Sectional Overhead Door Calculator

Magnús Þór Gestsson

Thesis of 30 ECTS credits
Master of Science (M.Sc) in Mechanical Engineering

July 2016

Sectional Overhead Door Calculator

Thesis of 30 ECTS credits submitted to the School of Science and Engineering
at Reykjavik University in partial fulfillment of
the requirements for the degree of
Master of Science (M.Sc) in Mechanical Engineering

July 2016

Supervisor:

Indriði Sævar Ríharðsson
Lektor, Reykjavik University, Iceland

Examiner:

Rúnar Unnþórsson
Professor, University of Iceland, Iceland

Sectional Overhead Door Calculator

Magnús Þór Gestsson

July 2016

Abstract

Sectional overhead doors have become the standard door type for garages and industrial drive-in doors. There is no complete software for small manufacturers to use for exact price calculations and manufacturing sheet creation given the type of door with all the optional components a customer wants. Python 3 with Tkinter was used to write an easy to use software with a user friendly graphical interface. The software calculates quantity and type of components, size of springs, price and writes out two csv files with the results. These two files are invoice with the information for the customer and manufacturing sheet for production. This software speeds up the price offer procedure significantly for the salesman. Mistakes from human error mostly due to input error is considerably less when moving information from invoice sent to customers to manufacturing sheet. Theoretical fatigue life of torsion springs were compared to tested data. For spring sizes $5mm$ to $7mm$ and fatigue life between 10.000 to 25.000 the results are within 30% of tested data. For higher fatigue life and spring sizes the error goes over 100% up to 400% in the most extreme case. This software can be modified to account for various hardware and panel manufacturers.

Sectional Overhead Door Calculator

Thesis of 30 ECTS credits submitted to the School of Science and Engineering
at Reykjavik University in partial fulfillment of
the requirements for the degree of
Master of Science (M.Sc) in Mechanical Engineering

July 2016

Student:

.....
Magnús Þór Gestsson

Supervisor:

.....
Indriði Sævar Ríkharðsson

Examiner:

.....
Rúnar Unnþórsson

.....
date

.....
Magnús Þór Gestsson
Master of Science

I dedicate this to my wonderful wife Katrin and my delightful daughter Iris Rut

Acknowledgments

The author expresses his gratitude to Ólafur Hreinn Jóhannesson sectional overhead door salesman at Vagnar og Þjónusta. For giving input on the user interface and knowledge on the hardware used in sectional overhead doors. Also to Gestur Bragi Magnússon CEO at Vagnar og Þjónusta for giving access to the components catalog used for price calculation.

Contents

List of Figures

List of Tables

Acronyms

FHL follow the roof high lift. 8, 14, 18, 20, 21, 24, 36

FLHR follow the roof low head room. 4, 20, 21, 22, 35, 36

FNL follow the roof normal lift. 4, 8, 20, 21, 22, 36

GUI graphical user interface. xi, 1, 8, 13, 14, 18, 43

HL high lift. 8, 14, 18, 20, 21, 24, 36

LHR low head room. 4, 20, 21, 22, 35, 36

NL normal lift. 3, 4, 8, 14, 20, 21, 22, 36

VL vertical lift. 4, 8, 14, 20, 21, 22, 24, 35, 36

List of Symbols

Symbol	Description	Value/Unit
E	Young's modulus	GPa
G	Modulus of rigidity	GPa
S_{ut}	Ultimate tensile strength	MPa
S_e	Fatigue limit	MPa
$S_e^{'}$	Unmodified fatigue limit	MPa
S_f	Infinite life strength	MPa
N	Newtons	Newtons
σ	Stress	MPa
$\boldsymbol{\tau}$	Torque	Nm
$\boldsymbol{r}$	Radius	mm

1 Introduction

Sectional overhead doors have become the standard door type for garages and drive in doors for industrial housing see figure 1 for an overview of the main components in these type of doors. Before the time of sectional overhead doors the standard door type was tilt up garage door. They can be seen in Iceland on older garages. Raynor was one of the first manufacturer of sectional overhead doors starting in 1944 [2] but they did not become very popular until later. The sectional door market can be split in two categories. First are enterprises that fill in the whole manufacturing chain. From manufacturing hardware and panel to the end product. A complete door with everything included. They then sell the doors to retailers. These type of companies include Raynor, Hörmann, Richard Wilcox and many more. The second category has a few companies that fill the manufacturing chain. There are the manufacturing wholesales that create the hardware and panels. Typically they are not the same corporation. They do not sell directly to the end user. Nor do they sell complete doors. Corporations that buy from them are in competition with each other and are the ones that finish the manufacturing specific to what their customer wants. They often brand the doors under their own name. In Europe there are two major manufacturers of hardware Doco and FlexiForce. They do not manufacture panels but point to specific panel manufacturers that they make hardware for. Hardware in sectional overhead doors are all components in the door other then the panel. This software is mainly built for companies that buy hardware and panels. There are many hardware components in each door. They come in different types and numbers according to the type of door. It can be time consuming and difficult to calculate all components of these doors if done manually. It also increases the danger of input or calculation error. There is no commercially available software that calculates all hardware components of a sectional overhead door. One that comes close to it is Create RSC software used by FlexiForce [3]. However that is difficult to operate. The problem is due to many various hardware lines available and a huge variety of optional items that the program needs to address with many input fields. It does not calculate the springs. They use a different software for spring calculation named SpringForce. There are several torsion spring calculators available. Most of them are focused on small application like a mouse trap. They will not calculate the torque needed for a given weight and height of a door and are thus unpractical to use for sectional overhead door spring calculation. There are a few torsion spring calculators available for overhead sectional doors. They are not free to use with the exception of SpringForce from FlexiForce. The author did not find any other free torsion spring calculators for sectional overhead doors. Using the SpringForce calculator gives the size and type of springs that the salesman needs to manually enter into the sales system to get the price. This is redundant and can be quite time consuming when calculating the price of many doors.

This software solves these problems by combining these factors within one package. Through a user friendly but robust GUI the salesman can enter the values for a specific door and get all relevant data about it. Size and quantity of all hardware components and

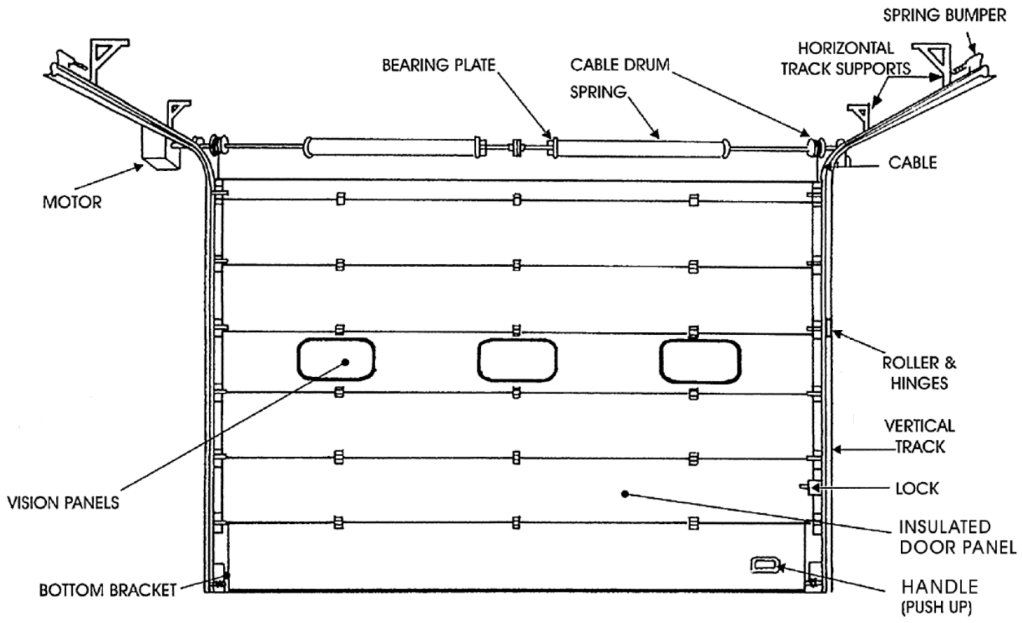

Figure 1: Sectional overhead door normal lift
[4]

panel are calculated. It prints the results in two simple .csv files. Invoice for the customer and manufacturing sheet for production. By creating these at the same time the possibility of input error by the salesman when transferring information from invoice to manufacturing sheet is removed. If the customer accepts the invoice the manufacturing sheet is printed. Thus it can be said that the customer reviews the manufacturing sheet for optional items and sizes as that information is mirrored in the invoice. To keep cost of stock down most companies have chosen their panel and hardware manufacturing lines. In this setup of the software two lines from FlexiForce have are chosen, industrial and stainless steel. For panels the H series from Epco which is a European sectional panel manufacturer is selected. These selections are utilized to simplify the GUI. Limiting the hardware lines does not limit the number of type selection or functionality of the doors. Different hardware lines are bound to different panels, safety devices and aesthetics. Most essential items are in the same ratio to the size of the door. Thus the output can be modified to account for different hardware lines. The code was written with this in mind.

1.1 Spring calculation

The most technical aspect of these doors are the springs. They are the only component in the door that is sold with a customizable fatigue life. It is common practice to not allow the spring fatigue life to go under 10.000 cycles. This can be seen in most spring calculators aimed at overhead doors. The most common range of fatigue life is between 10^4 and 10^5 [3] this is considered to be high life cycle [5]. The modified Goodman criteria can give fairly good results in that range [6] and is thus used in the theoretical calculations. This theoretical life is also compared to test data for the springs. This test data comes from FlexiForce. The test data was used for fatigue life of springs in the software. It is recommended to use test data when available [6]. This data is for the same setup and from the same manufacturer. So it is safe to assume it is more accurate than the theoretical life.

1.2 Sectional overhead doors

There are a few types of sectional overhead doors. What has the most effect on the functionality of the door is track type. Type of track changes how the door opens. There are many types of panel available. Different thickness and look. This software uses one line of panels from Epco, H-series with two sizes 610mm and 488mm height. And as previously stated two lines of hardware from FlexiForce. This is done to recreate a situation where a company wants to minimize item stock. This cutback on various lines of hardware and panels do not result in limited functionality of doors explained in the following section.

Main components of a sectional overhead door are the shaft system which includes torsion springs, lifting cable, bearing plates and the shaft that holds it all together. The shaft systems job is to make the door virtually weight less and works like a counter weight against the weight of the panel and all hardware bolted and screwed on it. There are two separate tracks a vertical track bolted on the wall of the daylight opening and horizontal track mounted on the ceiling with special brackets. These tracks hold the door in its place. Rollers connect the tracks and door together. Rollers slide within the tracks and are bolted with special roller brackets to the intersection of each panel. The hardware bolted or screwed on the panel is the same for all types of tracks with the exception of bottom and top rollers brackets that are located as their name indicates on the top and bottom of the door.

1.2.1 Normal lift

normal lift (NL) doors have the shaft system located above the door opening. Top roller of the door is located in a 300mm radius bending that connects vertical and horizontal tracks see figure 2. The door slides ninety degrees from the surface of the opening. Head room is a very important measurement in sectional overhead doors. It is measured vertically from the top of the daylight opening to the lowest part of the roof or any obstacle above the daylight opening. This is the space where the shaft system is located on most types of

doors. There is another version of NL tracks that hold two separate tracks in the horizontal track. This extra track is not connected to the vertical track and is located above the horizontal track that does. This is called double horizontal track. This enables the head room to be smaller and all tracks under 3000mm in height come with this track by default this is due to hardware line selection for this software. This becomes optional in higher doors. Only the top roller comes into this extra track and all other rollers are in the lower track when the door is fully opened. By using this extra track the door does not have to lift up as high as it would in the lower track before it goes into horizontal orientation. The radius on this extra track is much higher then the lower track. Thus the top section starts to slide along the roof with minimal vertical lift.

1.2.2 Follow the roof normal lift

Second type is follow the roof normal lift (FNL) which is similar to NL except the door does not slide ninety degrees from the surface of the opening. The slope can range from ninety one to eighty degrees, see figure 3 for an example of FNL door. As the name indicates it is used in buildings where the roof is not horizontal but has some other slope and the track follows that slope. This track type also has double horizontal track for doors lower then 3000mm and the second track is to lower the head room needed. The second track is optional for higher doors.

1.2.3 Low head room

Third type of door is low head room (LHR) also called springs in rear. They are the same as NL but with the shaft system located back on the horizontal tracks see figure 4. A pulley system is located on the junction of the horizontal and vertical tracks. This pulley system redirects the lifting cable to the rear of the horizontal track. This type is used where the head room above the daylight opening is under the minimum head room that the NL type needs. The minimum head room allowed for LHR is 86mm [3] but can be higher for large doors. This track type requires double horizontal tracks in all sizes. And in figure 4 one can see the difference from NL mainly how the top roller is located in the second track on the horizontal track. The pulley system is located in between the two tracks.

