Norwegian University of Science and Technology
Knowledge Based Engineering
Knowledge Fusion-Siemens NX
Group: Marco Antonio Merola and Álvaro Ortega

Second Assignment

Gear Train Design

Trondheim, October 2022

Problem definition and Approach

In the world of automation the fact of minimizing costs is the focus of the industries, so that a simple model can work as a template so it can be automated to generate more complex ones. By using Siemens NX and NX Open for Python we have the challenge of trying to extend a solution that generates chains of gears changing only the parameters that are taken. The thing is what parameters shall we take as input so that the solution can be extendable?

In the case of our design and along with the idea of creating different chains of gears by simply modifying the parameters, we consider that a generic structure for a gear must be created. After this step, the matter is about generalizing the creation of many instances of the gear, which is why our point of view consists of generating a “fabric” that creates the gear object with attributes such as the radius and places it in the correct position. The approach is to generate and place the required number of gears that the user plans to use as the main chain. After the main chain was generated the user can decide to add composed gears to the existing ones.

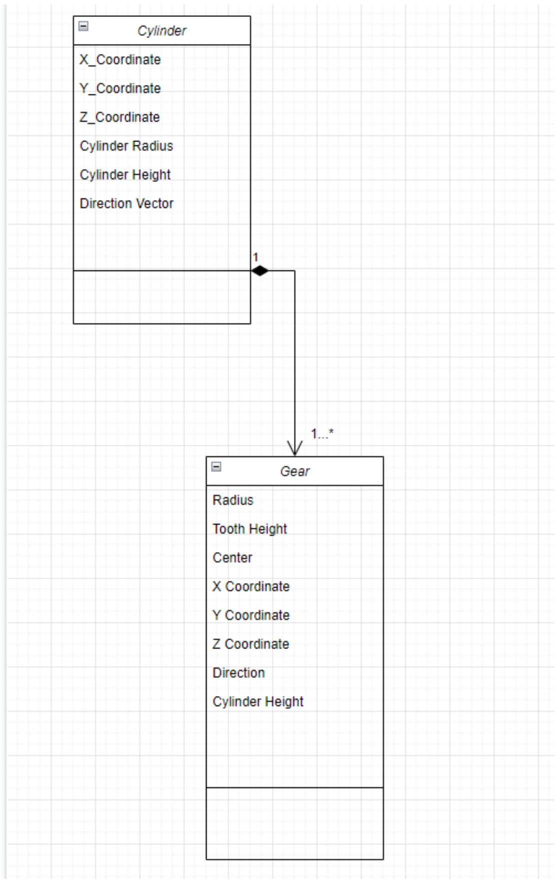
User view

The main purpose of our model is to generate adaptable and general aspects of the truck, so that it will be fast to change parameters and create many versions of trucks by using a sample. The product is supposed to be delivered to the manager of a company or maybe to a designer or supervisor of manufacturing processes, therefore we are trying to create the trucks in modular ways so that someone without programming experience or knowledge on basic software engineering tasks can easily modify the designs.

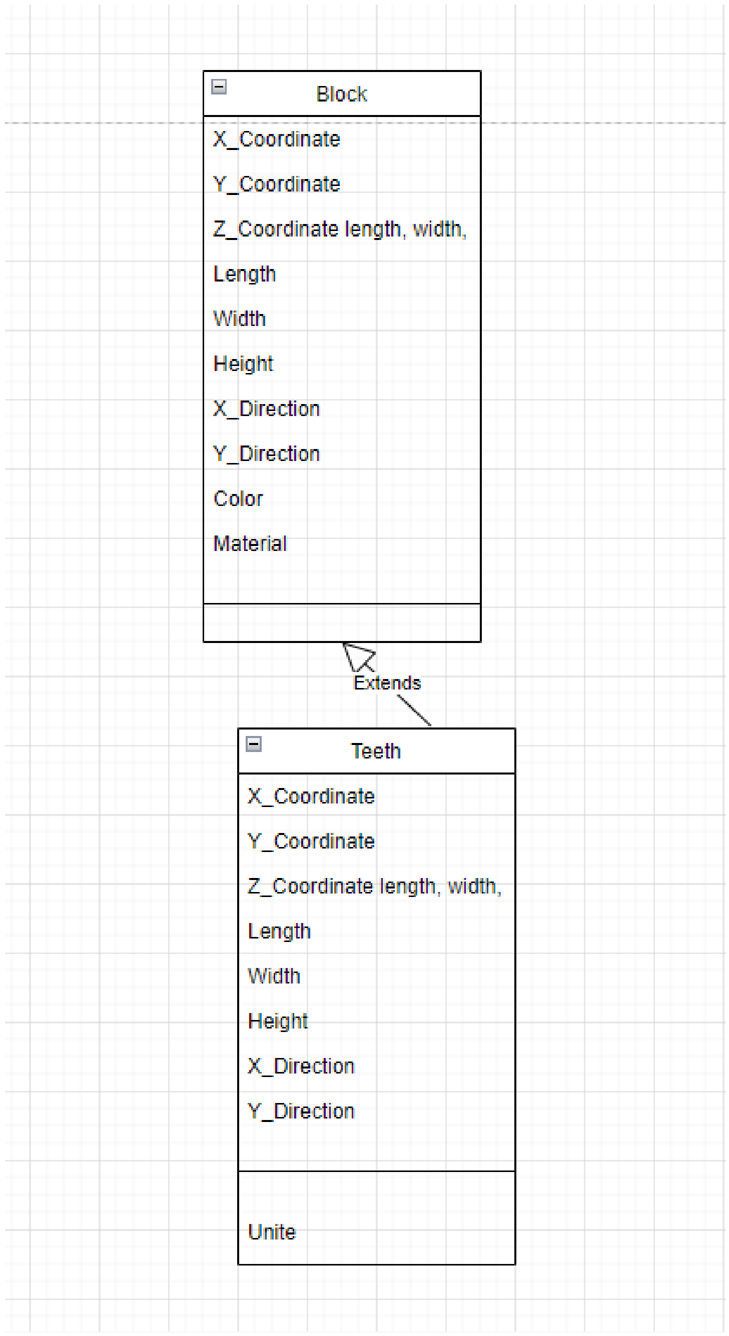
In the case of this tool the users can modify the size of the gears according to the features of the train. Furthermore, as the size of a train varies according to the purpose it has, the user is very likely to change the number of gears when generating another model from the original sample. Therefore, the user intends to adapt the characteristics of the model to other trains in the future, which means that it is important to easily generate different chains of gears by modifying a few parameters. Also, trains are required to pass physical and material tests, so according to some calculations the user is likely to delete some gears or change their size to meet the requirements for the construction of the train.

