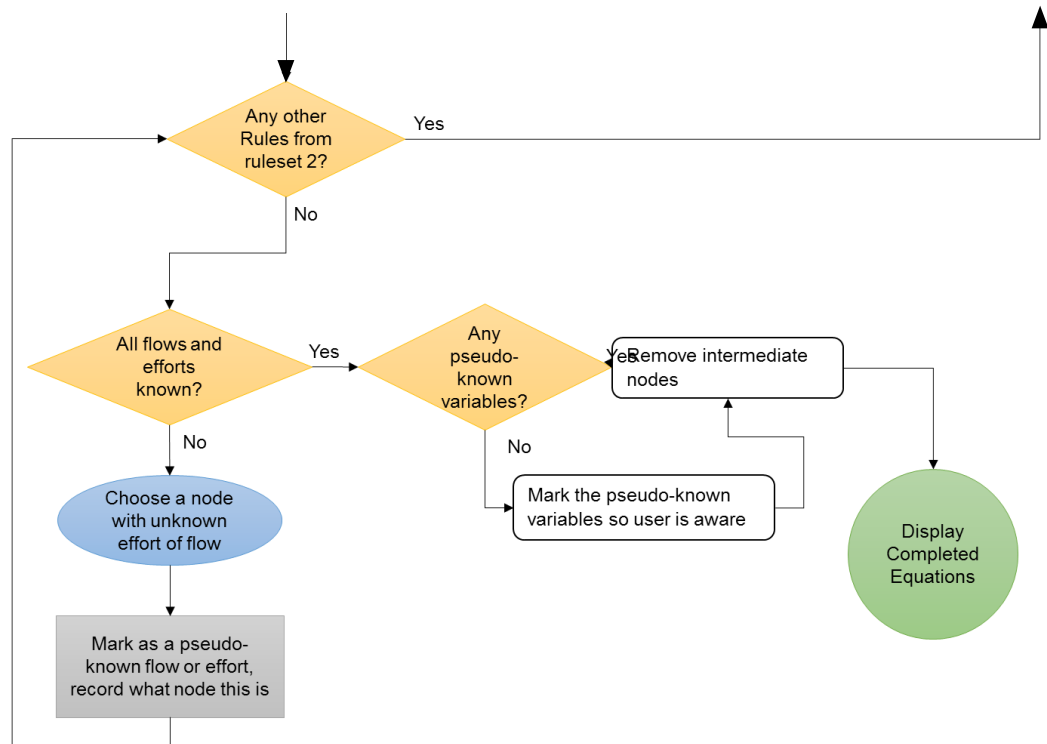


Figure 29 (continued)

1. Format graph

The subsequent rules assume that the flow and effort terms are associated with nodes. As a result, the causality graph must be altered slightly before the other rules are applied. Any arcs connecting a 1 junction and a 0 junction must be split such that a node exists between them to track effort and flow between the junctions. Four rules have been created to account for all combinations of arrows, pointing from $0 \rightarrow 1$ or from $1 \rightarrow 0$ and then with a label of *OPP* or *SAME*. The four rules follow the same pattern, but identify different labels. Figure 30 shows one of these rules and how it results in an intermediate node.

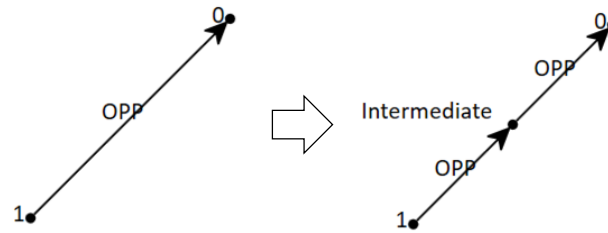


Figure 30. Bond graph rule to add intermediate node

2. Find integral and source elements

The state variables associated with integral I and C elements can be identified using two rules. Once a state variable is identified in the graph, this node is recorded as a state variable in the C# program. If an integral I node is identified, then its flow may be determined through equation 1 below. The bond graph rule will mark this junction with a “knownflow” because the flow variable has an equation.

Table 1. List of Equations used for rules

Element	Integral Causality Equation	Derivative Causality Equation
I-element	$f = \frac{1}{I} \int e dt$ (1)	$e = I \frac{df}{dt}$ (2)
C-element	$e = \frac{1}{C} \int f dt$ (3)	$f = \frac{1}{C} \frac{de}{dt}$ (4)
R-element	$f = \frac{1}{R} e$ (5)	$e = Rf$ (6)

Rules are also developed to search for input sources (forces or velocity sources). The corresponding variables can be substituted for a time varying force or effort, and the source’s node is marked as “knownflow” or “knowneffort”. Figure 31 shows the rule to identify a source of flow. A source of effort uses the same rule with labels for Se instead of Sf.

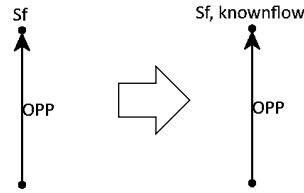


Figure 31. Identifying a source of flow in the bond graph

3. Apply derivative rules

With all state variables and sources identified, and the corresponding equations generated, it is possible to apply derivative causality rules. The rules look for nodes with derivative causality; an I or R node with an arc pointing away with the OPP label, or a C node with an arc pointing toward itself. Figure 32 shows the rule to identify an *I* element in derivative causality. If an “I” node has a known flow and is in derivative causality, the expression for its effort variable can be derived using equation 2 in Table 1. List of Equations used for rules Table 1 above. If there are no identified nodes, then the program moves on to step 4.

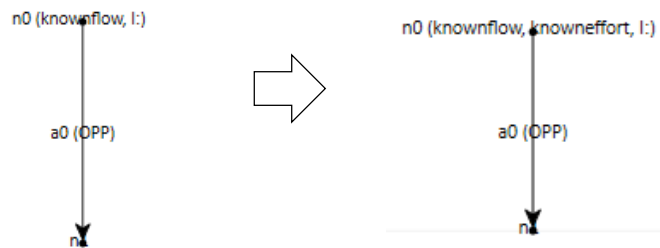


Figure 32. Rule to identify Derivative Causality for an I element

4. Apply summation and equal flow and efforts at junctions

If the flow at a 1 junction is known for some connected element, then all other flows of that junction are equal to that value. The same is true if an effort is known connecting to a 0 junction. These two conditions are tested for, and all other nodes attached to that junction are set equal to the known value.

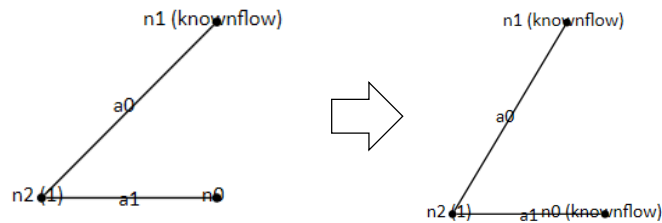


Figure 33. Equal flows at a one junction

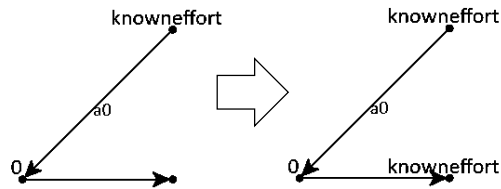


Figure 34. Equal Efforts at a 0 junction

If this ruleset made a change to the graph, then it will return to step 3 and check if there are any new identified rules. If no changes have been made, then the program will continue.

5. Check if equations are solved

The program now checks for any nodes without both known flow and known effort labels. If there are none, then all of the equations are complete and the system is solved. This will be the case when there are more integral elements than derivative. If the graph is not fully solved then the program will assign one element's flow or effort to be known, even if it is not. This node is recorded and then the program goes and checks to see if any more rules can be identified with

this new known element. This effectively creates a “pseudo known” variable. This may occur multiple times while solving for state equations, leading to multiple pseudo known variables.