1.2.4 Follow the roof low head room

Fourth type is follow the roof low head room (FLHR) it is a combination of LHR and FNL. It has the spring system mounted on the rear of horizontal tracks like LHR. It also follow the slope of the roof like FNL does. FLHR requires double horizontal track for all heights. Minimum head room is 86mm but can be higher for large doors and doors with high slope.

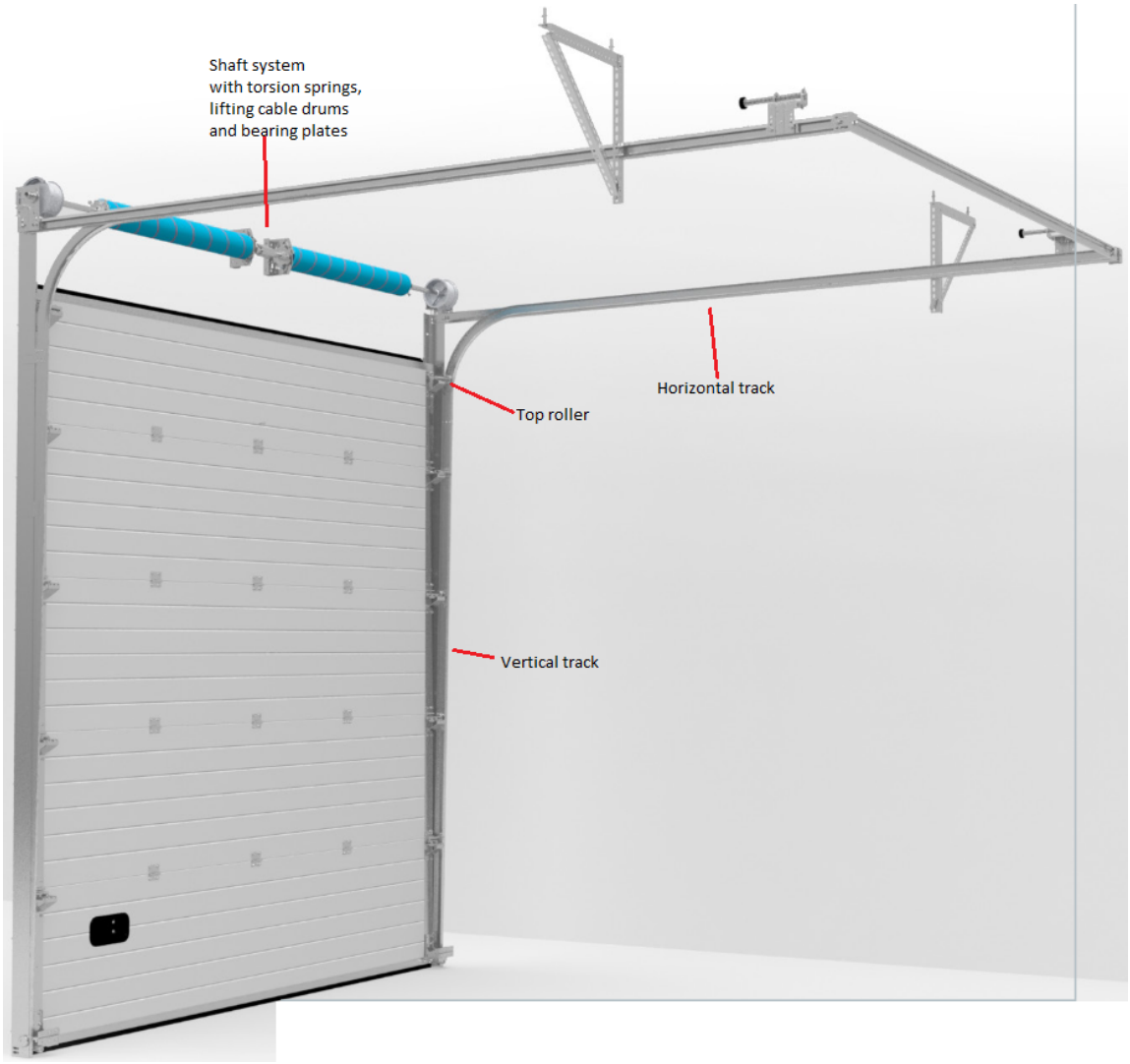

Figure 2: Normal lift sectional overhead door
[3]

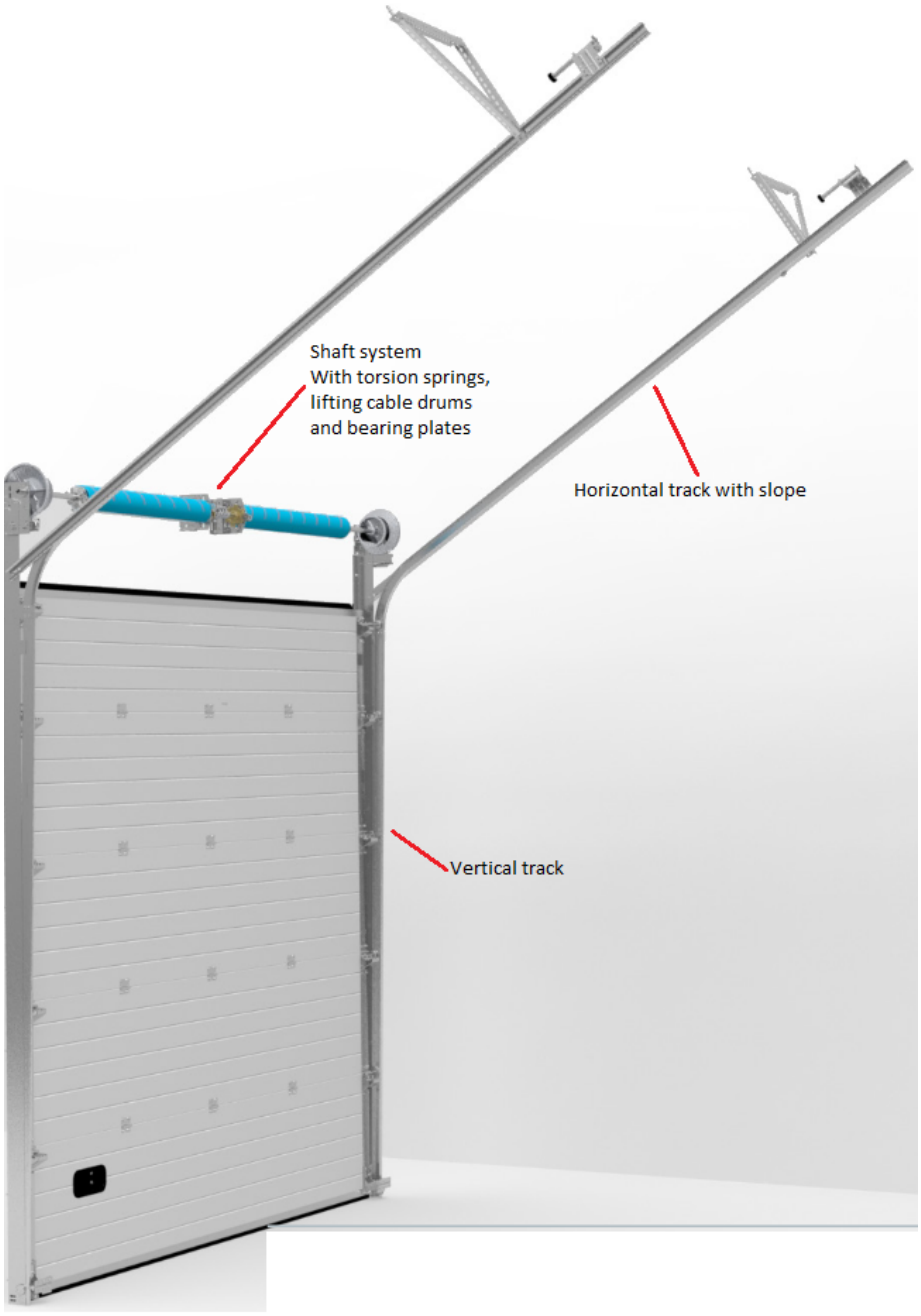

Figure 3: Follow the roof normal lift sectional overhead door [3]

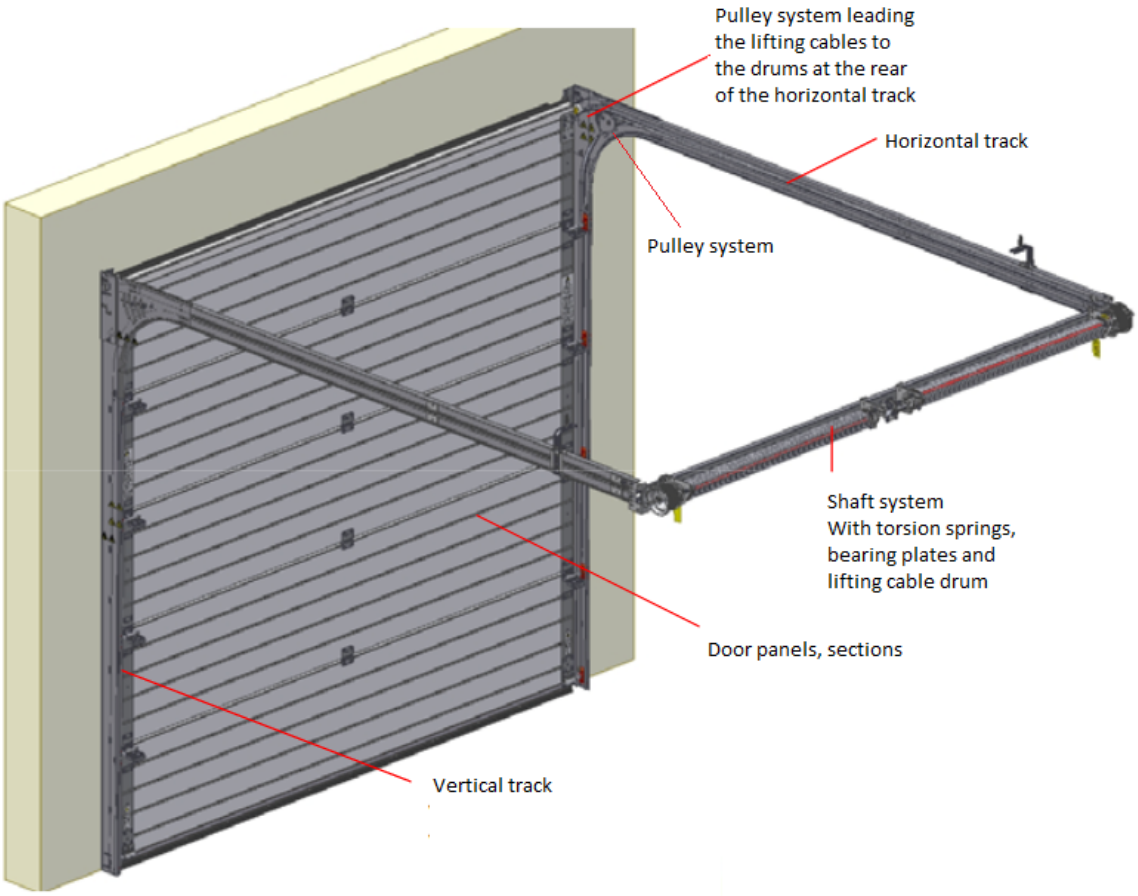

Figure 4: Low head room lift sectional overhead door
[3]

1.2.5 Vertical lift

Fifth type of door is vertical lift (VL) it has no horizontal track. As the name implies the door is lifted vertically from the daylight opening with a slight slope from the wall. Spring system is located on the top of vertical tracks or above them, see figure 5. It is used mostly in large warehouses where the ceiling height is high. VL requires minimum head room of at least the same height as the daylight opening height.

1.2.6 High lift

Sixth type of door is high lift (HL) it is a combination of NL and VL. The door lifts by a certain amount vertically before going ninety degrees from the daylight opening surface or in other words before it goes into the horizontal track, see figure 6. This type is used when the ceiling height is not enough for VL. And it is undesirable to have the horizontal tracks mid air in the building which is what happens if NL is used with large head room.

1.2.7 Follow the roof high lift

Seventh type of door is follow the roof high lift (FHL) it is a combination of HL and FNL. The door lifts by a certain amount vertically before going to the horizontal track. But now the horizontal track is not ninety degrees from the daylight opening surface, see figure 7. Like in FNL the slope can range from ninety one degrees to almost vertical.

1.3 Scope

The end result is a software that calculates and exports most aspects relevant to sectional overhead doors herein called the software. The software calculates all components they're quantity and which type. It gives exact price for each calculated door. It creates invoice with relevant information for the customer and a manufacturing sheet with relevant data for the manufacturing department. In the software relevant limits of component are used to select the right one for each door. All user input is trough a easy to use GUI.