UML Diagram and Structure of The Code

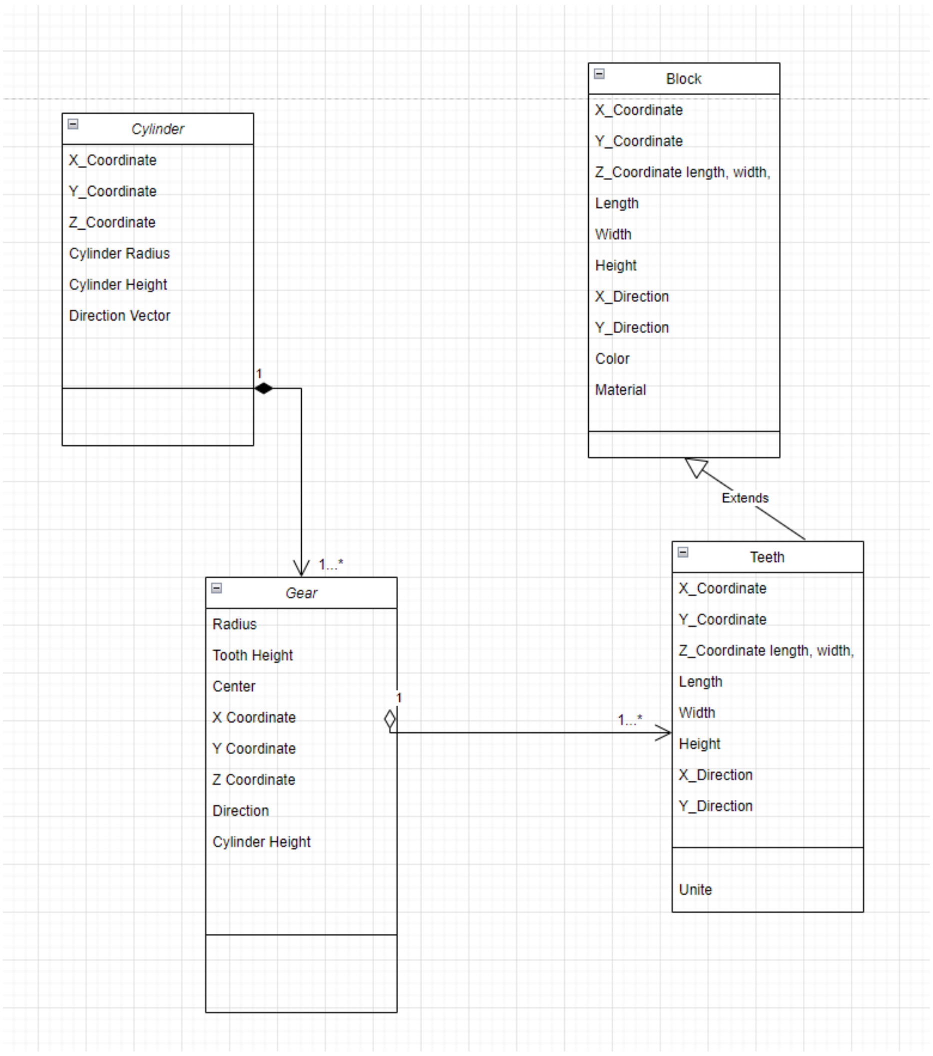
The first class consists of the Cylinder class, which generates the main structure for the gears, therefore we considered a composition relationship between the class gear and the class cylinder. As expected one cylinder can generate from one to N gears. It is a composition due to the fact that the child cannot exist independent of the parent.