The pseudo known variables still result in valid equations, but the equations generated will now have state variables *and* the pseudo known variables. Each additional pseudo variable introduced will result in one additional equation. This is one of the current limitations of the program as it cannot solve for just the state variables in large systems with many derivative elements. It may be possible to use some form of symbolic manipulation or algebra either with MATLAB or in C# directly to allow the system of equations to eliminate the pseudo known variables.

6. Format equations for output

Once all variables are known, the initial formatting of the graph, where intermediate nodes were added, is now removed reverting the graph to its state prior to creating the state equations. The state variables and their derived equations are then displayed in the console screen.

B. Results

To verify that the rules were properly generating, a simple mass spring damper system was tested. The equations generated match completely with a manual solution to the system.

Solved Equations:

$$P^2 = R1*(1/M2*P2)+K3*x3+F4(t)$$

$$x^3 = 1/M2*P2$$

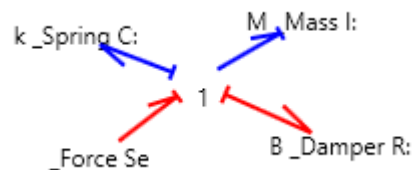


Figure 35. Causality bond graph of mass spring damper system

When using the program for systems that have a large numbers of elements, especially in derivative causality, the program may fail to solve for all terms. This can happen when there are too many unknowns. The same issue can arise when the graph has loops, where nodes connect back in a circle. This situation can prevent the rules from identifying the correct configurations.

C. Direction of Bond Graphs

The majority of the bond graph generation was completed by the previous MQP [3], [4]. One task that needed further work was assigning the directions of the bond graph arrows throughout the graph. The program previously assigned directions randomly, dependent only on which nodes were created first. It is important to consider the correct direction, as these arrows determine the direction of motion or rotation of a system.

It is necessary for the program to know where the input location of a system is to identify the correct direction of flow throughout a system. While it would be possible to detect the input by looking for a source of flow or effort, there may be multiple of these within a single system making it more convenient for the program to explicitly know the start location. This was accomplished by introducing a new label to the graph, a start node. This is attached with an arc to the starting location.

Additional direction nodes are also specified to indicate that the velocity of a node is in the positive or negative direction. This is signified by attaching a $-x$ or $+x$ to inertial nodes (Figure 36).

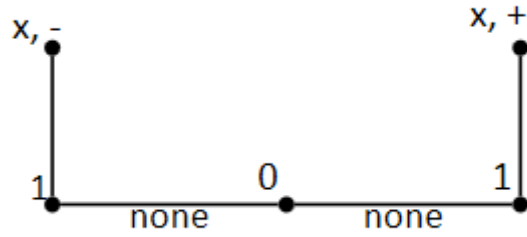


Figure 36. Direction assignment to the nodes

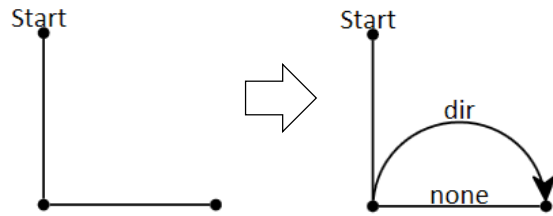


Figure 37. Rule to apply start direction to system

Rules then modify the graph so that all direction arrows point away from the starting location (Figure 38). This propagates throughout the system, until all direction labels have been applied.

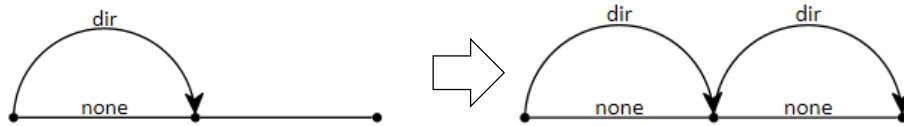


Figure 38. Rule to propagate positive direction through the system

Now any nodes that have a negative direction associated with them should be flipped(Figure 39).

This is done with an additional rule to switch the connected arcs.

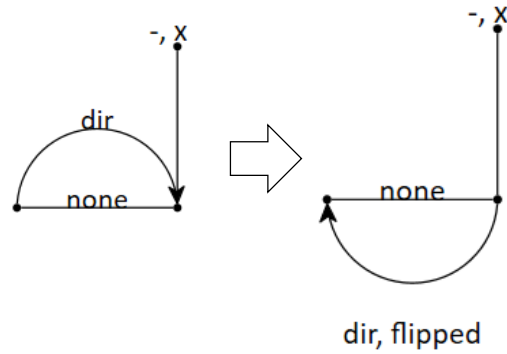


Figure 39. Rule to flip arc direction

The start labels and direction nodes are now removed from the graph, completing direction assignment.

D. Bond Graph Results

The changes to the bond graph program enables all directions to be set by the user, and ensure the direction of energy flows throughout the system is correct. In the event that there are nodes where a + or – direction have not been assumed, then a positive direction is assumed. It has also been designed such that the directions could eventually be applied in other dimensions other than just x, allowing for 3D systems to have directions assigned.

CONCLUSIONS

During this project, solutions to three limitations in design automation have been provided. The first solution provides the ability to automatically generate gear train parts and assemblies in SolidWorks to provide good visualization to users of design automation tools. This speeds up the design process by allowing many designs to be generated and analyzed in quick succession. The tool was tested among students in a design course and favorable feedback was received. The solution has been devised in such a way that automated gear-train designs found in XML format can also be extracted to 3D models.

The second solution is the ability to extract dynamic models of transmission systems using the bond graph technique. This system allows a user to apply dynamic analysis directly to a gear train and continue with bond graph modeling. This provides a more visual demonstration of how bond graphs relate to mechanical systems.

The final solution involved provides the ability to extract state equations directly from a bond graph through the use of graph grammar rules. This is the final part to the automated bond graph modeling, allowing all stages of a bond graph to now be generated. State equations were verified for basic systems, proving that the method for solving was correct. This tool will allow large systems to be quickly solved, preventing the need for manual calculations.

These solutions are developed for use in classrooms, providing tools to students to understand more complex engineering systems with less tedious handwritten work. Students will have gained an understanding of the topics in design through simple problems, and then this tool will augment their coursework to visualize and generate complex systems.

A. Recommendations and Future Work

Additional work could be done to these programs to streamline the process, fix minor bugs, expand the functionality, and make it more user friendly.

I. Automated Gear Generation

With regards to the gear generation portion of this project, the code can be further optimized to reduce system resource usage. In the current iteration it is required that a part must be open to be added to the assembly. Having many parts open, especially in large gear trains, uses a significant amount of RAM on the user's computer. Modifying the assembly process by changing how parts are added can reduce the use of RAM. Additional gears, such as helical bevel gears and planetary systems could be added to broaden the scope of this program. The creation of shafts could also be automated. Currently shafts need a specified length and diameter when using the GUI, and are placed centered within the first gear they are mated to. In the future, the diameter of shafts could be determined automatically from the bore diameter of the mated gears, and the length and position could be determined by the placement and distance between gears on the shaft. The survey completed in the ME4320 class also asked for recommendations. Common recommendations included, adding the ability to adjust shaft location, and automatically creating and inserting shafts based on the gears' locations. Some students also suggested that gear locations be automatically determined by using the gears diameter.

II. Extraction of Bond Graph

Additional functionality could also be added to the SolidWorks identification portion of the program. Currently the rule sets can only identify shafts and gears in a gear train. To broaden the scope of this tool future versions should identify more systems, such as springs, levers, dampers, and other common mechanisms. Given the parameters read from SolidWorks (mass,

material, size, and contact areas, etc.), numerical values for inertia, stiffness, or damping coefficients could be calculated automatically.

III. State Equations and System Response

Assigning causality and state equation generation can be improved upon by building in MATLAB functionality. Scripts can be written to solve and graph the differential state equations that are generated. Currently equations are displayed in the console window, and then would be copied into a MATLAB program and then solved. Removing this step would simplify the process and demonstrate how to solve the equations while generating.