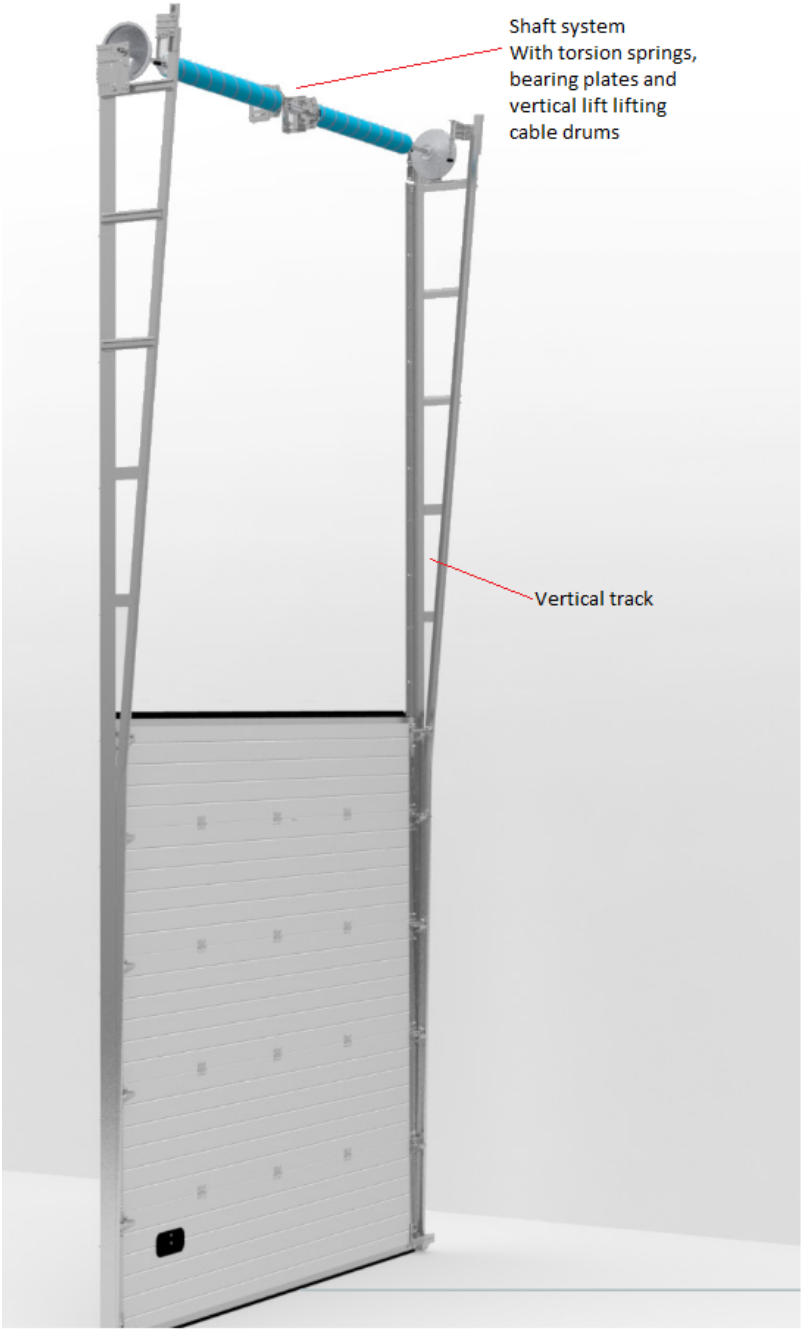

Figure 5: Vertical lift sectional overhead door
[3]

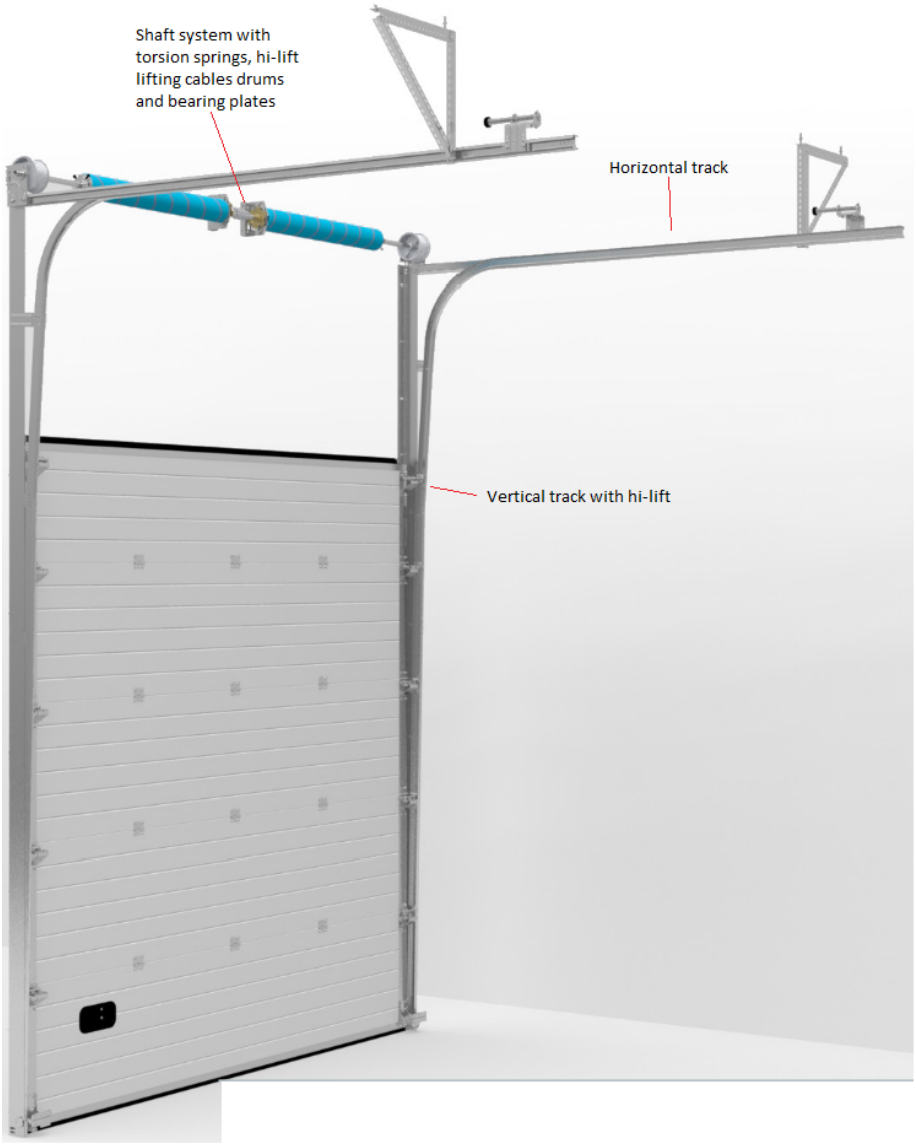

Figure 6: High lift sectional overhead door
[3]

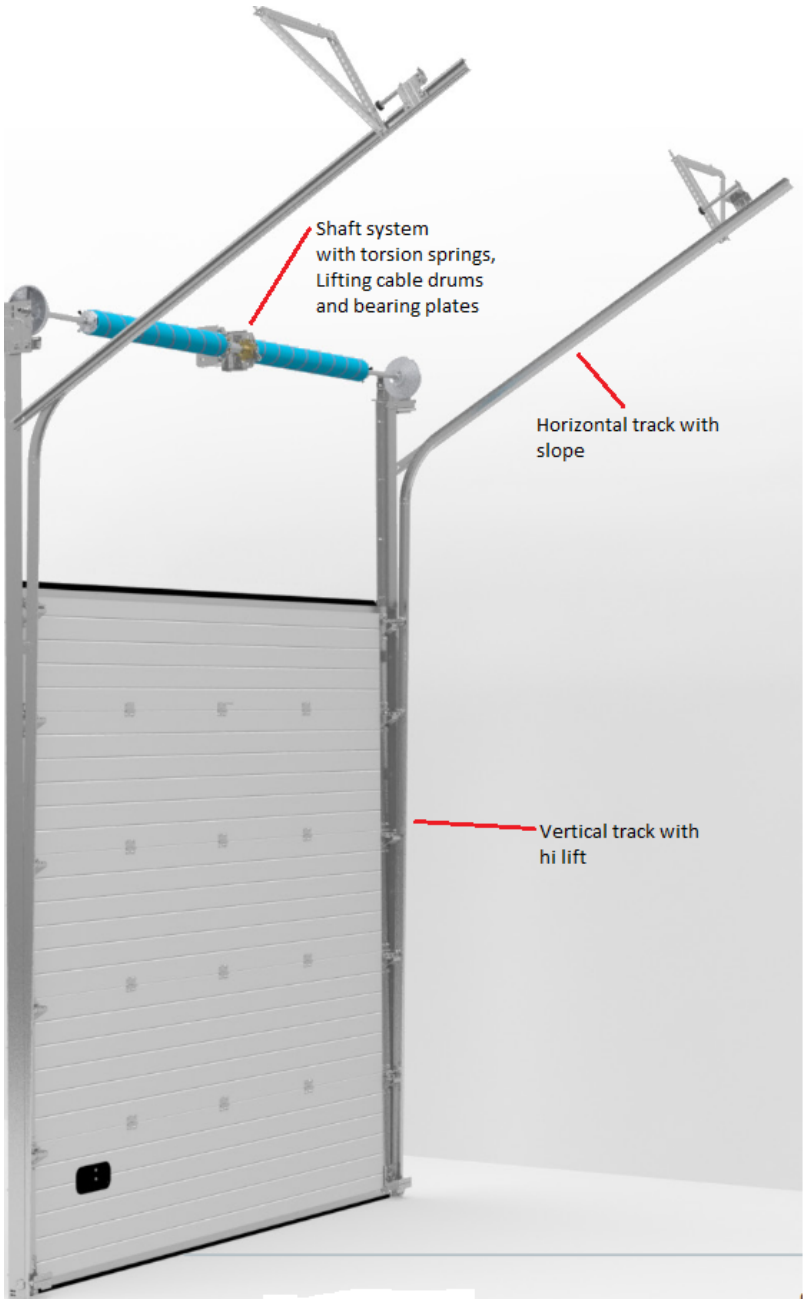

Figure 7: Follow the roof high lift sectional overhead door [3]

2 Methodology

The software was created with PyDev IDE for Eclipse with Python 3.5.1. The GUI was written in Tkinter. It was chosen as there is good documentation about it and comes bundled with most python 3 distributions. Many functions are used to give each one a clear objective. This makes maintenance and changing the software easier.

2.1 Software structure

A quick description of the hierarchy of the software. Main function calls GUI function and holds the window open. User inputs data into required fields. The GUI function validates the input by checking if the data entered is of correct type. This validation is done during runtime of the GUI as there are multiple calculations made after most inputs. This is explained in details in section 2.1.4. Then the user hits calculate then another set of validation is made to ensure all mandatory data has been entered. The GUI function calls all other functions to calculate all components and writes the csv files.

2.1.1 Database

Database of items used in the software are kept in a class. Variables are accessible when calling the class. If any changes are made in the manufacturing of the items or if other hardware lines from even different hardware manufacturers is chosen this class will need to be changed. It holds relevant data about the limiting functional factors about those items. For instance the maximum weight a 3mm lifting cable can hold, maximum length between intermediate hinges or weight per meter of 610mm panel section etc. This structure was chosen to ease maintenance of the software and make it easier to scale.

2.1.2 Price calculation

The prices of items are what most routinely change and is updated through reading a csv file. Csv file is exported from a accounting system called TOK that is sold and maintained by Advania. This accounting software was chosen as the author had access to it. The preferred way would be to read straight from the database that TOK runs on but due to security in the accounting software that was infeasible. TOK can however export the prices for items with its item identity number as .xml file that can easily be converted to .csv in excel without any other modification.

Read csv function scans the whole csv file and imports it into a dictionary. With item identity number as key and price as value. All other values in the csv file is omitted.

Price calculation function uses the item identity to find the relevant price. Within price calculation function is a set of item identity numbers that would also need to be changed if the software is supposed to use other hardware lines or hardware from different manufacturers. These identities would need to match those used in the relevant accounting

system given it can export into xml or csv files. Update of prices needs to be done when the prices from manufacturer change or the exchange rate changes. Unfortunately TOK can only export prices in its default currency. And thus its not possible to hold prices in the software in euros and multiply by the exchange rate. However it is possible to add a simple multiplier to account for exchange rate.

2.1.3 Write csv invoice and manufacturing sheet

Separate function was written to create the invoice and manufacturing sheet. The manufacturing sheet is much more detailed than the invoice. It holds all components of the door in what quantity and size, see table 1 for an example output csv file for 3000x3000mm NL door with 300mm head room. The relevant item identity numbers are given for each part that has more then one possible item filling that functionality. For instance wire drum has the item identity number as there are many possible wire drum to choose from. For bolts there is only one type so only the quantity is written. Another example of manufacturing sheet can be seen in table 2 but there all optional equipment has been chosen for a door 2500x2500mm. Now stainless is written next to all parts that are supposed to be stainless steel. A pull motor has been added and a pass door.