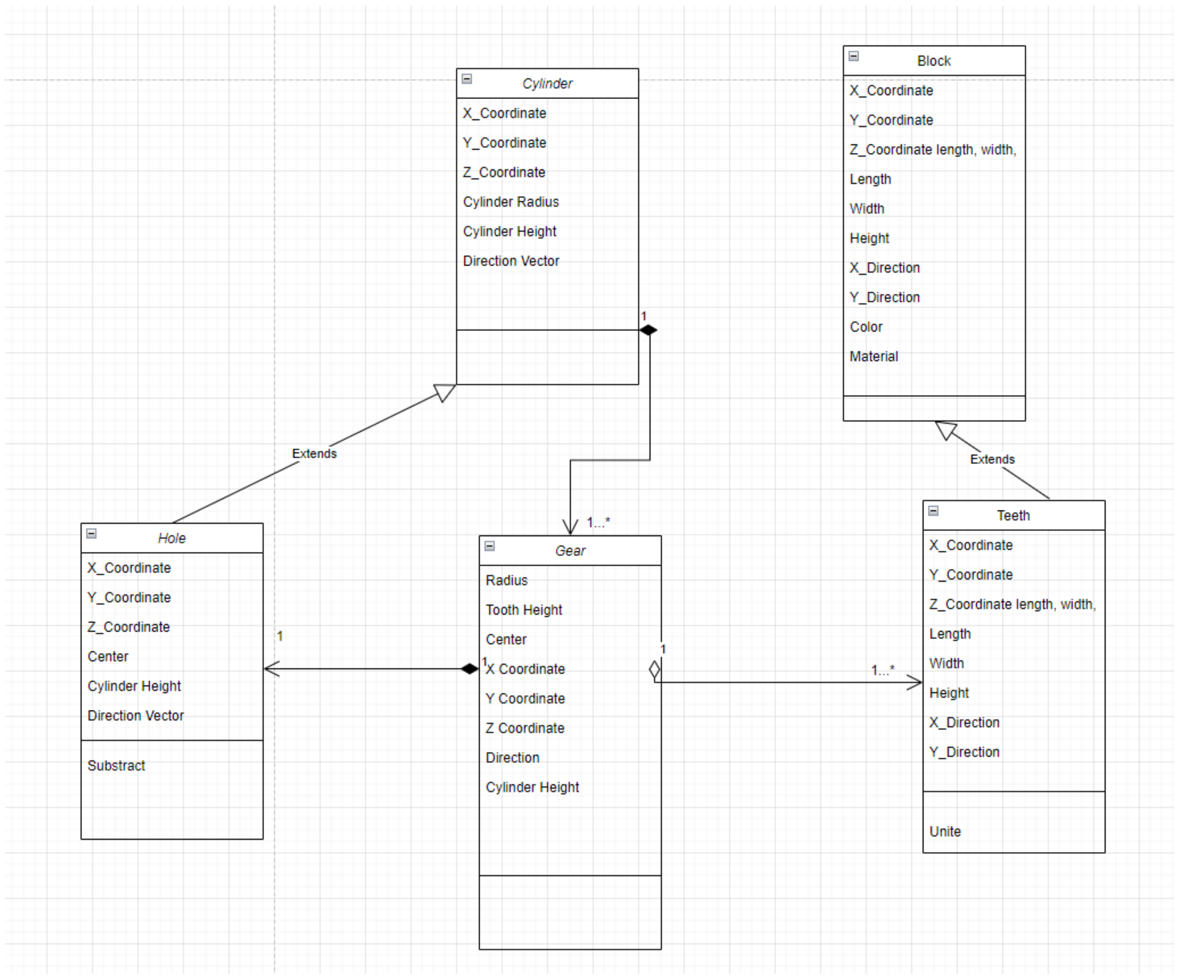
The teeth will be instances of the block, so we consider a tooth has the same attributes as a block (inheritance)



Now, as a tooth can exist independently, the relationship between the gear and the tooth is an aggregation