Lastly, all aspects of this activity can be compiled into a singular program. Currently the gear generation, SolidWorks identification, and bond graph tools reside in three separate programs. The SolidWorks identification portion can be merged into the Automated Virtual Lab, which currently handles the bond graph generation, assigning causality, and generating state equations. Merging these aspects would allow for a more intuitive process to take a SolidWorks assembly into a bond graph or to create the SolidWorks Assembly.

REFERENCES

- [1] Schmidt, L., Shetty, H., and Chase, S. "A Graph Grammar Approach to Mechanism Synthesis." *ASME J. Mech. Des.* (2000): 371–376.
- [2] Li, X., & Schmidt, L. "Grammar-Based Designer Assistance Tool for Epicyclic Gear Trains." *Journal of Mechanical Design* (2004): 126(5), 895.
doi:10.1115/1.1767823.
- [3] Mancini, F., Grande, D., & Radhakrishnan, P., "An Automated Graph Grammar Based Tool to Automatically Generate System Bond Graphs for Dynamic Analysis," *Volume 1B: 36th Computers and Information in Engineering Conference*, 2016.
- [4] Mancini, F., Grande, D., & Radhakrishnan, P, "An Automated Virtual Lab for Bond Graph Based Dynamics Modeling Using Graph Grammars and Tree Search," *Volume 5: Education and Globalization*, pp. doi:10.1115/imece2016-66110, 2016.
- [5] "20-Sim," [Online]. Available: <http://www.20sim.com/>.
- [6] X. Xu, *Integrating Advanced Computer-Aided Design, Manufacturing, and Numerical Control: Principles and Implementations*, IGI Publishing, 2009, pp. 90-92.
- [7] W. Fu, A. Eftekharian, P. Radhakrishnan, M. Campbell and C. Fritz, "'A Graph Grammar Based Approach to Automated Manufacturing Planning," in *ASME 2012 International Design Engineering Technical Conferences*, Chicago, 2012.
- [8] Stock Drive Products/ Sterling Instrument, *ELEMENTS OF METRIC GEAR TECHNOLOGY*.

- [9] R. L. Norton, *Machine Design: An Integrated Approach*, New Jersey: Prentice Hall, 2006.
- [10] Swantner, A., & Campbell, M. I, "Topological and parametric optimization of gear trains," *Engineering Optimization*, pp. 44(11), 1351-1368.
doi:10.1080/0305215x.2011.646264, 2012.
- [11] Tudose, L., Buiga, O., Ştefanache, C., & Sóbester, A., "Automated optimal design of a two-stage helical gear reducer," *Structural and Multidisciplinary Optimization*, pp. 42(3), 429-435. doi:10.1007/s00158-010-0504-z, 2010.
- [12] Kwon, H. S., Kahraman, A., Lee, H. K., & Suh, H. S., "An Automated Design Search for Single and Double-Planet Planetary Gear Sets," *Journal of Mechanical Design*, p. 136(6), 2014.
- [13] Lin, Yi-Shih, Kristina Shea, Aylmer Johnson, John Coultate, and Jamie Pears. "A Method and Software Tool for Automated Gearbox Synthesis," *Proceedings of the ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, 2009.
- [14] Schwarz, C., Bachinger, M., Stolz, M., & Watzenig, D, "Tool-driven Design and Automated Parameterization for Real-time Generic Drivetrain Models," *MATEC Web of Conferences*, pp. 28, 03001. doi:10.1051/mateconf/20152803001, 2000.
- [15] "3D CAD Design Software SolidWorks", [Online]. Available:
<http://www.solidworks.com/>.
- [16] "Romax Technology," [Online]. Available: <https://www.romaxtech.com/>.

APPENDIX A: PSEUDO CODE OF GEAR GENERATION

```
GET list of all running processes on computer
IF SolidWorks is not running
    Start SolidWorks process, wait until it is loaded
IF SolidWorks should run in background
    Set SolidWorks app to be invisible
ELSE
    Make SolidWorks visible
IF data is from xml document
    GOTO reading xml
ELSE read GUI data

FOR all gears to be created
    CASE: type of gear
        spur:
            open spur template file
            set Number of teeth
            set Gear width
            set Bore diameter
            set Pitch
            set Material
            Rebuild the part
            Save part to the output directory
        bevel:
            open bevel template file
            set Number of teeth
            set Cone angle
            set Gear width
            set Bore diameter
            set Pitch
            set Material
            Rebuild the part
            Save part to the output directory
        helical:
            open helical template file
            set Number of teeth
            set Gear width
            set Bore diameter
            set Pitch
            set Material
            set helix angle

            IF right hand helix angle
                Unsuppress right hand tooth cut and pattern
            ELSE left hand helix angle
                Unsuppress left hand tooth cut and pattern
            Rebuild the part
            Save part to the output directory

        worm:
            open worm template file
            set Number of teeth
            set Gear width
            set Bore diameter
            set Pitch
            set Material
            set gamma angle

            Rebuild the part
            Save part to the output directory
        worm wheel:
            open worm wheel template file
            set Number of teeth
            set Gear width
            set Bore diameter
```

```

        set Pitch
        set Material
        set gamma angle
        set number of threads of meshed worm wheel
Rebuild the part
        Save part to the output directory
    END CASE
END FOR
Close all documents

FOR each shafts in the assembly
    open shaft template file
    set shaft diameter
    set shaft length
    set material
    rebuild the part
    Save shaft to the output directory

    IF bearings are being added
        open bearing template file
        set shaft diameter
        rebuild the part
        Save shaft to the output directory
    END FOR

Open all gear and part files

FOR each gear to be added
    Add the gear into the assembly
    Close the gear part file
    Add a point into the 3D sketch (location of gear's coordinate)
    Make sketch point and gear origin coincident
    Make Gear central axis parallel to its axis of rotation (x/y/z)
END FOR

FOR all gears in the assembly
    FOR each other gear in assembly
    IF two gears should be meshed
        Align one gears "Gear_GAP" with other gear's "Gear_TOOTH" plane
        Suppress this mate
        Add gear mate, use number of teeth for ratio
    END FOR
    END FOR

FOR each shaft to be added
    Add shaft into the assembly
    Set shaft axis concentric to the gear it connects to
    Center the shaft on the gear (distance mate with distance=0)
END FOR

FOR all gears
    IF gear is locked to shaft (not idler)
        Make plane of gear and plane of shaft coincident
    END FOR

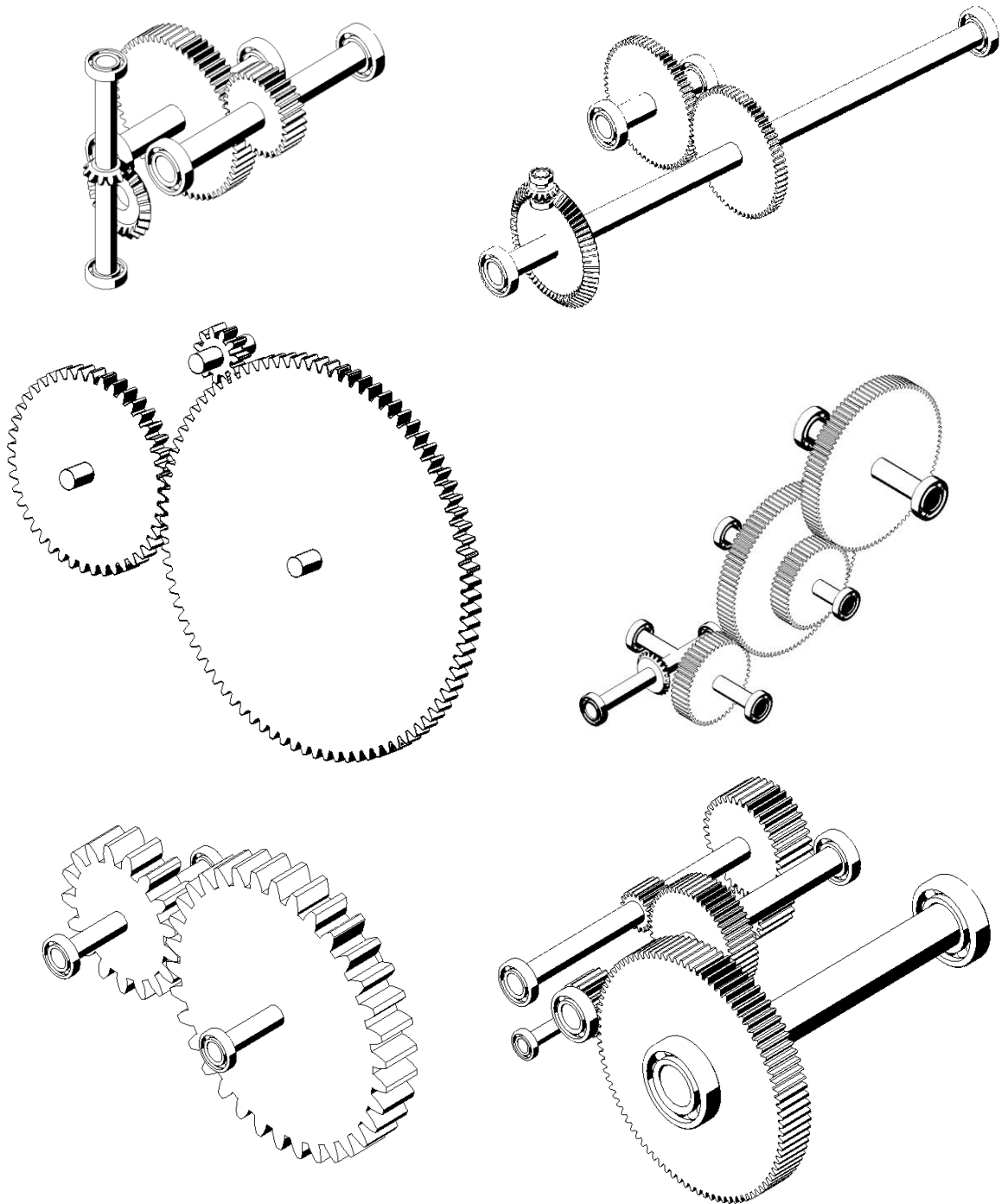
FOR each shaft with a bearing
    Add corresponding bearing to the assembly
    Make concentric to shaft
    Align to the end of the shaft

    Add a copy of the bearing
    Make concentric to shaft
    Align to the other end of the same shaft
END FOR

Save completed assembly as assembly and as Edrawing
Zoom to fit Assembly