Information on the invoice is quite different see table 3 this is for the same door as the manufacturing sheet in table 1. There is minimum information on this sheet as there are no optional equipment taken. Another example invoice where all optional equipment is selected is in table 4. The customer does not need to know the amount or specific hardware type used. There needs to be perhaps more general information about the door and how it works but that is left for the salesman to add. It can easily be added within the write invoice function.

2.1.4 Graphical user interface

The GUI is made simple with as few input fields as possible but still being robust enough to address all types of sectional overhead doors and most accessories available. The interface is derived somewhat on SpringForce from FlexiForce. A screen-shot of the GUI can be seen in figure 8. There is some optional and mandatory input based on track type. Width and height are mandatory for all track types. Head room is mandatory for all track types except HL, FHL and VL. Pitch is mandatory for all track types that follow the roof. High lift is mandatory for both hight lift tracks. When the software is launched the GUI pops up with many default values. They can be seen in figure 8. All text input fields take integer numbers as valid input. These are width, height, head room, high lift, pitch, color RAL, windows and weight (kg). The drop down lists are locked for user input. Only values in the drop down can be chosen. The only drop down list that change is the manual drum selection field. It changes in accordance to track type. Values in the drop down fields can be seen in table 5. Where life is the fatigue life of springs and can be chosen between 10.000 to 100.000

Table 1: Manufacturing sheet no optional equipment 3000x3000mm

Name		
Address		
Contact		
Phone		
	Width	Height
Daylight size of opening	3000.0	3000.0
Cut down size	3005.0	
Number of 610mm sections	5	
Number of 488mm sections	0	
Head room	300	
Total height	3090	
Color of door RAL	9002	
Springs	2	(6.0, 50.8, 1068, 10.5)
Wire	3mm	4020.0
Shaft	701-3500Z	1
Bearing plate	310LH-RH	2
Wire drum	FFNL12	
Vertical track	3000	
Horizontal track	3000	
Seals on vertical track	2	
Top roller bracket	417	
Bottom roller bracket	428TAI	
Intermediate hinge	8	
Hinge	8	
Roller bracket on hinge	8	
Roller	12	short
Lock on door	1	
Door handle	1	
Rubber end stop	2	
Screw 6,3x35	80	
Bolt 6mm	16	
Bolt 8mm	4	
Track bolts	4	
6mm nut	20	
8mm nut	4	
Optional manual		

Table 2: Manufacturing sheet with optional equipment 2500x2500mm

Name			
Address			
Contact			
Phone			
	Width	Height	
Daylight size of opening	2500.0	2500.0	
Cut down size	2505.0		
Number of 610mm sections	1		
Number of 488mm sections	4		
Pass door in door.	Location: Left	Opening: Right hand	
Head room	300		
Saw of top section	102.0		
Total height	2500.0		
Color of door RAL	9002		
Windows	2		
Springs	2	(5.0, 50.8, 989, 12.2)	
Wire	3mm	3225.0	
Shaft	701-2750Z	1 PCS	
Bearing plate	310LH-RH	2	
Wire drum	FFNL10		
Vertical track	2500		
Horizontal track	2500		
Seals on vertical track	2		
Top roller bracket	417-304		
Bottom roller bracket	428TAI		
Intermediate hinge	8	Stainless	
Hinge	8	Stainless	
Roller bracket on hinge	8	Stainless	
Roller	12	short	Stainless
Door handle	1		
Rubber end stop	2		
Screw 6,3x35	100	Stainless	
Bolt 6mm	16	Stainless	
Bolt 8mm	4		
Track bolts	4		
6mm nut	20	Stainless	
8mm nut	4		
Optional manual			

Table 3: Invoice no optional equipment 3000x3000mm

Invoice for a sectional overhead door
Size of daylight opening Width: 3000.0mm Height: 3000.0mm
Size from highest point of daylight opening to lowest point on roof: 300mm (head room)
Track opening: Normal Lift (NL)
Color of door: 9002
Manual opening
Price of the door: 189.086,- kr

Table 4: Invoice with optional equipment 2500x2500mm

Invoice for a sectional overhead door
Size of daylight opening Width: 2500.0mm Height: 2500.0mm
Size from highest point of daylight opening to lowest point on roof: 300mm(head room)
Track opening: Normal Lift (NL)
Color of door: 9002
Stainless steel fittings
Number of windows: 2
Passdoor in door. Location: Left. Opening: Right hand
Automatic pull operator
Price of the door: 185.849,- kr

cycles. The seven types of tracks are in the track type drop down. Weight ($kg/m2$) holds a changeable weight per square meter ranging from 10,5 to 14. This should not be changed from the default value of 12 unless another panel is used or the door has extra equipment not accounted for in the software. Manual drum selection drop down changes with the track type specified. If the automatic drum selection box is ticked the software chooses the smallest available drum for that specific input. In table 5 all available drum selections are showed. There are three types of operation to open the door. Manual is simply when the user lifts and pulls the door manually. Track operator is the typical electric operator used in garages. It is connected to the door via bracket mounted on top of the top section. The operator comes with its own track that it follows that is typically located in between the horizontal tracks of the door. It comes in two varieties where it is either driven with chain or belt. Axle or shaft electric operators are located on the shaft of the door. It rotates the shaft in a similar way the spring does. This motor is typically used in industrial application and come in a wide variety of models. These motors tend to last longer then the track type operator. Number of springs is a drop down with one, two or four springs selectable. If more springs are required a duplex system might be needed where one spring is put inside another spring with high inside diameter. This setup is not addressed in the software as it is very rear and needs special equipment. Pass door location is for the location of the pass door in the sectional overhead door. It is required to have it located 500mm from either left or right side of the door. Position is seen from outside looking at the door. Pass door opening is how the door opens to the left or right. Pass doors cannot open inside due to sealing system used on them.

In Tkinter the grid geometry manager was used. It is best suited for stacking widgets in horizontal and vertical grid. The GUI is split into 10x4 grid with informative text in columns one and three. And user interactive fields in columns two and four as can be seen in figure 8. Most input fields trigger an action when the user has finished entering values, choosing item from drop down list or checking/unchecking checkboxes. The following list explains what functions each field calls and what they do. There are three kinds of triggers used. Selecting from a list in drop down menus and focusing out of input field. Some fields have multiple triggers.

1. When focusing out of width or height input fields the weight is calculated from area of the door. If empty string is in either field nothing happens. If non numeric string is in either field ValueError is thrown.

2. Focusing out of height the manual drum selection is changed. If the drum selected for instance is to small for the height it is taken from the drop down list and the next size that fits is chosen.

3. Focusing out of height or head room calculates number of sections and if the top section needs to be cut down. If either field is empty nothing happens. If non integer values are entered ValueError is thrown.

Table 5: Lists in GUI drop down menus

Life (cycles)	**Track type**	**Weight (kg/m2)**
10.000	Normal lift	Auto
15.000	Low head room	10,5
25.000	High lift	11
50.000	Vertical lift	11,5
100.000	Follow roof NL	12
	Follow roof LHR	12,5
	Follow room HL	13
		13,5
		14

Manual drum selection (Vertical lift)	**Manual drum selection (High lift)**	**Manual drum selection (Normal lift)**
FFVL11	FFHL54	FFNL10
FFVL18	FFHL120	FFNL12
FFVL28	FFHL164	FFNL18
		FFNL32

Type of operator	**Number of springs**	**Passdoor location**
Manual	1	Left
Track	2	Right
Axle	4	

Passdoor opening
Right hand
Left hand

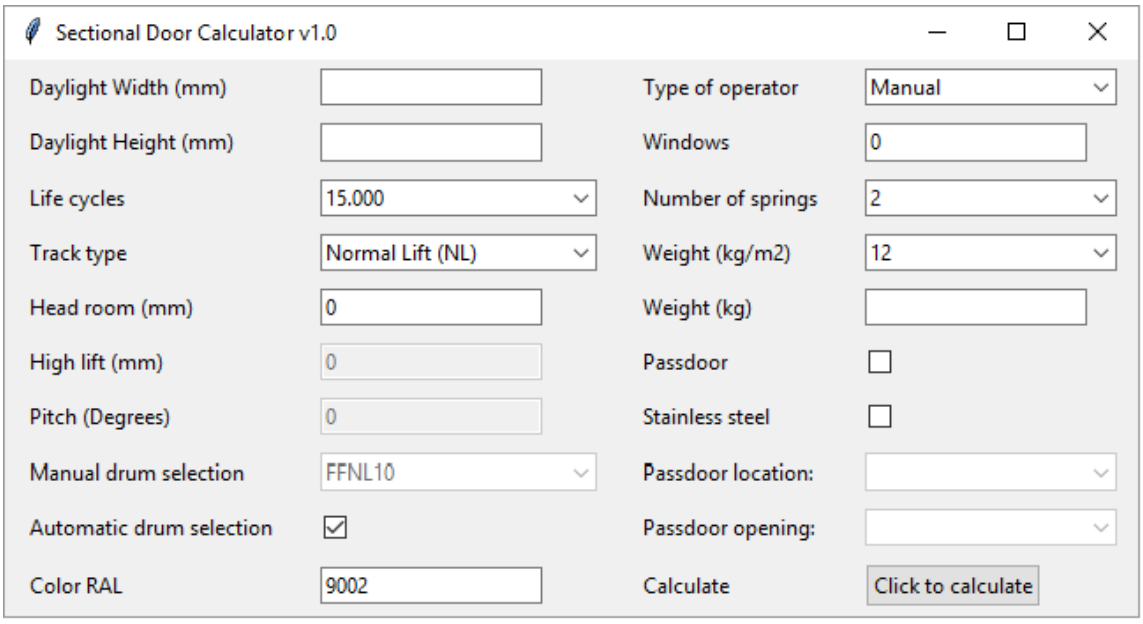

Figure 8: Graphical user interface for sectional door calculator

4. Selecting a track type in the drop down menu triggers a function that changes the GUI in accordance with track type selected. It calls a function when item in the drop down is selected.
   Selecting HL or FHL changes:

   (a) Head room input field becomes inactive.
   (b) High lift input field becomes active and sends its value to the head room field.
   (c) Pitch input field becomes active if track type is FHL.
   (d) Pitch input field becomes inactive if track type is HL with value 0.
   (e) Manual drum selection changes to high lift drums.

   Selecting VL changes:

   (a) Head room input field becomes inactive with value of height.
   (b) High lift input filed becomes inactive with value 0.
   (c) Pitch input field becomes inactive with value 0.
   (d) Manual drum selection changes to vertical lift drums.

   Selecting NL, FNL, LHR or FLHR changes:

(a) Head room input field becomes active.

(b) High lift input field becomes inactive with value 0.

(c) Pitch becomes inactive with value 0 for NL and LHR.

(d) Pitch becomes active for FNL and FLHR.

(e) Manual drum selection changes to normal lift drums.

5. Automatic drum selection check box controls if the user can change the drum in manual drum selection. If checked manual drum selection is inactive. If unchecked manual drum selection is active.

6. Weight (kg/m2) calls a function to calculate the weight when a new selection is made. If height or width is empty nothing happens.

7. Pass door check button enables pass door location and pass door opening drop down menus.

8. Click to calculate calls the functions to calculate all items, springs, price and write the results in csv files.

2.2 Torsion spring calculations

Springs are the most complex hardware component of the door. Spring fatigue life is what controls the endurance of doors as it is the only component in the door that has variable life cycles ranging from 10.000 to 100.000 cycles. The spring material used is EN-10270-1 SH cold drawn steel. There are seven types of track type and each needs its own set of calculation. The procedure is similar for most of them. The track types are grouped in three. First we have NL, FNL, LHR and FLHR that all share the same calculations with the exception of open door weight. Open door weight is how much mass of the door is pulled into the vertical track. For instance if the horizontal track has 20 degree slope all weight of the door contributes to open door weight. This is explained better in the following procedure. Second group is VL. Third group is HL and FHL. HL and FHL they can be thought of as a hybrid model of NL and VL. The first windings on the wire drum are similar to VL and the rest of the windings are like NL.

2.2.1 Procedure of calculating the torque and turns needed for each track type

1. Lifting cable diameter is found from weight of door.

2. Find the size of the bottom section either 610mm section or 488mm. The highest section is always in the bottom.

3. Open door weight is affected by the track type used. For NL and LHR the open door weight is only affected by weight of the bottom section as part of that section is in a bending that connects vertical track to horizontal this is a 300mm radius bending. For follow the roof track types weight of the entire door adds to the open door weight. That is affected by slope of the track. For high lift tracks part of the door that is still in vertical position when the door is open adds to open door weight. For vertical track type the entire weight of the door is the open door weight. These factors add to each other. For instance FHL has part of the door in vertical position and rest in a slope so the factors need to add.