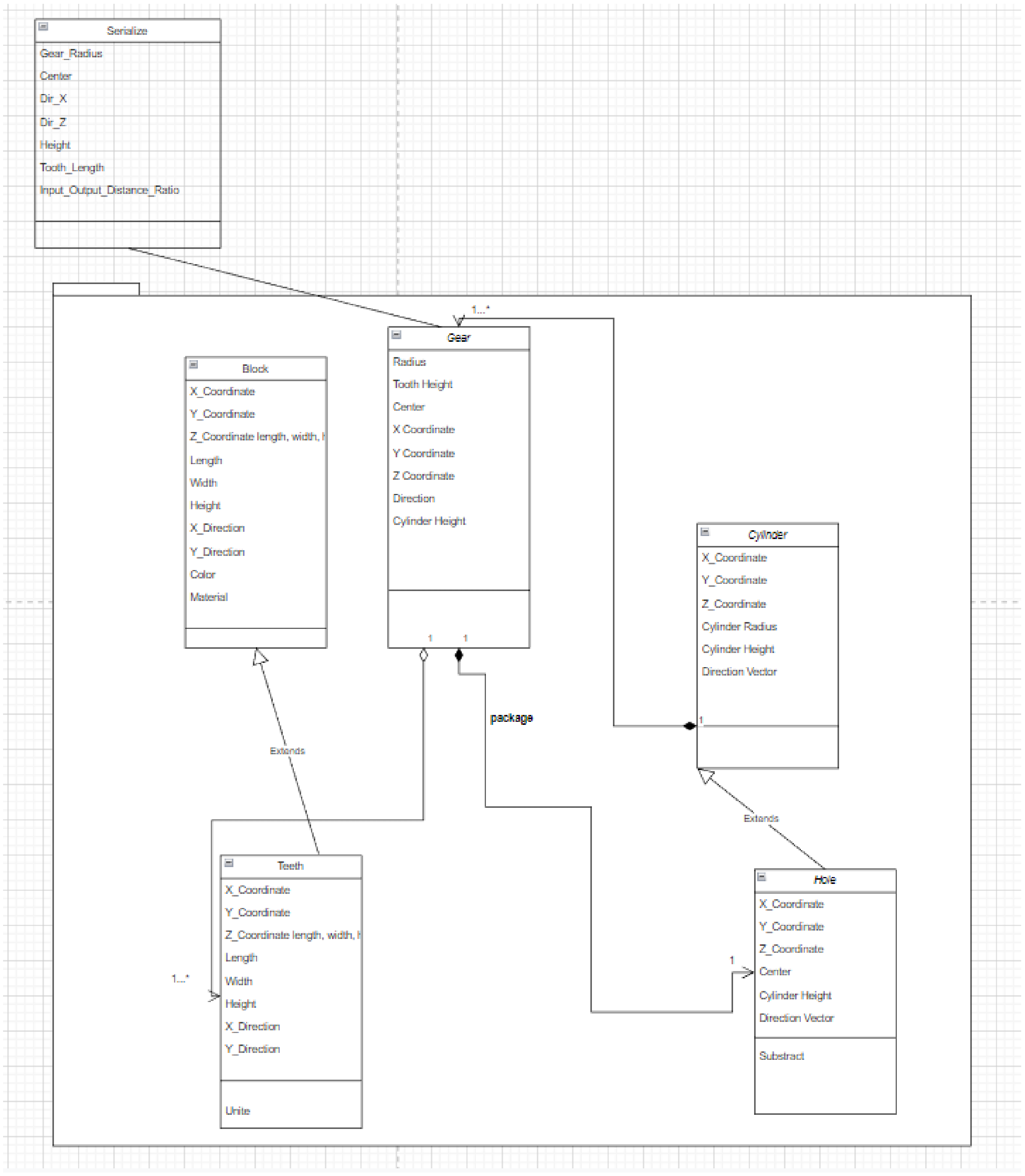


The same logic is applied to the hole inside the gear, however this hole cannot exist without the gear (it is a subtract operation) so it is a composition



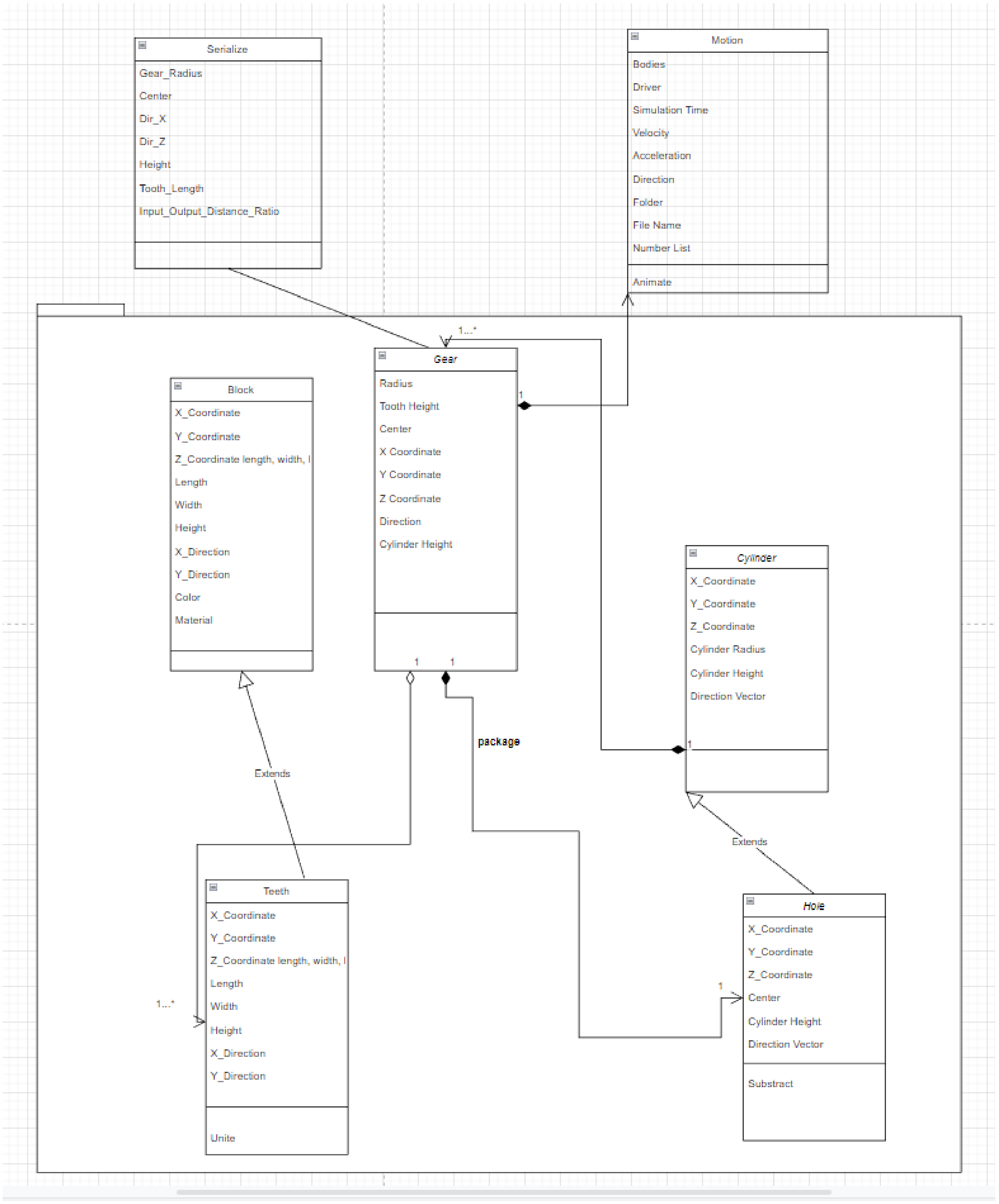
As all the classes above are called an generated by a superior class called Serialize (that includes a fabric function) the above classes will be part of a Package