```

APPENDIX B: STUDENT GENERATED GEAR TRAINS



Gear Designs Courtesy of ME4320 Class Members: (From top down) Mathew Lepine, Fredrick Burgwardt, Mackenzie Miner, Aaron Pepin, Kayleah Griffen, Alessandra Paolucci.

APPENDIX C: USER GUIDE FOR GENERATING GEARS


Overview:

This program was created to speed up the creation of solid model gear files and assemblies. Users specify various design parameters for each gear, shaft and assembly and then the program uses solid works API tools to generate the models.

System Requirements

OS: Windows 7/10

Installed Programs: SolidWorks 2016 (potential compatibility issues with other release versions)

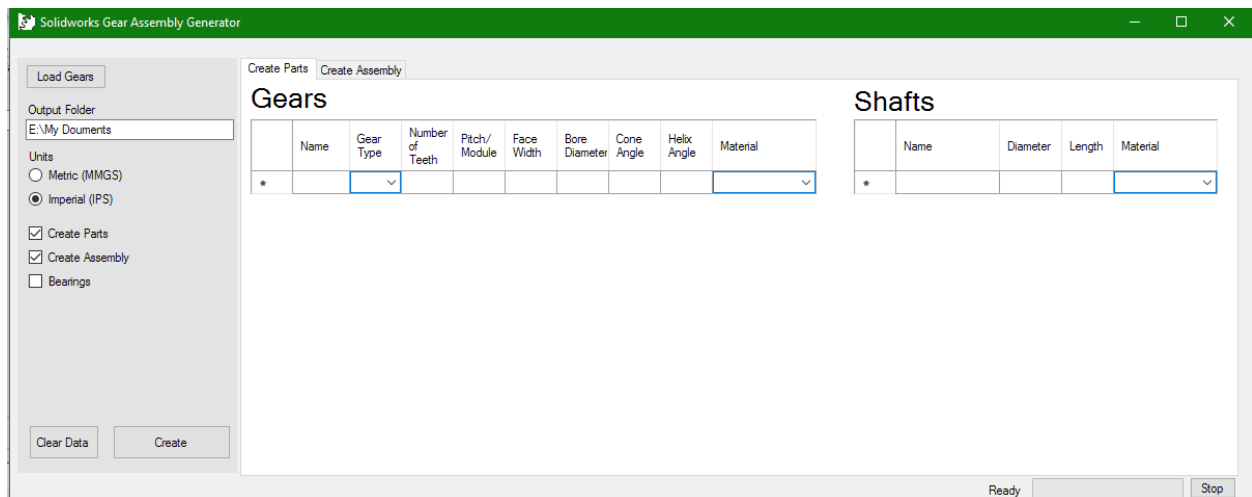
To run the program, download all of the files and place the folder somewhere on your PC. Then Run the SW Gear Generation shortcut located in the main folder:  SW Gear Generation

It is recommended that SolidWorks already be running, with no solidworks files open when gear generation begins.

The large number of files being generated and opened at once can tax your computer system or SolidWorks depending on the resources available.

User Interface

Upon opening you will be presented with this screen:



You can begin entering gear information right away into the columns in the fields. You may also load information from previously generated files by clicking “Load Gears”.

All text fields must be entered for a gear to generate properly.

Step 1: Enter Gears and shafts:

Create Parts		Create Assembly												
Gears										Shafts				
	Name	Gear Type	Number of Teeth	Pitch/Module	Face Width	Bore Diameter	Cone Angle	Helix Angle	Material		Name	Diameter	Length	Material
▶	Gear A	be...	22	12	1	1	0	0	Copper	▶	Shaft 1	1	2	Brass
	Gear B	be...	43	12	1	1	0	0	ABS PC		Shaft 2	1	2	Copper
*										*				

The first tab is where you specify the details about each gear you create.

Name: The can be any name you want, the SolidWorks part file will have this name

Type: (Spur, Bevel, Helical, Worm*, Worm Wheel*) *Work in progress, not currently functional

Bore Diameter: The size of the hole that the shaft fits through

Pitch/Module: Enter diametral pitch if working in Imperial, enter module if in metric

Cone Angle: Bevel gears only, (Set to 0 if any other gear)

Helix Angle: Helical gears only. Enter a (+) angle for right hand cut and (-) for left hand cut (set to 0 if any other gear)

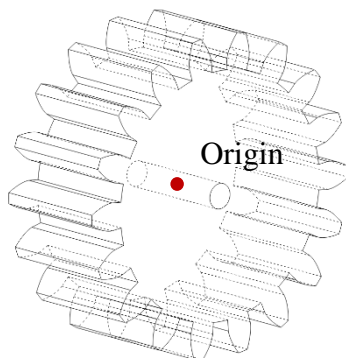
Material: Sets material from some common Materials in Solid works, can be manually changed later

Step 2: Creating an Assembly

Next switch to the Create assembly tab to enter information for the assembly.

You must specify gears and shafts in the **Create parts tab** before entering data here.

Create Parts		Create Assembly							
Assembly									
	Gear	X coord	Y coord	Z coord	Axis of Rotation	Idler	Connected Gear	Connected Shaft	
▶	Gear A	1	1	2	X	<input type="checkbox"/>	Gear B	Shaft 1	
	Gear B	1	1	3	X	<input type="checkbox"/>	Gear A	Shaft 2	



Coordinates: Coordinates are specified from the center of the gear along its axis of rotation.