4. Calculations for torque required and number of turns the spring needs to be wound is different for track types they are also grouped in three:
   For NL, LHR, FNL and FLHR the same procedure is used.

   (a) The number of turns are found from the height of the door and circumference of the lifting drum.

   (b) Maximum torque is found with equation 1. This torque is needed when the door is closed and the spring need to overcome all weight of the door. Where $\boldsymbol{r}_{\text{drum}}$ is radius of lifting cable drum, $\boldsymbol{r}_{\text{lifting cable}}$ is radius of lifting cable and $weight_{door}$ is the weight of door in newtons.

$$\tau_{max} = weight_{door} \times (\boldsymbol{r}_{\text{drum}} + \boldsymbol{r}_{\text{lifting cable}}) \tag{1}$$

   (c) Similarly minimum torque (τ_{min}) is calculated with equation 1 but weight of the door is now open door weight. This torque is required to hold the door fully opened. This is to counter the open door weight.

   (d) Torque per turn is found with equation 2

$$\tau_{\text{per turn}} = \frac{\tau_{max} - \tau_{min}}{\text{turns}} \tag{2}$$

   (e) Turns already calculated in a) do not account for the pretension found from the open door weight. So total turns are found with equation 3

$$turns_{total} = \frac{\text{turns} + (weight_{\text{open door}} \times (\boldsymbol{r}_{\text{drum}} + \boldsymbol{r}_{\text{lifting cable}}))}{\tau_{\text{per turn}}} \tag{3}$$

   (f) Lifting cable length is found with equation 4

$$\text{Cable length} = \text{daylight height} + 2 \text{ safety wraps} + \text{head room} \tag{4}$$

   For VL a different procedure is used. The main difference is that the drum diameter changes with a specified rate for each winding in other words the drum is a spiral

Figure 9: Vertical lift lifting cable drum
[3]

see figure 9. Same force is required to hold the door fully opened and closed. To compensate for lost torque in springs the wire drum radius or torque arm gets smaller as the door opens. This results in the same force being applied to the door both when the door is fully opened and closed even though the torque from the springs are much less.

(a) A while loop is used instead of integration to find how many turns the drum needs to accommodate for the daylight height. This is not as accurate as integration but easier for the software. The difference is in the fractional turns. For integer turns the results are exact. The fractional turn is found with equation 8. The integer turns are found in the same while loop. Counter is used to keep track of how much wire length the drum is covering. While this counter is smaller then daylight height the loop continues. Equation 5 shows how count is increased. Also the torque arm (τ_{arm}) changes with each iteration see equation 6. Number of integer turns are also counted see equation 7. Here τ_{change} is the rate of torque change per wrap on the drum.

$$count = count + 2\pi(\tau_{arm} + \tau_{change}/2) \tag{5}$$

$$\tau_{arm} = \tau_{arm} + \tau_{change} \tag{6}$$

$$turns_{integer} = turns_{integer} + 1 \tag{7}$$

$$turns_{fractional} = \frac{height - count}{\pi(\tau_{change} + 2\tau_{arm})} \tag{8}$$

(b) Maximum and minimum torque is found with equations 9 and 10 respectively. Where τ_{low} is the lowest torque arm used on the drum i.e. the torque arm when the door is fully open. τ_{total} is the highest torque arm i.e. when the door is closed.

$$\tau_{max} = weight \times (r_{total} + \boldsymbol{r}_{\text{lifting cable}}) \tag{9}$$

$$\tau_{min} = weight \times (r_{low} + \boldsymbol{r}_{\text{lifting cable}}) \tag{10}$$

(c) Torque per turn is found with equation 2

(d) Pretension turns are found with equation 11

$$turns_{pretension} = \frac{\tau_{min}}{\tau_{\text{per turn}}} \tag{11}$$

(e) Lifting cable length is very important in VL tracks. If the cable is too long then the torque arm is too small. If it is too short the torque arm will be too big. The length is found by adding length from floor too shaft center with length needed to get too the correct torque arm. This length is found by subtracting height of daylight opening from length from shaft too floor and adding a specific length given with the drum. This specific length is kept in the database class for that specific drum.

For HL and FHL a hybrid model of the two procedures explained above is used. The vertical lift procedure is used for the high lift part and normal lift procedure for the rest of the length of the daylight height. High lift drums come with specific maximum vertical lift capacity and maximum capacity for normal lift height. The vertical lift capacity of the drum is a spiral like the vertical lift drum and normal lift capacity of the drum has a constant radius.

2.2.2 Procedure to calculate the spring diameter and length

1. The minimum diameter of spring wire is calculated with equation 15.

$$D = d_{inner} + d \tag{12}$$

$$C = \frac{D}{d} \text{ [6]} \tag{13}$$

$$K_i = \frac{4C^4 - C - 1}{4C(C-1)} \tag{14}$$

$$d = \left(\frac{32 M_{total} K_i}{\pi \sigma}\right)^{(1/3)} \tag{15}$$

Where C is the spring index found with equation 13, D is the mean coil diameter found with equation 12, d_{inner} is inner diameter of spring, d is spring wire diameter and σ is the bending stress for a round wire torsion spring. K_i is the inner stress correction factor found with equation 14. The minimum ultimate tensile strength is used when calculating the spring diameter. The springs are manufactured in accordance to EN-10270-1 SH standard. In this standard minimum ultimate tensile strength is given. This has been summarized in table 8. The maximum working stress in the spring wire is found with equation 16.

$$\sigma = K_i \frac{32 F r}{\pi d^3} \tag{16}$$

2. Equation 17 is rearranged into equation 18 to find number of active coils. The constant 10.2 is often set to 10.8 in tightly wound torsion springs to make up for friction. But the manufacturer of the spring does not do so in its calculator SpringForce and thus its not done here. It is assumed that this is done as a result from testing.

$$k' = \frac{d^4 E}{10.8 D N_a} \tag{17}$$

k' is the spring rate or torque per turn, N_a is the number of active coils

$$N_a = \frac{E d^4 2\pi}{64 D \tau_{\text{per turn}}} = \frac{E d^4}{10.2 D \tau_{\text{per turn}}} \tag{18}$$

3. The length of the spring is found with equation 19. The four extra coils are because of coils that are not active due to the spring fitting.

$$\text{length of spring} = d(N_a + 4) \tag{19}$$

If length of the spring exceeds door width minus length of wire drums and spring fitting then the spring is to long. If this happens an error is produced and the minimum cycle life needs to be lowered or the need to change to fewer springs is needed. Lowering fatigue cycle life results in smaller spring wire diameters to be allowed that result in shorter springs with the same spring rate. For example 4,5mm

spring wire diameter with 50.8mm inner diameter and 50.6 active coils has 2963,4 Nmm/turn and when wound 5,3 turns it has fatigue life of 10.000 cycles the total length of this spring is 227,7mm. Compared to a 5.0mm spring wire diameter with 50.8mm inner diameter and 77 active coils it has 2958,1 Nmm/turn and when wound 5,3 turns it has fatigue life of 50.000 cycles the total length of this spring is 385mm. These springs have a similar torque per turn and can thus be used on the same door setup with the exception of fatigue life. But the 5.0mm spring is longer then the 4.5mm. The software does not change the fatigue life and iterates to find a shorter spring it only catches the error and returns it. This problem can also be solved by using the same wire diameter of the spring but with a higher inner diameter. However in this software each spring wire diameter has a single inner diameter. This is done to minimize stock a company holds of springs.

2.2.3 Theoretical fatigue life for torsion springs

Two methods are used to find fatigue life of the springs. Theoretical life that is calculated from a few known parameters. Using test data provided by FlexiForce. There is not much difference between those two methods when some parameters are tweaked in the theoretical calculations.
Procedure to calculate theoretical life:

1. Alternating stress within the spring is found see equation 20 where σ_a is the alternating stress and σ is the stress calculated with equation 16.

$$\sigma_a = \frac{\sigma}{2} \tag{20}$$

2. Minimum tensile strength S_{ut} of the spring is found in table 8.

3. The endurance limit $S_e^{'}$ is found to be $700MPa$ for S_{ut} larger then $1400MPa$.

4. Surface endurance limit modifying factor k_a is found with equation 21. The springs are cold drawn thus a is 4.51 and b is -0.265. Size factor is not used because S_{ut} changes in accordance to diameter in table 8.

$$k_a = aS_{ut}^b = 4.51S_{ut}^{-0.265} \tag{21}$$

5. Modified endurance limit $S_e = S_e^{'} * k_a$

6. When working with high cycle fatigue, data has shown that there is a linear relation between bending stress within the spring and fatigue life when fatigue life is put in a log_{10} scale [6]. Stress is not transformed only the fatigue life. Equation 22 can be

used to estimate fatigue life between 10^3 and $10^6 - 10^7$. $10^6 - 10^7$ is the endurance limit (S_f). Meaning infinite life.

$$S_f = aN^b \tag{22}$$

Where N is cycles to failure and the constants a and b are defined by the points 10^3 $(S_f)_{10^3}$ and 10^6. With $(S_f)_{10^3} = fS_{ut}$. Substituting these point into equation 22 gives equation 23 and 24. f is fatigue strength fraction and is found in figure 6-18 in [6]. It is found to be 0.77.

$$a = \frac{(fS_{ut})^2}{S_e} \tag{23}$$

$$b = -\frac{1}{3}\log(\frac{fS_{ut}}{S_e}) \tag{24}$$

7. Rearranging equation 22 give equation 25. Thus the fatigue life can be calculated.

$$N = (\frac{\sigma_a}{a})^{\frac{1}{b}} \tag{25}$$

This procedure is used to calculate fatigue life of a 6mm spring wire with 50.8mm inner diameter and 110 active coils. The results are plotted on two graphs where bending stress is on the y axis and fatigue life on the x axis. In figure 10 a logarithmic transform has been made on the fatigue life. In figure 11 no transform has been made. In the logarithmic graph one can see the linear relation between bending stress and fatigue life as predicted.

2.2.4 Fatigue life from test data

Tables of life cycles vs turns and active coils is provided by FlexiForce [3] for different spring wire diameters, inner diameter and lengths. These tables are based on test data for these springs. They do not specify how these test were conducted nor what diameters were tested or how many. To get more detailed information on these test they must be purchased from institutions that specialize in fatigue tests. Results in these tables are used to calculate bending stress within the springs. Two graphs are plotted for each setup as was done in theoretical calculations. With bending stress on the y axis and life cycles on the x axis. Again logarithmic transform is done on life cycles on the first graph and untransformed on the second. These graphs can be seen in figures 12 and 13 respectively.

As can be seen in fig 12 where a 6mm spring wire with 50.8mm inner diameter and 110 active coils is tested. The life of this spring shows a linear relation between stress and life cycles when the life cycles are put in a logarithmic scale with a small deviation when going from $10^{4.7}$ to 10^5. This is in harmony with what is used to calculate the theoretical fatigue life. On the x axis is life cycles in logarithmic scale and in y axis is bending stress found with equation 16. For 10^4 cycles the stress is 1840 MPa calculated with equation 16