Final UML Diagram Before Animation



Final UML Diagram After Animation

As the Animation cannot exist without the generated objects (gears), the animation class is in a composition relationship with Gear. We consider that one to N gears can generate one to N animations because the animations can change even if the gears are the same (for example if the velocity is changed)

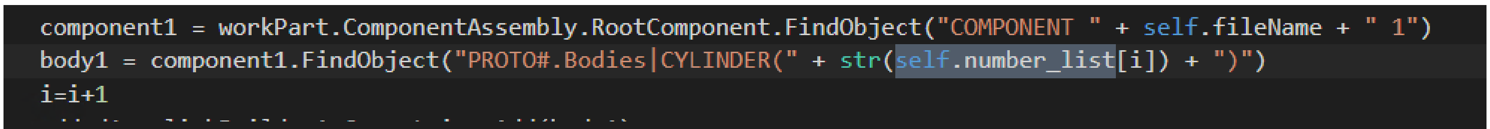


In order to generate a clear code, we decided to implement a general class that works like a factory does. This class has a generalization on how to put all the elements of the gear chain together, so this code organizes the deployment of the gears through the different axis according to the parameters the customer wants to use. To generate the gears the second class was created. The gears are generated inside the class Serialize in the function called fabric. The gears generated are objects from the class named General Gear.

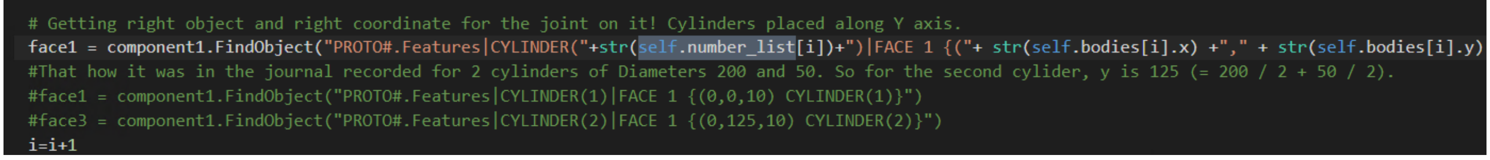
The class General Gear generates each individual gear, however this class has no responsibility in the deployment of the gears through the plane. This class creates the gears as a cylinder object, however through several mathematical computations we added teeth to the gears. These teeth are block objects. To define the number of teeth we defined a floor function relationship with the radius of the gears. Several aspects had to be changed, for example the cylinder radius, because we had to compute a subtraction for the original radius and the height of the tooth.

In order to properly receive the parameters of the customer, we created a text file where the client is supposed to introduce the parameters. We added a constraint so that the ratio between input and output gears is correct and through this interface the user can feel comfortable and does not need to know programming languages or computer science related fields. This is why we used the sys.argv function to collect the parameters passed by the customer, as the parameters are passed in plain text we constructed the arrays of arrays for the radius of the simple and compound gears according to "," separators. We also provided support for displaying messages through the NX interface using the DisplayMessage function.

Furthermore, regarding the animation part we decided to work with a list of bodies in order to parameterize as maximum as possible our design. To achieve our goal, we created a list with the generated gears, therefore every time a gear was created in the generate_gear function we added the new one to the list of objects. However, this caused several problems when we tried to incorporate the solution to the Motion file. During the course we saw that the cylinders were generated by order, however in our case we created gears, so in some cases the cylinder one existed, but the next gear was called cylinder 26 by NX. We realized that in our case the next gear id was 2 times the number of teeth plus 4. We created a parallel list to store the id values in order to pass them to the Motion file as a parameter: Here we show an example of how we used this parameter to our program:



```
component1 = workPart.ComponentAssembly.RootComponent.FindObject("COMPONENT " + self.fileName + " 1")
body1 = component1.FindObject("PROTO#.Bodies|CYLINDER(" + str(self.number_list[i]) + ")")
i=i+1
```



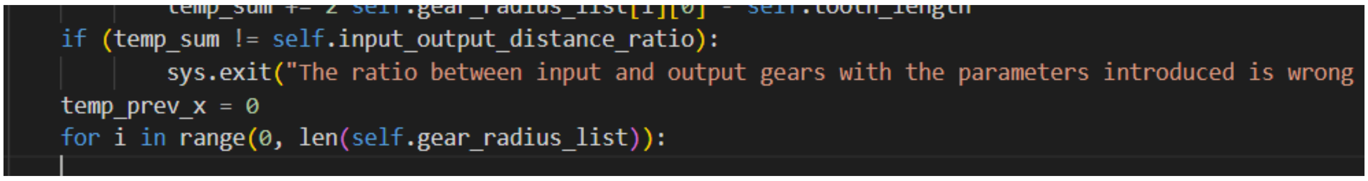
```
# Getting right object and right coordinate for the joint on it! Cylinders placed along Y axis.
face1 = component1.FindObject("PROTO#.Features|CYLINDER("+str(self.number_list[i])+")|FACE 1 {("+ str(self.bodies[i].x) +"," + str(self.bodies[i].y)
#That how it was in the journal recorded for 2 cylinders of Diameters 200 and 50. So for the second cylider, y is 125 (= 200 / 2 + 50 / 2).
#face1 = component1.FindObject("PROTO#.Features|CYLINDER(1)|FACE 1 {(0,0,10) CYLINDER(1)}")
#face3 = component1.FindObject("PROTO#.Features|CYLINDER(2)|FACE 1 {(0,125,10) CYLINDER(2)}")
i=i+1
```