Axis of rotation: Sets which axis the gear rotate about.

Idler: If checked the gear will not be lock to the shaft it is centered on

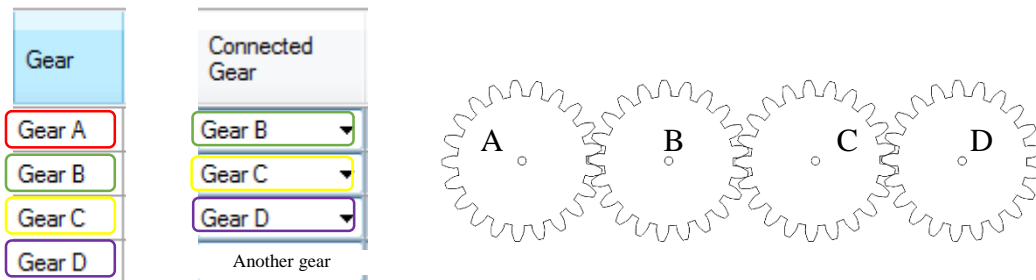
Connected Gear: Used to specify gear meshing. If two gear should mesh, set the “connected gear” to the other gears name.

Ex: Gear A and Gear B should mesh together

Gear	X coord	Y coord	Z coord	Axis of Rotation	Idler	Connected Gear
Gear A	1	1	2	Y	<input type="checkbox"/>	Gear B
Gear B	1	1	3	X	<input type="checkbox"/>	Gear A

Multiple gears in a row: To create a “chain” of gears the first gear is connected to the second gear. The second gear is connected to the third. The final gear is then connected to the first

Ex: 4 gears meshed together in a row



-Gear D may be connected to another gear, or if it is only on a shaft, Select itself as the connected gear.

** The two gears aren't guaranteed to align correctly just because they are meshed. The distance between the gears is specified by setting the coordinates of each gear. Teeth will line up, but there are some known issues with having the teeth align correctly. **

Connected Shaft: Specify the shaft that the gear should be attached to. Multiple gears may exist on the same shaft. All gears should have a shaft specified

Step 3: Settings

Before creating the parts, ensure that the settings are properly set:

Output Folder

Specify Output Folder: The output folder is where all files will be created and saved. You may leave it blank, the system will prompt you to select a folder when you begin creating the part.

****Any Files with the same part names in the output folder will be overwritten, including the configuration.csv file**:** make sure you don't have any files that will be lost!

Units

Metric (MMGS)

Imperial (IPS)

Units: All units are specified by the two buttons on the left of the program. If metric is selected, dimensions will be in millimeters, grams, seconds. If imperial units are; Inches pounds seconds. All angles are in degrees.

Create Parts

Create Assembly

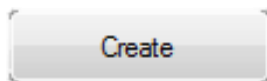
If you wish to only create the parts, uncheck the "Create Assembly". If the parts have already been created, and are located in the Output folder, then you may uncheck "Create Parts". **Unchecking "Create Parts" is not recommended as the assembly may fail if the parts have been changed or features are missing from the part.**

Bearings

If checked bearings will automatically be added to either end of all shafts. The bearings are simple mock ups of bearings, and act more as place holders than properly designed bearings.

Make sure that SolidWorks is open now!

After entering all specifications and you are ready to create the gears, click create.



Step 4: Waiting

Once pressing "Create", the system will begin to initialize SolidWorks. It starts creating all the parts, then adds them into the assembly one at a time, creating all the necessary mates and relations to create the files. The entire process should take less than a few minutes (system dependent), approximately 60-90 seconds for a system of 6 gears and shafts.

- Once complete, the assembly file will be opened and displayed in solid works.
- All files are now created in the output folder specified.
- A file called "Gear config.csv" is also created, which contains all the data for the system. If you need to modify your system later you can load this into the program by clicking "Load gears".

If you come across any bugs or need assistance, please email us:

cjalicchio@wpi.edu , jsvitiello@wpi.edu, crtaylor2@wpi.edu

APPENDIX D: RULES TO GENERATE SYSTEM GRAPHS

The following rules convert an identity graph to a system graph. The first rule, ‘LabelGears’ (Figure 40) adds the labels ‘Gear’ and ‘Include_Inertia’ to two nodes connected by a Gearmate arc, and then removes the Gearmate arc.

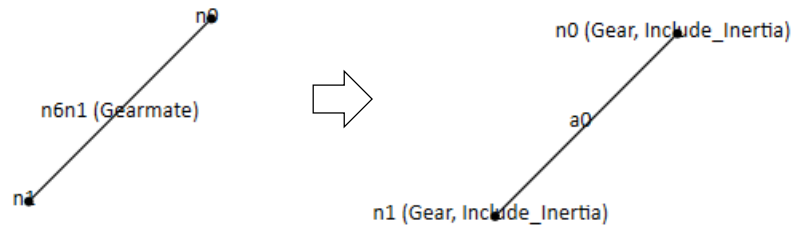


Figure 40: LabelGears Rule

The next rule, LabelShafts (Figure 41), adds the labels of ‘Shaft’ and ‘Include_Stiffness’ to a node which is connected to a ‘Gear’ node and which also does not have the ‘Gear’ label already applied.

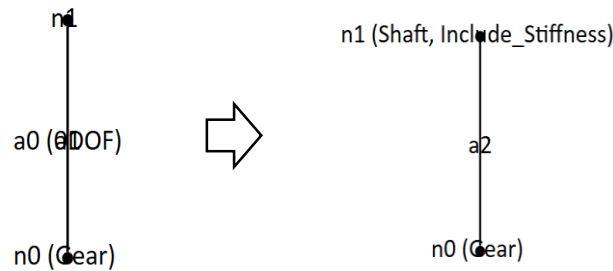


Figure 41: LabelShafts Rule

The RemoveCoincident rule, Figure 42, removes the ‘0DOF’ and ‘Coincident’ labels from an arc connecting a gear to a shaft, as these labels are no longer needed.

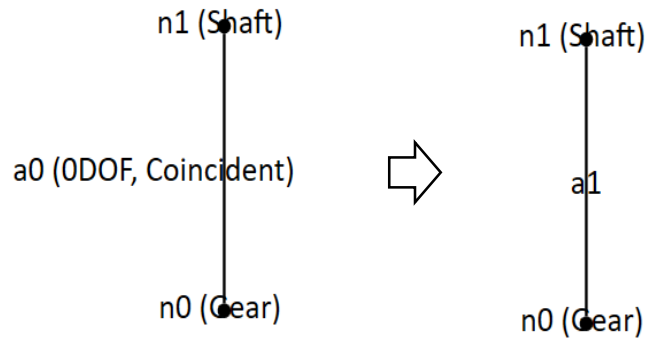


Figure 42: RemoveCoincident Rule

The RemoveGearmate Rule, Figure 43, confirms that there is no Gearmate arc between two gear nodes. Without this rule there would be multiple arcs connecting two nodes, which is not expected by the AVL program.

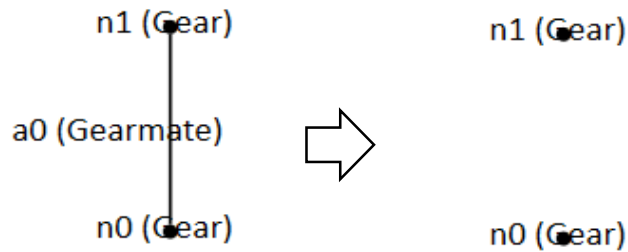


Figure 43: RemoveGearmate Rule

The last rule, RemoveGear0DOF (Figure 44), is similar to the previous in that it assures there are no extraneous arcs connecting gear nodes. In this case it removes the 0DOF arc.