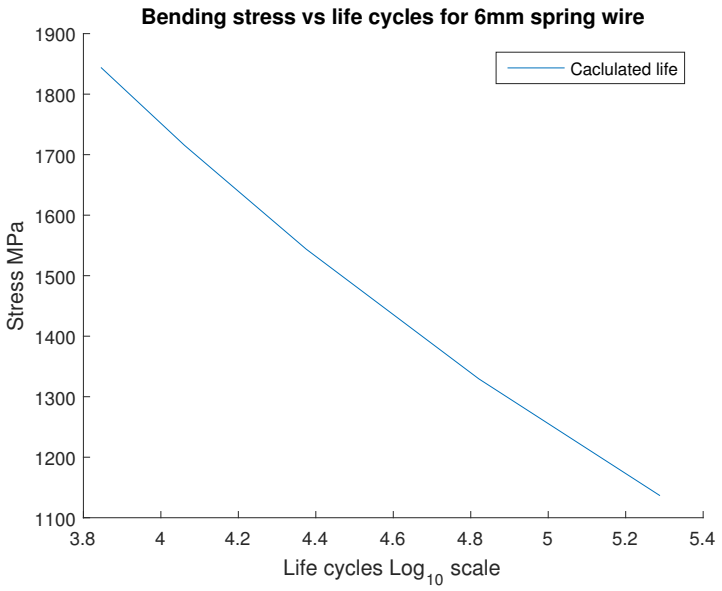

Figure 10: Bending stress vs theoretical fatigue life logarithmic transform

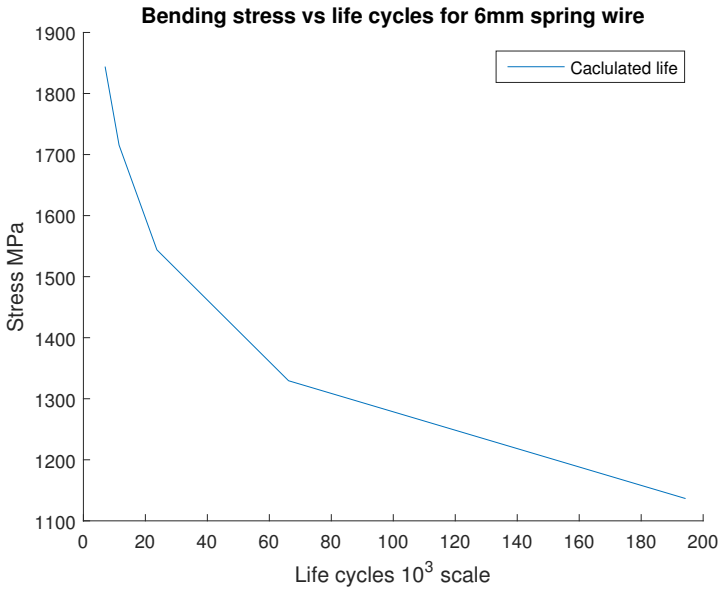

Figure 11: Bending stress vs theoretical fatigue life

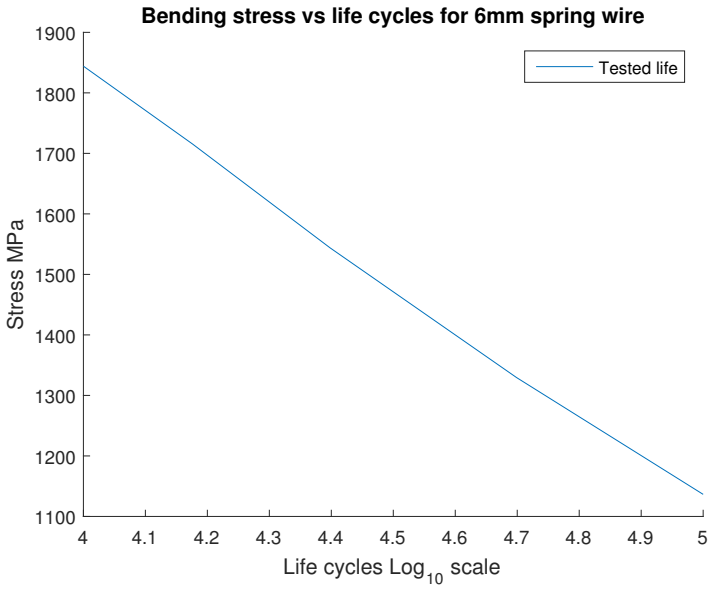

Figure 12: Bending stress vs tested fatigue life logarithmic transform

with the K_i stress factor for 6mm spring wire diameter and 50.8mm inner diameter. This value is above the minimum tensile strength of the material. The reason for the spring not breaking is that stress is not uniform in its cross section. Even if the material yields on the inner surface it does not break the spring but affects its endurance. Thus it only has 10^4 cycles until breaking. The linear relation between stress and life cycles after logarithmic transform is used to determine the life of the spring using slope of the line. The stress for 10^4 cycles is used as a base point. This stress is calculated for different spring wire diameters with equation 16 from data in tested fatigue life from FlexiForce. Equation 26 is used to calculate how much fraction is added to the power in equation 27 where the fatigue life cycles are found.

$$\text{add life} = (\sigma_{10^4} - \sigma) \times slope \tag{26}$$

$$cycles = 10^{4+\text{add life}} \tag{27}$$

2.2.5 Comparison of fatigue life

Calculated fatigue life is affected by two variables, fatigue strength fraction (f) and endurance limit ($S_e^{'}$). f is bound to tensile strength and for $1700 - 1400MPa$ it is around

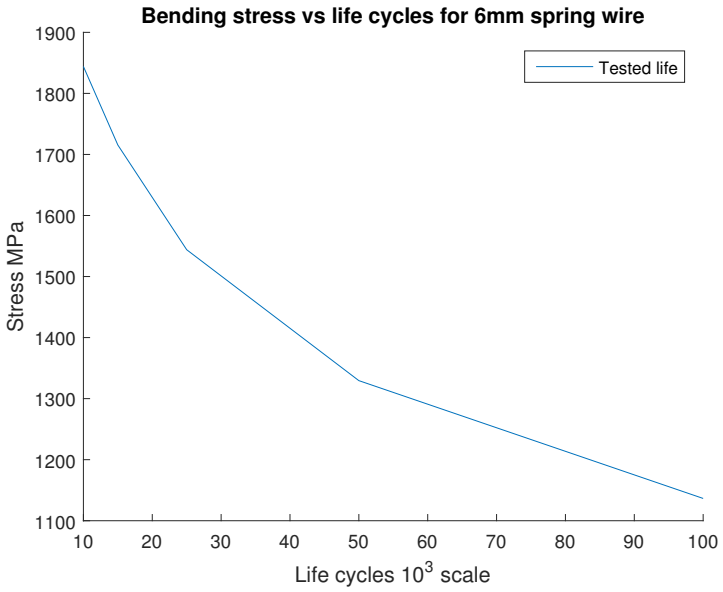

Figure 13: Bending stress vs tested fatigue life

0.77 [6]. $(S_e^{'})$ ranges between 40 to 60 percent of S_{ut}. The percentage is affected by how high S_{ut} is. $(S_e^{'})$ is set at $700MPa$ for S_{ut} greater then $1400MPa$. The results from these calculation can be found in figures 14 and 15 with the x axis scaled with 10^3 and log_{10} scale respectively for life cycles. On the y axis is bending stress calculated with equation 16. In these examples a spring with 5mm wire diameter, 50.8mm inner diameter and 109 active coils was used.

If f and $(S_e^{'})$ are modified to $f = 0.9$ and $(S_e^{'}) = 0.35S_{ut}$ the difference is very small see figure 16 and 17 for comparison. For the example in figures 16 and 17 a 5mm diameter spring wire is used with $S_{ut} = 1660MPa$ taken from table 8. That gives $(S_e^{'}) = 0.35S_{ut} = 494MPa$ instead of $700Mpa$ given in figure 14 and 15. These values of f and $(S_e^{'})$ are however found by iterating until the smallest error exists between test fatigue life and calculated. Thus they cannot be said theoretical. For true difference between test data and theoretical the values of f and S_{ut} should be 0.77 and $700MPa$ respectively. The difference in percentages can be found in table 6. This error was also calculated with $f = 0.9$ and $(S_e^{'}) = 0.35S_{ut}$ see table 7. In these tables spring wire diameter 5mm to 10mm with increment of 1mm are compared to test fatigue life ranging from 10.000 to 100.000 cycles. The theoretical life can be under the tested life and above. The error grows with increased diameter and more cycles. Similar results are from the calculated fatigue life with $f = 0.9$ and $S_e^{'} = 0.35S_{ut}$ but with much smaller errors.

From table 6 it can be seen that the error grows as the diameter of the spring grows for

10^5 cycles. But if we look at the range between 10^4 to $25 * 10^3$ and spring diameter sizes 5-7mm the error is less then 30%. This range is most common in sectional overhead doors as it covers most doors under 4000mm width and 3500mm height. However when going to spring sizes and life cycles larger then that the error can be more then 200%. When looking at table 7 where $f = 0.9$ and $S_e^{'} = 0.35S_{ut}$ the largest error is 15.9% which is much more accurate or in the order of 28 times better for the largest error. It is worth noting that when values for f and ratio for $(S_e^{'})$ were found with the iterative method only values for 5mm spring wire diameter was used.

Table 6: Fatigue life difference for test and theoretical calculations with $f = 0.77$ and $S_e^{'} = 700MPa$. Test cycles are the reference point

cycles/spring diam	10.000	15.000	25.000	50.000	100.000
5	27,2%	23,6%	6,9%	-22,4%	-55,4%
6	30,0%	23,3%	5,1%	-32,4%	-94,3%
7	23,9%	11,7%	-15,7%	-71,6%	-140,7%
8	3,4%	-8,4%	-46,7%	-122,5%	-264,2%
9	8,2%	-15,6%	-53,9%	-137,6%	-285,3%
10	-20,3%	-43,3%	-94,1%	-232,2%	-446,7%

Table 7: Fatigue life difference for test and modified theoretical calculations with $f = 0.9$ and $S_e^{'} = 0.35S_{ut}$. Test cycles are the reference point

cycles/spring diam	10.000	15.000	25.000	50.000	100.000
5mm	3,3%	9,2%	6,9%	3,3%	2,2%
6mm	11,4%	15,4%	14,4%	10,1%	2,0%
7mm	10,1%	11,7%	8,6%	2,4%	-0,3%
8mm	-0,6%	4,8%	1,3%	-4,1%	-15,5%
9mm	6,9%	6,2%	5,4%	1,5%	-5,9%
10mm	-5,4%	-1,2%	-1,2%	-9,7%	-15,9%

2.3 Panel calculations

Panels comes in variety of sizes ranging from 488 to 732mm in height. The length of panels come in much more variety ranging from 2200 to 13500mm. In these calculations two heights of panels were used 488mm and 610mm. Number of panel sections is impacted by the height of the daylight opening. In both end of each panel is an end cap that is made for that panel height. These end caps are to reinforce the binding of hinges, bottom brackets and top brackets to the panel. They also guide the location of screws as they are predrilled.