Challenges

As mentioned in the previous sections, we focused on making our solution as extendable as possible. To achieve this goal we had to play a lot with Python functionalities and of course we had to face some programming issues to parameterize our solution. At first, we passed the parameters in the same file where the program was written, however we realized that this might not be the best option when presenting the solution to a customer. Therefore, we decided to generate a text file in which the customer can introduce the parameters. This solution of course caused many issues at first because we worked with arrays of arrays for the solution, so we had to convert the plain text into a valid Python vector.

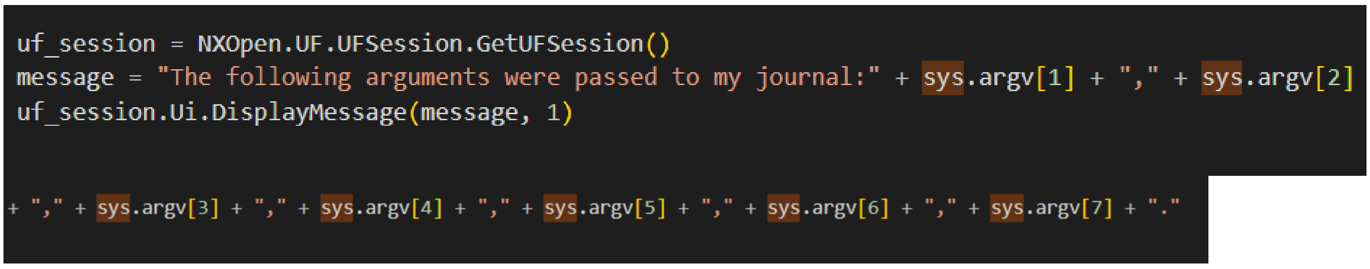
Another aspect we considered was learning functions of the system to show errors to the customer in case invalid parameters were given as input. In order to do so we used the function sys.exit to say that the ratio between input and output gears with the parameters introduced is wrong so the solution will not be logical.

Sys function used:



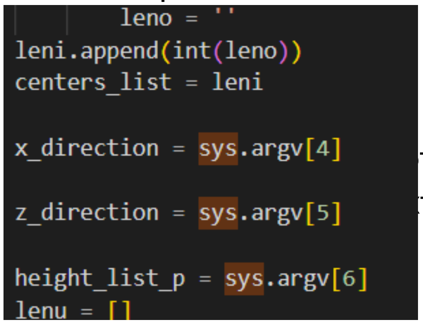
```
            temp_sum += 2 self.gear_radius_list[i][0] - self.tooth_length
    if (temp_sum != self.input_output_distance_ratio):
            sys.exit("The ratio between input and output gears with the parameters introduced is wrong
    temp_prev_x = 0
    for i in range(0, len(self.gear_radius_list)):
```

In order to make it more clear to the customer we also used this function to communicate through the NX user interface the parameters the user introduced:

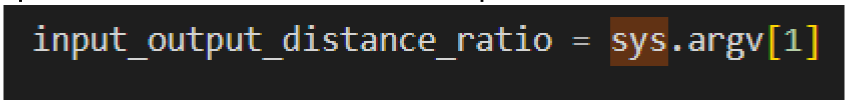


```
uf_session = NXOpen.UF.UFSession.GetUFSession()
message = "The following arguments were passed to my journal:" + sys.argv[1] + "," + sys.argv[2]
uf_session.Ui.DisplayMessage(message, 1)

+ "," + sys.argv[3] + "," + sys.argv[4] + "," + sys.argv[5] + "," + sys.argv[6] + "," + sys.argv[7] + "."
```

In the next picture we show examples of how we worked with this parameters:

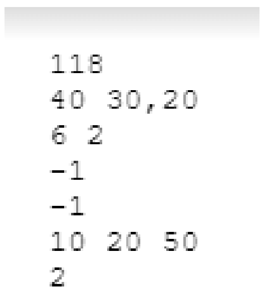


```
            leno = ''
leni.append(int(leno))
centers_list = leni

x_direction = sys.argv[4]

z_direction = sys.argv[5]

height_list_p = sys.argv[6]
lenu = []
```



```
input_output_distance_ratio = sys.argv[1]
```

t of conversion tasks because the customer introduces t file like this:



```
118
40 30,20
6 2
-1
-1
10 20 50
2
```

In these cases we had to identify the “,” to generate the arrays of gears and to identify which cases implied simple gears and compound gears. In the case of gears, we generated both arrays and arrays of arrays, which gave us so much work to do in order to combine this way of passing parameters to our solution.