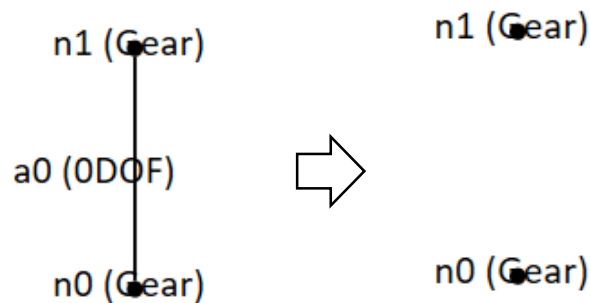


Figure 44: RemoveGear0DOF Rule

APPENDIX E: RULES TO GENERATE STATE EQUATIONS

A. Rules to Format Graph

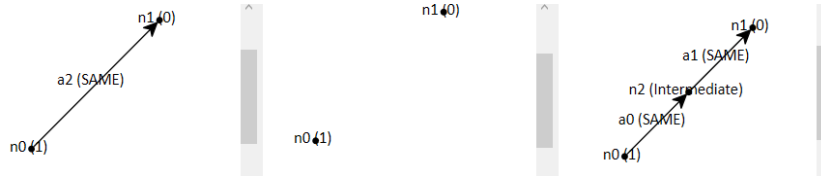


Figure 45. Rule to add intermediate nodes, configuration 1

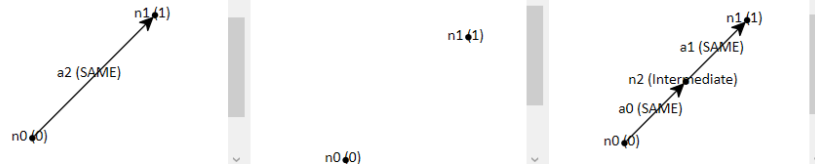


Figure 46. Rule to add intermediate nodes, configuration 2

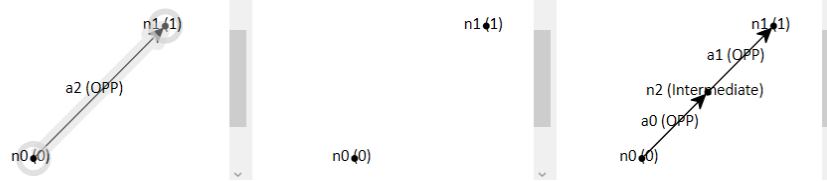


Figure 47. Rule to add intermediate nodes, configuration 3

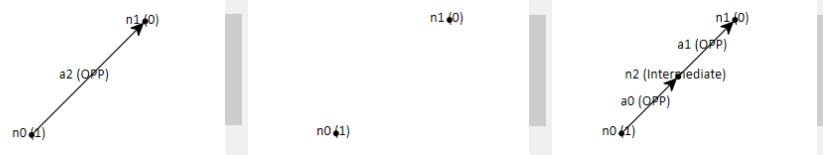


Figure 48. Rule to add intermediate nodes, configuration 4

B. State Equation Ruleset 1

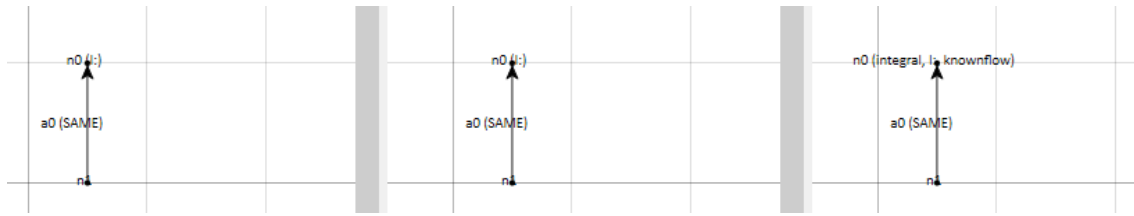


Figure 49. Rule to identify integral causality

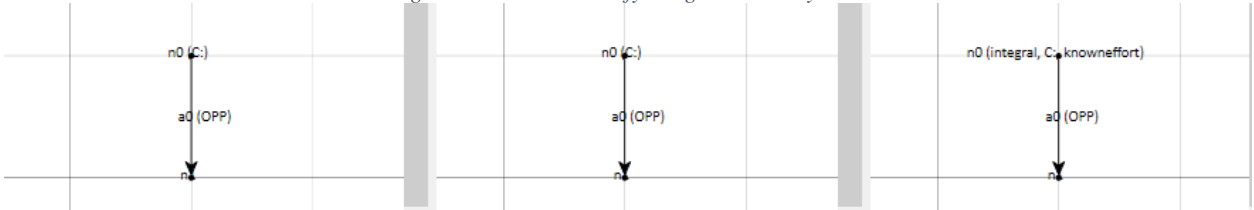


Figure 50. Rule to identify integral causality of C element

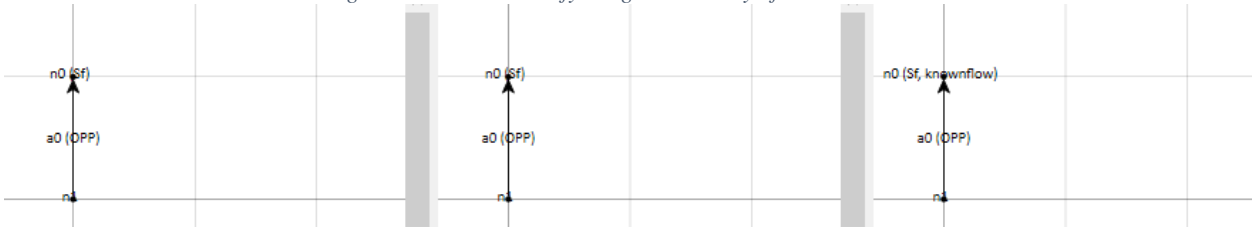


Figure 51. Rule to identify source of flow

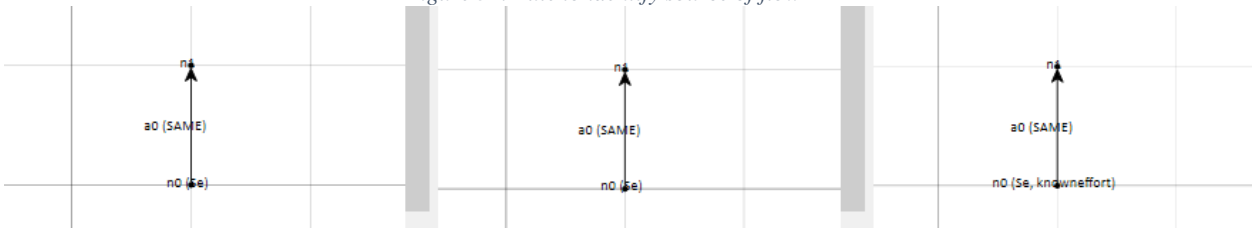


Figure 52. Rule to identify a source of effort

C. State Equation Ruleset 2

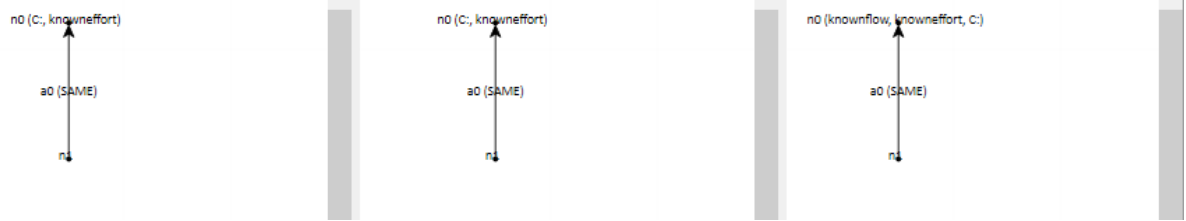


Figure 53. Rule to identify derivative causality of C element

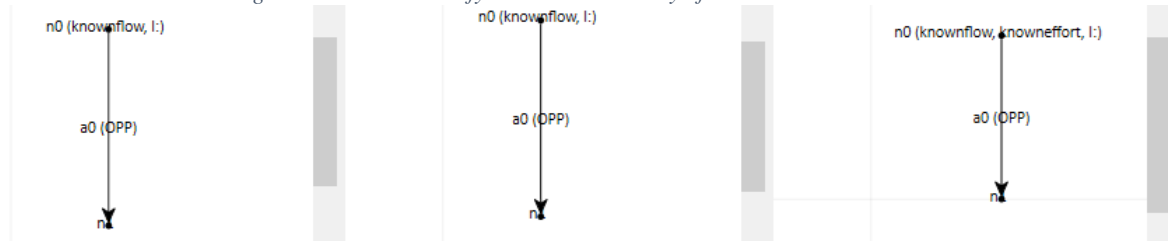


Figure 54. Rule to identify derivative causality of I element

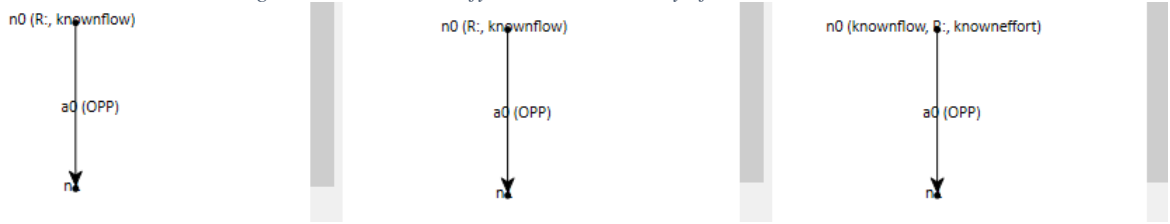


Figure 55. Rule to identify derivative causality of R element

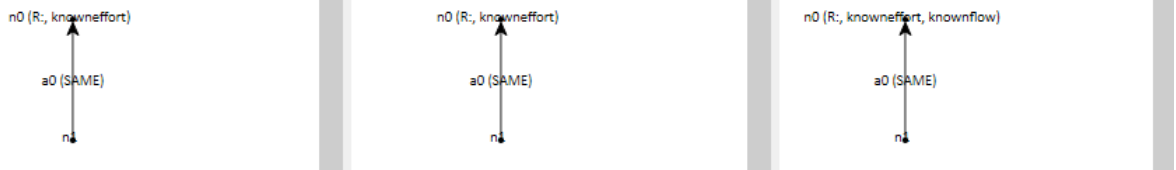