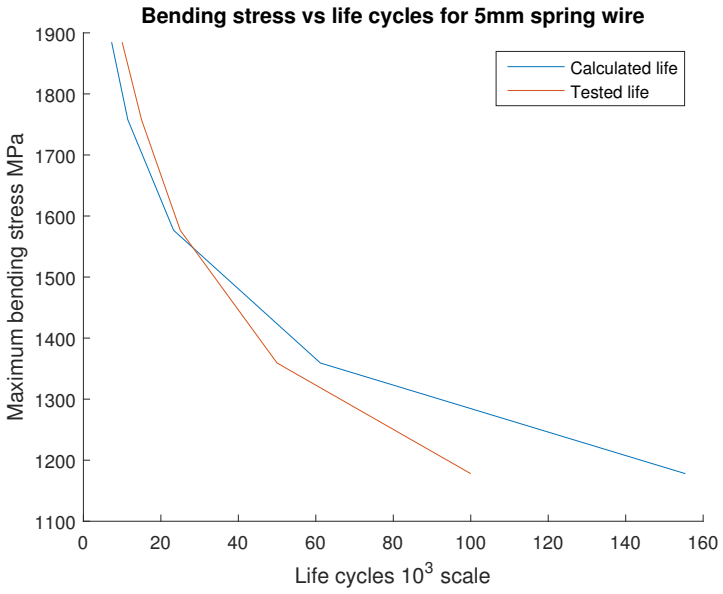

Figure 14: Fatigue comparison 10^3 scale $f = 0.77$, $S_e^{'} = 700MPa$

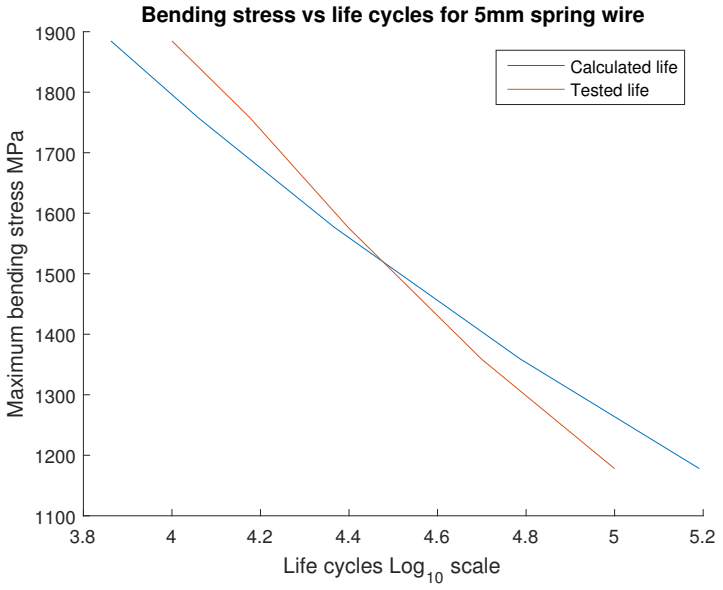

Figure 15: Fatigue comparison log_{10} scale $f = 0.77$, $S_e^{'} = 700MPa$

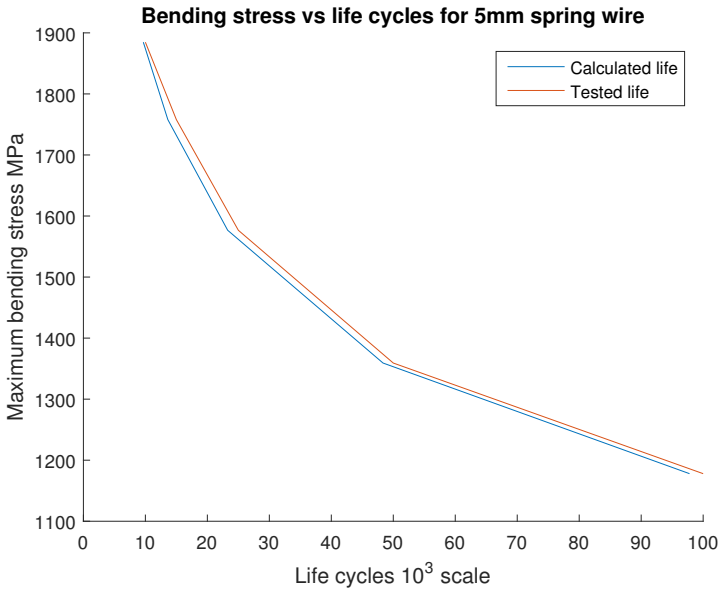

Figure 16: Fatigue comparison 10^3 scale $f = 0.9$, $S_e^{'} = 0.35 S_{ut}$

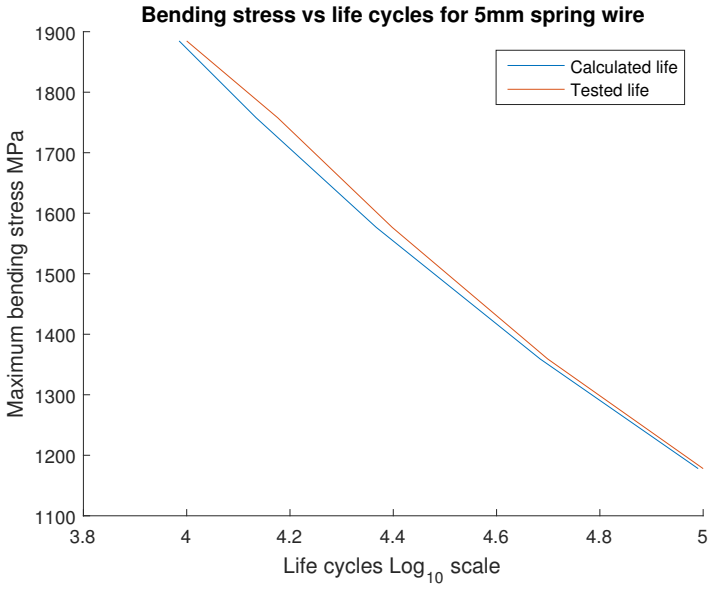

Figure 17: Fatigue comparison log_{10} scale $f = 0.9$, $S_e^{'} = 0.35 S_{ut}$

Table 8: Ultimate tensile strength of EN-10270-1 SH spring material [1]

Diameter (mm)	Lower (MPa)	Upper (MPa)	Mean (MPa)
4,5	1690	1880	1785
5	1660	1840	1750
5,5	1626,7	1806,7	1716,7
6	1590	1770	1680
6,5	1560	1740	1650
7	1540	1710	1625
7,5	1510	1680	1595
8	1490	1660	1575
8,5	1470	1630	1550
9	1450	1610	1530
9,5	1430	1590	1510
10	1410	1570	1490

These come on two varieties single width and double width. Double width end caps are used when width of door exceeds five meters in width. The algorithm used to find the number of sections and size of them:

1. $$wh = dh - 40 \tag{28}$$

   Where wh is the working height of the door and dh is the daylight opening height. The subtracted number 40mm are due to top/bottom lists and rubber seals.

2. $$ns = ceil\left(\frac{wh}{610}\right) \tag{29}$$

   Where ns is number of section rounded up to nearest integer. Highest section height is used as denominator here it is 610mm.

3. $$diff = 488 - 610 = 122 \tag{30}$$

   $diff$ is the difference in height of the 488 and 610 panels.

4. $$ch = ns \times 610 \tag{31}$$

   ch is calculated height

5. A while loop is used to cut ch down. First check if ch minus $diff$ is larger then wh if true minus $diff$ from ch and count. This is run until false is produced. Now the number of 610 and 488 sections are known. Count holds the number of 488 sections and count minus number of sections is the number of 610 sections.

6. Through this loop the difference between calculated height and working height can be as big as 121 mm. This means that the door can be 121 mm higher then the daylight opening height. This does not produce any problem unless the head room is small and if so the top section needs to be cut. This is written in the manufacturing sheet. It is worth noting that the difference between working height and calculated can be higher for doors under 1464 mm in height. This height is extremely rear and is addressed in the calculator by cutting off the top section.

Width of the panel is bound by width of the daylight opening and track type. Width for all track type except LHR and FLHR is the daylight opening width plus 45mm. For the low head room types it is plus 35mm. This difference is due to additional hardware used with low head room tracks.

2.4 Hardware calculation

Most hardware items have their quantity bound to the height and width of the door. Other are bound to track type. Calculation for each item was put in a separate function.

2.4.1 Wind reinforcement

When width of doors exceeds 3500mm a reinforcing strut is added to each sectional junction. The type of strut is bound by width of the door. Also type of panel used impacts the type of strut. In this software Epco H-series panel is used. They are in compliance with resistance to wind EN 12424: class 4 [8]. For panels of width 4000mm. The limiting factor for width of doors are the struts. If door width exceeds 6520mm a special strut is needed that is custom made for the door. Thus this is the limiting factor for width the software can calculate.

2.4.2 Vertical and horizontal tracks

There are seven types of track opening. All of them use two types of tracks, vertical and horizontal with the exception of VL that only uses vertical tracks. Tracks from FlexiForce come in variety of sizes. Both for residential and industrial doors. The main difference between the two is that residential tracks come preassembled with maximum height of 3000mm and double horizontal tracks. It has nothing to do with whether the door is for industrial use or residential. Vertical tracks in the residential line come in sizes 2250, 2370, 2500, 2750 and 3000mm. The software finds correct size of vertical track and if it is required to saw off the track. Headroom becomes smaller by same amount that the vertical tracks

exceed the daylight height. Thus the vertical tracks are cut from the bottom to correct size if the headroom becomes smaller then the minimum headroom. Horizontal tracks come in the same variety of sizes as the vertical. Preassembled tracks are cheaper so it is preferred to use them. But that is not possible for all track opening types that are below 3000mm in height. The following list explains what track types can use these preassembled tracks and in what height.

1. For track opening type: NL, LHR, FNL and FLHR preassembled vertical tracks can be used with height under 3000mm.

2. For track opening type: HL and FHL preassembled vertical tracks can be used if the height plus high lift is under 3000mm.

3. VL tracks always require industrial tracks.

For horizontal tracks:

1. If the track opening type is VL then there is no horizontal track.

2. For track opening type HL and FHL the high lift length is subtracted from the height. So a 3500mm high door with hi lift 500mm will still be able to use a 3000mm horizontal track.

3. For NL, LHR, FNL and FLHR opening tracks, preassembled horizontal tracks are used for height under 3000mm.

4. All other sizes uses industrial tracks.

2.4.3 Bearing plates and shaft

Bearing plates hold the shaft with springs and lifting cable drums called spring system in its place see figure 18 for close up drawing of spring system colored in red for NL track. Amount of bearing plates is controlled by the width of door. There is a minimum of two plates. For width of 3050mm and above three plates are used. For more then 4050mm four plates and above 5050mm five plates. There are many different bearing plates available. Here six types are used with varying heart distances see figure 19 for a drawing of bearing plate with heart distance 67mm. The type of bearing plate used is controlled by lifting cable drum diameter and spring diameter. The software selects smallest bearing plate based on those values. Bearing plates are what controls minimum head room each door needs. The shaft is what connects the torque from springs to lifting cable drums. Its length is equal to daylight width plus 250mm. There are two types of shafts one that has key way and one that does not see figure 20 for picture of a shaft with key way. If door weight is over 240 kg a shaft with key way is required. The spring fittings and lifting cable drum all come with key way and 8mm bolts used to lock them in on both types of shaft.

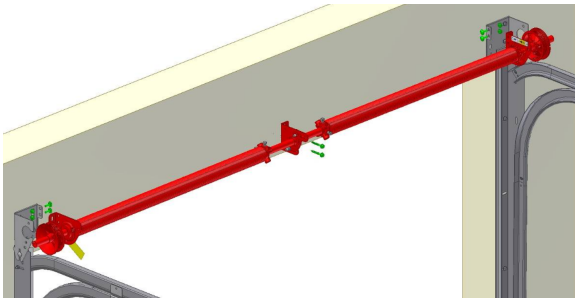

Figure 18: Shaft with springs, lifting cable drums and bearing plates [3]

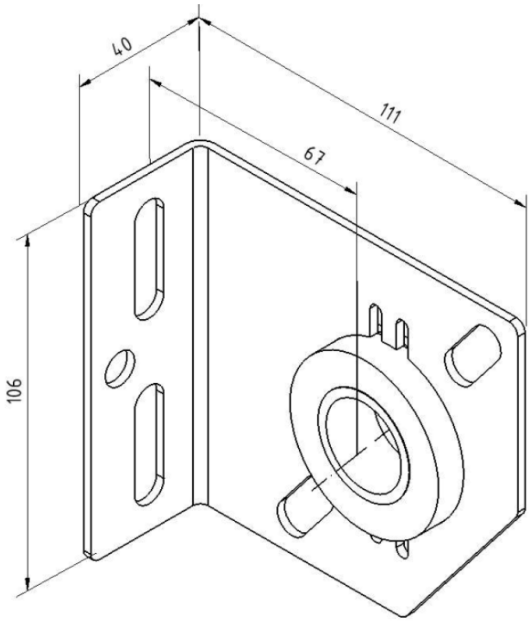

Figure 19: Bearing plate for smaller wire drums than 67mm radius [3]

Figure 20: Shaft with key way
[3]

2.4.4 Top and bottom roller brackets

Top roller brackets come in two types adjustable and non adjustable. If the door and vertical track are of equal height and under 3000mm then the non adjustable bracket is used. Also if the door is LHR or FLHR then the non adjustable bracket is used. Otherwise its adjustable bracket. It is also possible to have this item made from stainless steel.

Bottom brackets are what the lifting cable connects to. These brackets come in various types. In the software two types are used as they address the majority of doors. First aluminum adjustable bracket see figure 21 they are used if stainless steel is selected or for doors over 3000mm in width or weigh more then 300 kg which is the maximum weight the other bottom bracket can hold. For all other doors a small bottom bracket is used that is not adjustable.

2.4.5 Hinges

Hinges with the top and bottom roller brackets are what hold the rollers. These rollers are then what connects the door to the tracks. Hinges are located on the intersections of sections at each end of the door. They are screwed on both sections and thus holds them together. Number of hinges are bound to number of sections see equation 32 for calculations on number of hinges. Where $sections_{no}$ is number of sections. See figure 22

Figure 21: 428TAI aluminum adjustable bottom bracket
[3]

for hinge screwed on end cap with roller bracket and roller.

$$hinges_{no} = 2(sections_{no} - 1) \tag{32}$$

Intermediate hinges are put at the intersections of sections like hinges. They are located between hinges with maximum 100 cm between them see equation 33 for calculations on amount of intermediate hinges. Ceil rounds up the results to the nearest integer. Intermediate hinges come in two varieties stainless steel and steel.

$$intermediate_{no} = (sections_{no} - 1) * (ceil\left(\frac{width}{1000}\right) - 1) \tag{33}$$

On the hinges are roller brackets that rollers are inserted in. Similar roller brackets are located on bottom and top brackets. Thus the number of rollers needed are the sum of hinges plus four. These rollers come in three varieties stainless steel, steel and double steel shaft length. Double length roller is needed for doors exceeding five meters in width. Also double the amount of hinges are required. This means that each roller now passes through two hinges instead of one. This is done to reinforce the binding of hinges to the door.

Figure 22: Hinge with roller bracket and roller
[3]

2.4.6 Horizontal track stopper

Located on the end of the horizontal track is a stopper. It is put there to ensure that the door will not leave the track when being opened fast and swinging high up to the horizontal track that the top roller might leave the track. Instead it hits this stopper. However when using a track electric operator this piece is not required. When using a shaft electric operator a spring bumper is required to push the door down.

2.4.7 Bolts and screws

Hinges, intermediate hinges, top and bottom brackets are all fastened to the door panel with 6,3x35 mm screws either stainless steel or steel see equation 34 for calculation on amount of screws. Plus twenty is for the bottom and top brackets. All types use the same amount of screws. Six screws for each bottom bracket and four screws for each top bracket. Also the wind reinforcing struts are screwed to the sections with maximum 100cm between screws.

$$screws_{no} = 6 * hinges_{no} + 4 * intermediate_{no} + 20 \tag{34}$$

Bolts also come in either stainless steel or steel. There are two types of bolts 8x25 mm and 6x20 mm. 6 mm bolts are only used in fastening roller brackets to the hinges. Two bolts are used for each hinge. 8 mm bolts are only used in fastening spring fittings to bearing plate. Two bolts are used for each spring.