Parameterizing the given direction in which the gears were generated gave us problems. At first it took time to realize that we could create a parameter called direction and this parameter can be one or minus one, so when generating the cylinders we passed this parameter to the z axis as the last argument the cylinder can take, so we also had to play with the shapes to generate our gear chain through the positive z axis or the negative z axis. A similar approach was done to generate the gears through a positive or negative x direction.

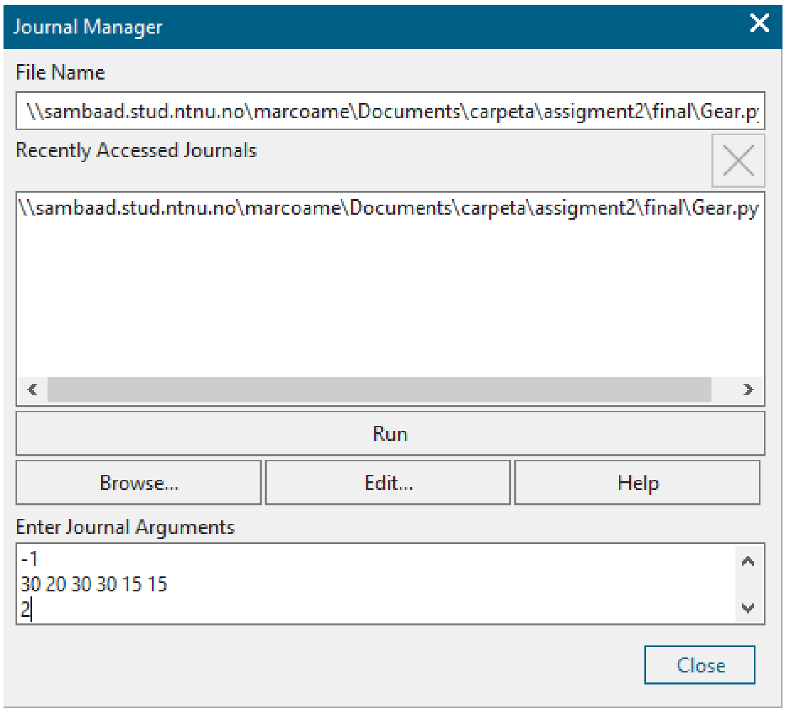
How extendible is the solution

This solution can be extendible to many types of gears a customer can generate due to the hard work that was put into designing a parameterized solution. The customer can decide an arbitrary number of gears and compound structures, which means it is possible to decide which gears will have compound structures and how many compound gears will exist. Obviously the customer can decide the radius of the gears to be generated and the holes.

Furthermore, the customer must provide an input output distance ratio in order to make sure that the solution is restricted to real cases. The client can decide the center of the gears, so they can be placed in the origin or in any other point in the plane, taking into account that compound gears must share the same center point. It is also possible to generate gears through the positive or negative x axis (right or left side) so that the chain of gears can be displayed in different directions. The same is done with the direction in the z axis if you want the compound gears to be generated on top or underneath. The height of the gears was also parameterized to customize wider or thinner gears. The tooth of the gears can be passed as a parameter as well. To summarize, the solution allows the clients to create a very customized design of each gear, regarding size features, tooth length, customized deployment through the axis x and z and the amount of simple and compound gears they want. Also, the solution is extendible for the animations, which means that our Motion file can take any number of gears to generate the animation. In fact, our Motion file works with bodies in general as it was explained during the lessons, but even if our code can generate any kind of gear train, we only considered cases with two compound gears.

Examples:

Parameters for the examples are passed here. To execute correctly our project, please introduce the content of the text file in the Journal Arguments (note that the parameters follow a strict format with separators and a very specific order taking into account commas and blank spaces)



File Name

\\sambaad.stud.ntnu.no\marcoame\Documents\carpeta\assigment2\final\Gear.p

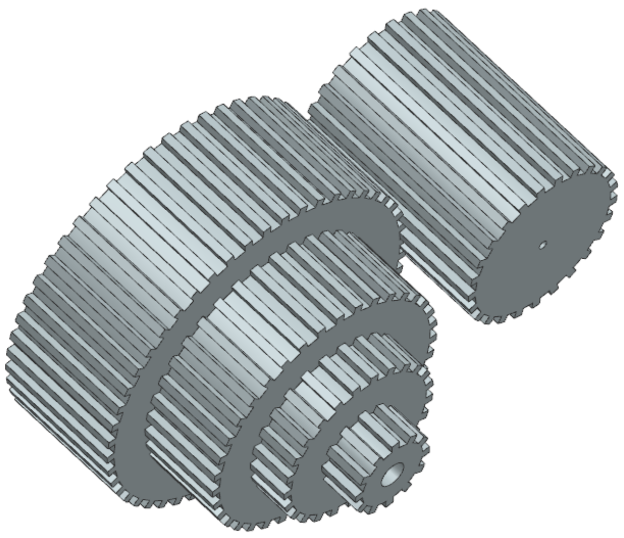
Recently Accessed Journals

\\sambaad.stud.ntnu.no\marcoame\Documents\carpeta\assigment2\final\Gear.py

Enter Journal Arguments

```
-1
30 20 30 30 15 15
2
```

Example 1



Parameters:

Ratio between input output: 118

Radius of the Gears: 40 30 20 10,20

Size of the Holes for the Different Base Gear Structures: 6 2

X Direction :-1

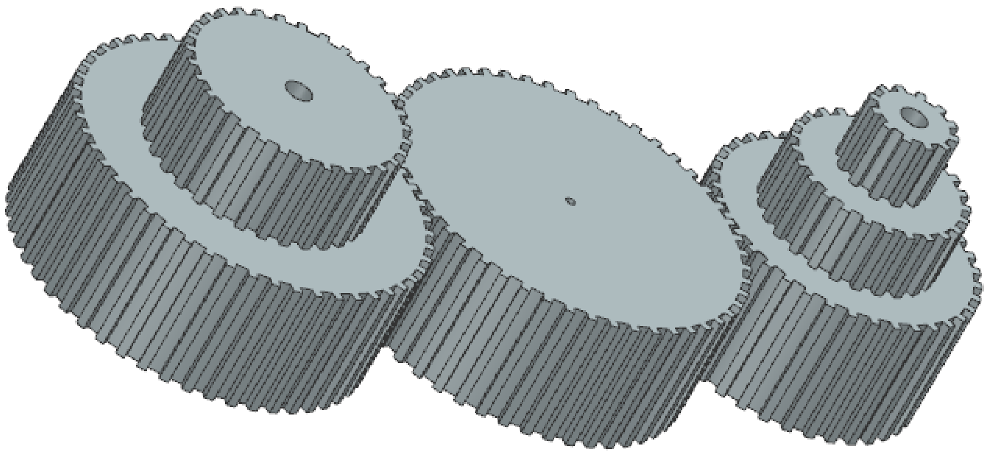
Z Direction: -1

Wide of the Gears :30 20 10 10 50

Tooth Length: 2

Direction :1

Example 2



Parameters:

Ratio between input output:216

Radius of the Gears: 40 25,40,30 20 10

Size of the Holes for the Different Base Gear Structures: 6 2 6

X Direction :1

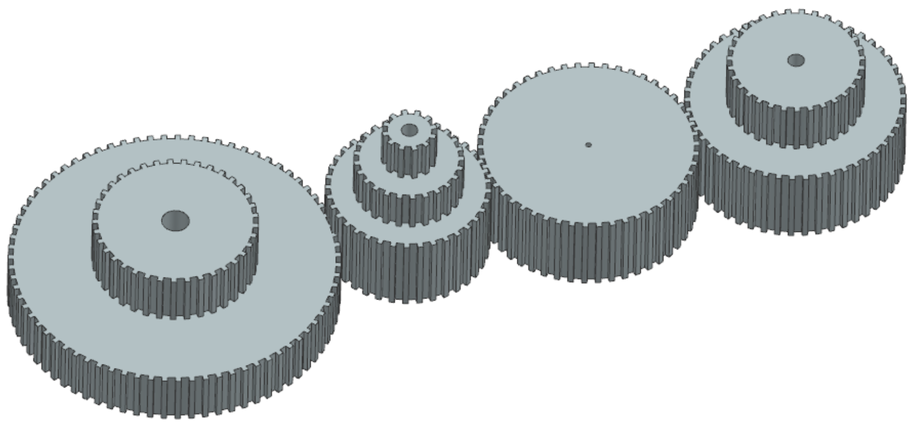
Z Direction: 1

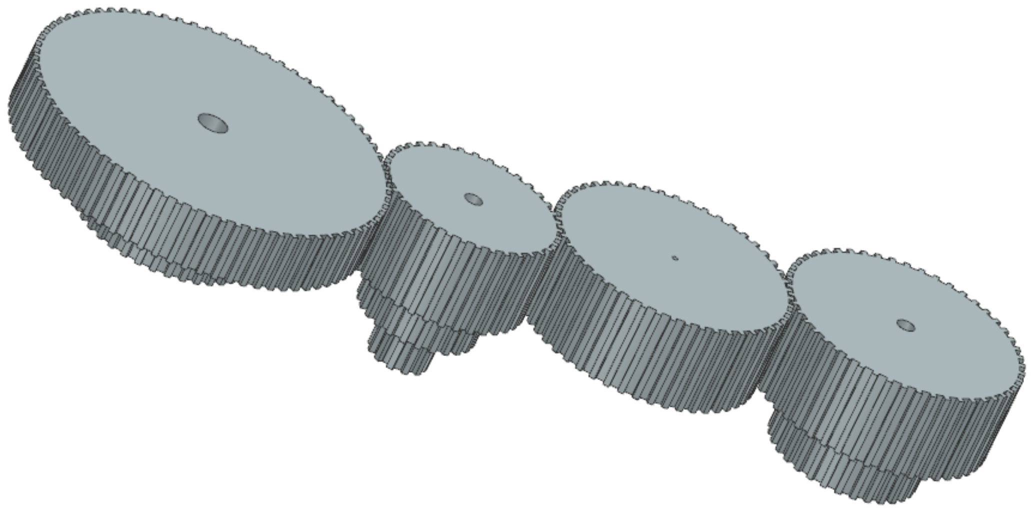
Wide of the Gears : 30 20 30 30 15 15

Tooth Length: 2

Direction: -1

Example 3





Parameters:

Ratio between input output:334

Radius of the Gears: 40 25,40,30 20 10,60 30

Size of the Holes for the Different Base Gear Structures: 6 2 6 10

X Direction :-1

Z Direction: -1

Wide of the Gears : 30 20 30 30 15 15 20 20

Tooth Length: 2

Direction: 1