Figure 56. Rule to identify integral causality of R element

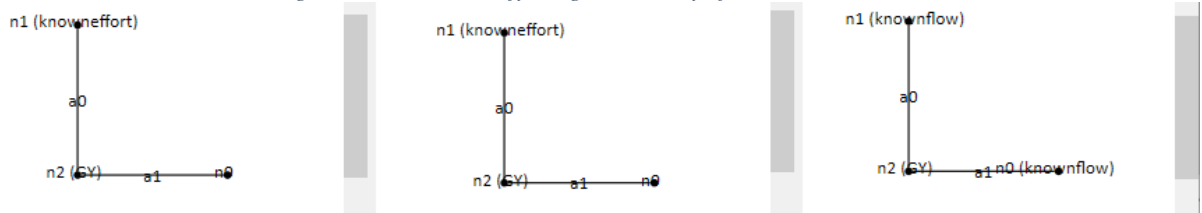


Figure 57. Rule to identify gyrator node with known effort

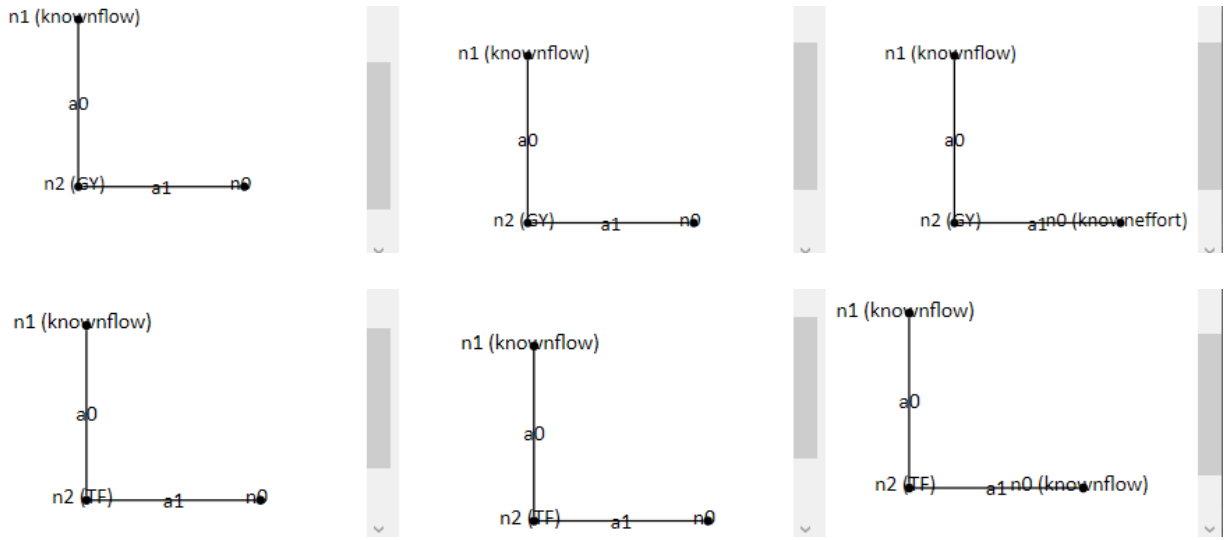


Figure 58. Rule to identify gyrator node with known flow

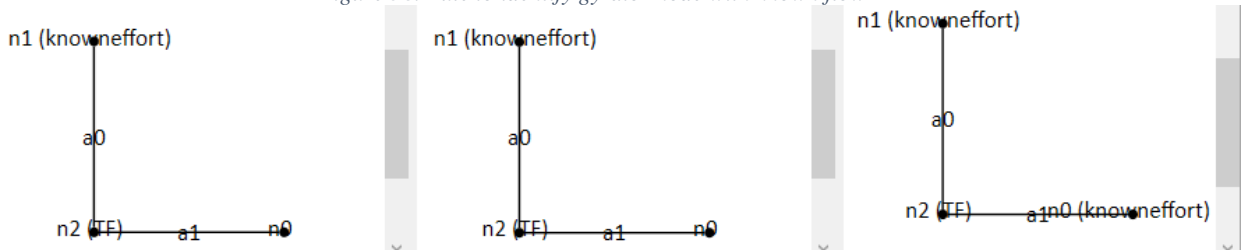


Figure 59. Rule to identify transformer node with known effort

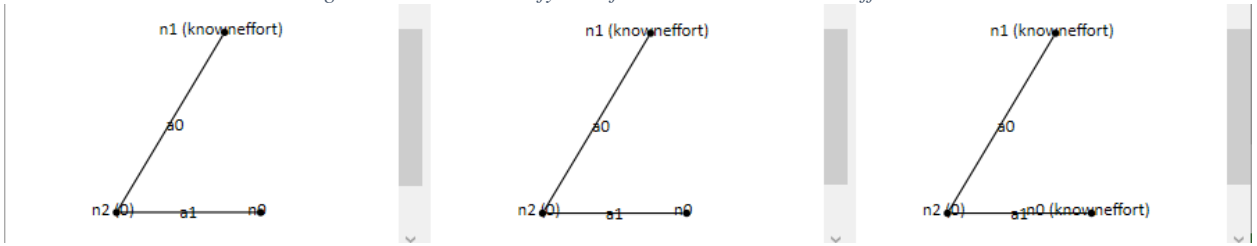


Figure 60. Rule to identify 0 junction with a known source of effort

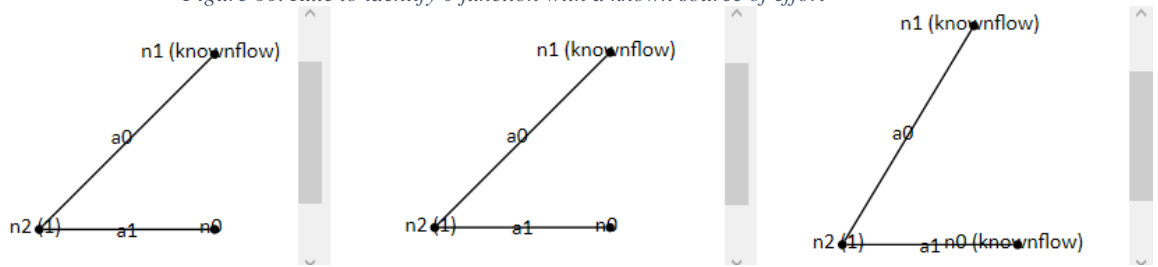


Figure 61. Rule to identify 1 junction with a known source of flow

D. State Equations Summation at junction

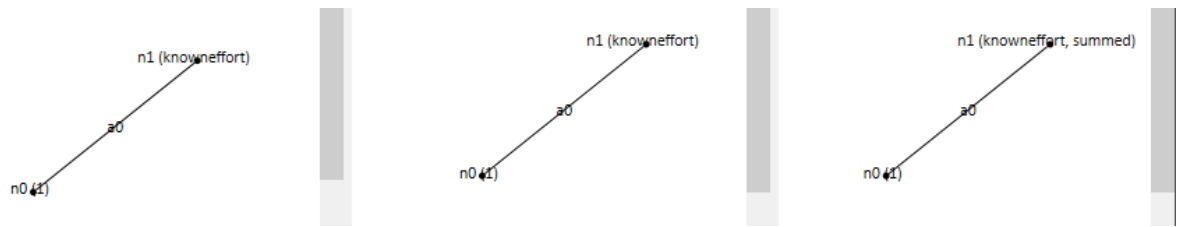


Figure 62. Rule to indicate if a node has been used in a summation equation of a 1 junction

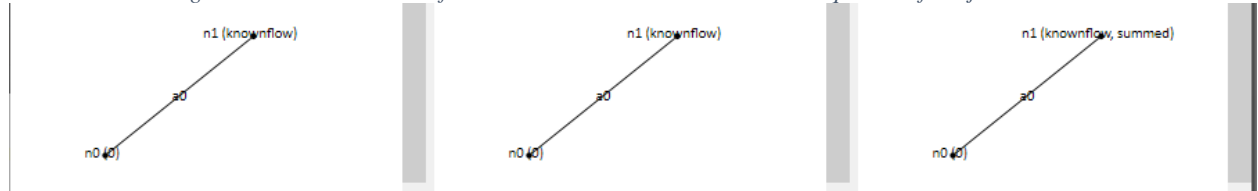


Figure 63. Rule to indicate if a node has been used in a summation equation of a 0 junction

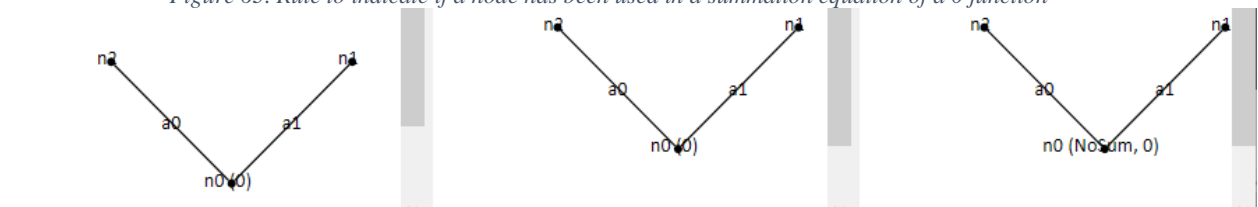


Figure 64. Rule to mark if node can be summed at 0 junction

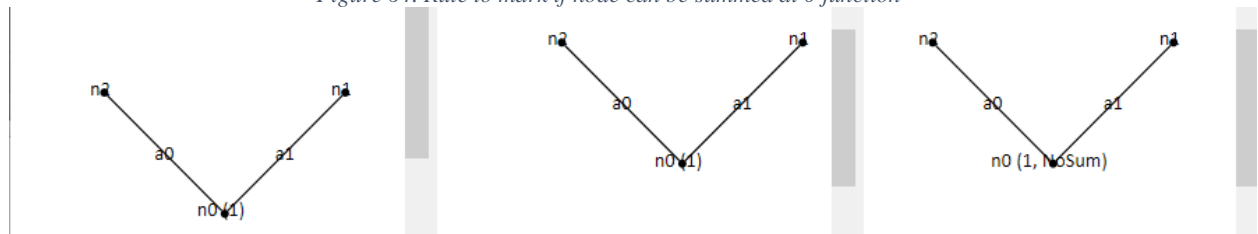


Figure 65. Rule to mark if node can be summed at 0 junction

Summation can only occur if only 1 node has an unknown flow, if two or more exist, then the summation ruleset cannot be applied, so the node is marked with “NoSum”.

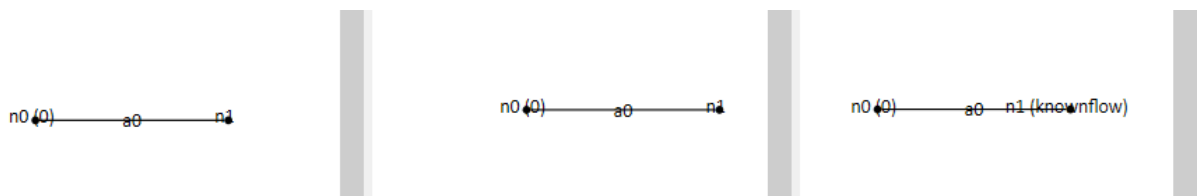


Figure 66. Rule to apply the summation equation to a node at a 0 junction

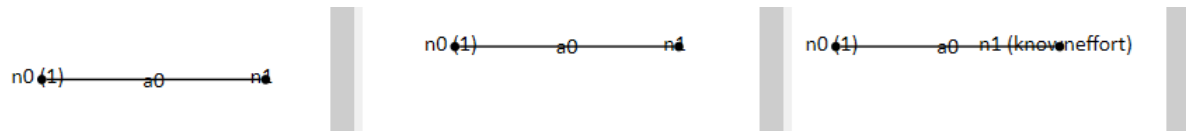


Figure 67. Rule to apply the summation equation to at node at a 1 junction

AUTHORSHIP

Corey Alicchio

Justin Vitiello

Callum Taylor

calicchio1@yahoo.com

vitiellojs@gmail.com

callumtaylor@icloud.com

The table below describes which aspects of the report each team member contributed to outlined by section. Primary editing was done by Callum Taylor, with each team member reviewing all the sections.

Abstract.....	Alicchio, Taylor, Vitiello
Executive summary.....	Vitiello
Introduction.....	Vitiello
Literature Review.....	Alicchio, Vitiello
Objectives:	Alicchio
Design and Implementation	
A. Automated Creation of SolidWorks Assemblies	
I. Read Data from User Entry.....	Alicchio
II. Interpreting Graph XML Data	Taylor
III. Generate Gears, Shafts and Bearings.....	Alicchio
IV. Creating an Assembly.....	Alicchio
V. Results.....	Alicchio
B. Extracting Bond Graphs from SolidWorks Assemblies	
I. Reading part data	Vitiello
II. Creating System Graphs	Vitiello
III. SolidWorks Identification Results	Vitiello
C. Analyzing Bond graphs	
I. State equations	Alicchio
II. Results.....	Alicchio
III. Direction of Bond graphs.....	Alicchio
IV. Bond Graph Results	Alicchio
Conclusions.....	Vitiello, Alicchio
A. Recommendations and Future Work	Vitiello, Alicchio