There are special track bolts that are 6x16 mm and have a flat head. They are used to fasten the vertical and horizontal tracks to each other. Four bolts are required for each door. These bolts are also used when fastening the horizontal track to the ceiling through horizontal track supports. Depending on weight of the door. If the door is lighter than 100kg two extra bolts are required else they are four.

3 Results

The results of the spring calculation is in coherence with other commercial spring calculators mainly SpringForce calculator from FlexiForce. As the same material is used and test data for fatigue life. The software saves the most time when standard doors are calculated. All default values in the GUI are set to standard doors. And the minimum input is height, width and head room. Also the price calculation is much more accurate then tabulated price list as most doors are not sold in some standard sizes. As the software creates manufacturing sheet at the same time as price offer sheet they are always in harmony. Thus the customer can be seen as reviewing the accessories and specific information on the manufacturing sheet when accepting price offer sheet. This is fundamental in minimizing input error when creating the manufacturing sheet.

Theoretical spring fatigue calculations were quite close to tested fatigue for a given range. However when leaving that range the error grew to more than 200%. With modification of f and $S_e^{'}$ resulted in much better results. But these modifications were found by iterating them until the error got very small between calculated life and tested. Thus it cannot be said that is is theoretical. But these factors might prove valuable if used with other spring material that share similar characteristics with EN-10270-1 SH spring material.

There are some limiting factors on size of doors. Width cannot exceed 6520mm due to wind reinforcement struts not available in mass production. They can be ordered larger but that is a special manufacture for that specific door. Weight of the door cannot exceed 700kg due to lifting cable drums and height cannot exceed ten sections due to danger of the second lowest section splitting in two from weight of section that are stacked on top of it. There are some special equipment available to get higher doors but that requires special manufacturing specific to that door.

4 Discussion

There were no information on how the fatigue life was tested by FlexiForce other then the fact they created their own testing rig. So it is unknown for what spring diameters and what working stress they were tested. These results might be from a few samples giving room to error. However the testing data is still probably more reliable then the theoretical calculations. It is widely accepted that theoretical fatigue calculations can be far away from true life and thus tested data is preferred.

Weight of each door is based on its square meters instead of summing the weight of all hardware screwed or bolted on the panel. This is done due to inaccurate weight of hardware components. The need to weigh each component accurately would be needed to use the summed weight of all components. This would however result in more accurate weight of the door. But it is time consuming to do and out of scope for this project.

No corporate testing has been conducted on the software. The testing was mainly conducted by the author. So for further development the software needs to be put in use in beta testing to fix possible bugs.

There are two useful added features that would be good to add. Quantity of all components of doors that have been ordered but not yet been manufactured. This would give a clear picture of how big the stock of components need to be to manufacture them. Also a good added feature would be a technical drawing of the door with all relevant dimensions. This would give the customer much better idea of what he is buying.

References

[1] *Steel wire for mechanical springs.* European Standard EN 10270-1, 2001.

[2] Raynor garage doors - corporate - history. [Online]. Available: http://www.raynor.com/corporate/history.cfm

[3] FlexiForce, Hanzeweg 19, 3771NG Barneveld, The Netherlands. [Online]. Available: http://www.flexiforce.com

[4] Sectional overhead doors. [Online]. Available: http://doorfix.ie/doorfix/Files/Sectional-Overhead-2pp.pdf

[5] H. E. Boyer, *Atlas of Fatigue Curves.* ASM International, 1986.

[6] R. G. Budynas and K. J. Nisbett, *Shigley's Mechanical Engineering Design.* McGraw Hill Higher Education, 2011.

[7] A. M. Whal, *Mechanical Springs.* Penton Puplishing Company Cleveland, Ohio, 1944.

[8] Technical handbook. Epco. [Online]. Available: http://epco.be/img/pdf/manuel_technique_en.pdf?1465212852

Appendix A

Spring calculation code

```
#!/usr/bin/python3
#coding: utf 8
import math
from Database1 import Data


def Drum(drum):
    # Use the correct drum:
    if drum == "FFNL12":
        drum = Data().drum_FFNL12
    elif drum == "FFNL10":
        drum = Data().drum_FFNL10
    elif drum == "FFNL18":
        drum = Data().drum_FFNL18
    elif drum == "FFNL32":
        drum = Data().drum_FFNL32
    # Hi Lift
    elif drum == "FFHL54":
        drum = Data().drum_FFHL54
    elif drum == "FFHL120":
        drum = Data().drum_FFHL120
    elif drum == "FFHL164":
        drum = Data().drum_FFHL164
    # Vertical Lift
    elif drum == "FFVL11":
        drum = Data().drum_FFVL11
    elif drum == "FFVL18":
        drum = Data().drum_FFVL18
    elif drum == "FFVL28":
        drum = Data().drum_FFVL28
    else:
        print("ERROR␣Wrong␣wire␣drum␣selected")
        raise ValueError("Wrong␣wire␣drum␣selected␣for␣spring␣
            calculations")
    return(drum)

def spring(height, width, weight, life, drum, track, number_60_sections
    , no_springs=2, slope=0, hi_lift=0):
    ### Calculates the size of springs ###

    slope = math.radians(slope) # convert from degrees to radians
    # Constants:
```

```
E = 206000
G = 81500
g = 9.81 # earth gravity
open_door_fraction = 0.68 # for 300mm radius vertical tracks
slope_fatigue = 0.001521862209147
track_friction = 0.06
bottom_section_extra_weight = 2.1 # Weight for bottom list and
    rubber. per meter

# database variables:
section61 = Data().section61_kg_p_m
section488 = Data().section488_kg_p_m
wire_strength = Data().wire_strength
spring_diam_avail = Data().spring_diam_avail
length_fittings = Data().length_fittings # Add to spring length to
    get total length
spring_ut = Data().spring_ut # the stress at 10.000 openings.
    without K_i. Used as base line in fatigue calculations.

# Find the correct wire diameter
if weight <= wire_strength[3]:
    d_wire = 3
elif weight <= wire_strength[4]:
    d_wire = 4
elif weight <= wire_strength[5]:
    d_wire = 5
elif weight <= wire_strength[6]:
    d_wire = 6

# If there is at least one 60 section then that is used as a bottom
     section (rule of thumb is use the biggest sections on the
    bottom)
if number_60_sections >= 1:
    bottom_section_weight = width/1000 * (section61 +
        bottom_section_extra_weight)
else:
    bottom_section_weight = width/1000 * (section488 +
        bottom_section_extra_weight)

# Find the open door weight. Depends on track type and bottom
    section
if track == "Normal␣Lift␣(NL)" or track == "Low␣Head␣Room␣(LHR)":
    open_door_weight = bottom_section_weight * open_door_fraction
elif track == "Follow␣the␣Roof␣NL␣(FNL)" or track == "Follow␣the␣
    roof␣LHR␣(FLHR)":
```

```
    open_door_weight = bottom_section_weight * open_door_fraction +
        (weight    bottom_section_weight) * math.sin(slope)
elif track == "High Lift (HL)":
    open_door_weight = (hi_lift / height) * weight
elif track == "Follow the Roof HL (FHL)":
    open_door_weight = (hi_lift / height) * weight + (weight
       bottom_section_weight    ((hi_lift / height) * weight)) *
       math.sin(slope)
elif track == "Vertical Lift (VL)":
    open_door_weight = weight
else:
    print("ERROR track type incorrect")
    raise ValueError("Incorrect track type in spring calculation")


drum = Drum(drum)

# Calculations:
weight_frict = weight * (1+track_friction)
weight_N = weight_frict * g

# NL, LHR, FNL, FLHR:
if track == "Normal Lift (NL)" or track == "Low Head Room (LHR)" or
     track == "Follow the Roof NL (FNL)" or track == "Follow the
    roof LHR (FLHR)":
    drum_U = drum["flat torque arm"] * 2 * math.pi
    turns = (height    drum["extra height"]) / drum_U
    M_max = weight_N * (drum["flat torque arm"] + (d_wire/2)) /
        no_springs
    M_min = open_door_weight * g * (drum["flat torque arm"] + (
        d_wire/2)) / no_springs
    M_perturn = (M_max M_min) / turns
    turns_total = turns + (open_door_weight * drum["flat torque arm
        "] * g) / (M_perturn * no_springs)
    wire_length = drum["wire length from floor to shaft center plus
        "] + height


# VL
elif track == "Vertical Lift (VL)":
    count = 0
    cycles = 0
    torque_arm = drum["flat torque arm"]
    rate = drum["rise per wrap"]
    while count <= height:
        count = count + ( torque_arm + (rate/2) )*2*math.pi
```

```
            if count >= height:
                count = count    ( torque_arm + (rate/2) )*2*math.pi
                break
            torque_arm = torque_arm + rate
            cycles = cycles + 1
            frac = (height count)/(math.pi*(rate+2*torque_arm))
        turns = cycles + frac
        M_max = weight_N * (torque_arm + (d_wire/2)) / no_springs
        M_min = open_door_weight * g * (drum["flat torque arm"] + (
            d_wire/2)) / no_springs
        M_perturn = (M_max M_min)/turns
        pretension_turns = M_min / M_perturn
        turns_total = turns + pretension_turns
        wire_length = drum["Size from floor to shaft center, minus
            opening height plus"]    height

    # HL, FHL
    elif track == "High Lift (HL)" or track == "Follow the Roof HL (FHL
        )":
        # Find the torque arm and cycles for the correct hi lift size
        count = 0
        cycles = 0
        torque_arm = drum["flat torque arm"]
        rate = drum["rise per wrap"]
        frac = 0
        while count <= hi_lift:
            count = count + ( torque_arm + (rate/2) )*2*math.pi
            if count >= hi_lift:
                count = count    ( torque_arm + (rate/2) )*2*math.pi
                break
            torque_arm = torque_arm + rate
            cycles = cycles + 1
            frac = (hi_lift count)/(math.pi*(rate+2*torque_arm))
        turns_hi_lift = cycles + frac
        #M_max = weight_N * (drum["flat torque arm"] + (d_wire/2)) /
            no_springs
        M_max = weight_N * (torque_arm + (d_wire/2)) / no_springs
        # Find the cycles for the flat torque arm
        drum_U = drum["flat torque arm"] * 2 * math.pi
        M_min = open_door_weight * g * (drum["flat torque arm"] + (
            d_wire/2)) / no_springs
        turns = (height    hi_lift) / drum_U
        M_perturn = (M_max M_min) / (turns + turns_hi_lift)
        turns_total = turns + turns_hi_lift + (open_door_weight * drum[
            "flat torque arm"] * g) / (M_perturn * no_springs)
```

```
        wire_length = drum["wire length from floor to shaft center
            minus HL size plus"]    hi_lift

    else:
        print("ERROR track selection not recognized")
        raise ValueError("Track selection not recognized")

    # Spring calculations:
    S_start = 1756
    d = math.pow((32*M_max)/(math.pi*S_start),(1/3))
    d = math.ceil(d*2) / 2
    if d < 4.5: # 4.5 is the smallest possible wire diameter
        d = 4.5
    S_ut = spring_ut[d]

    if (d in spring_diam_avail.keys()):
        inner_d = spring_diam_avail[d]
    else:
        print("ERROR diameter of spring out of range use more/less
            springs") # throw exeption / error
        raise ValueError("Error The calculated wire diameter of the
            spring is out of range. Use more/less springs")

    S = ((32*M_max) / (math.pi*(math.pow(d,3))))


    # If stress exceeds ultimate. Get bigger diameter
    while S > S_ut:
        d = d + 0.5
        S = ((32*M_max) / (math.pi*(math.pow(d,3))))
        S_ut = spring_ut[d]

    # Spring life: use tabulated fatigue data from FlexiForce
    add_life = (S_ut    S) * slope_fatigue
    cycles = math.ceil(math.pow(10, 4+add_life))
    # get higher diameter wire if cycles are less then predefined life
    while cycles < life:
        d = d + 0.5
        S = ((32*M_max) / (math.pi*(math.pow(d,3))))
        S_ut = spring_ut[d]
        add_life = (S_ut    S) * slope_fatigue
        cycles = math.ceil(math.pow(10, 4+add_life))

    D = inner_d + d # Mean diameter
    N_a = (E*d**4)/(10.2*D*M_perturn) # Number of active coils
    length_spring = round((N_a + 4) * d + length_fittings[inner_d])
```

```
    if length_spring * no_springs > width:
        print("combined spring length exceeds door width")


    return(d,inner_d,length_spring,round(turns_total,1),cycles,d_wire,
        wire_length)

    # Line 240 is the end line in latex
    # Use For Debugging:
    #
        ==========================================================

    # print("Bottom section weight {}".format(bottom_section_weight))
    # print("open door weight {}".format(open_door_weight))
    # print("turn before pretension {}".format(turns))
    # print("Torque per turn {}".format(M_perturn))
    # print("Max torque {}".format(M_max))
    # print("Min torque {}".format(M_min))
    # print("spring ultimate strength {}".format(S_ut))
    # print("Diameter of wire {}".format(d))
    # print("available wire sizes {}".format(spring_diam_avail))
    # print("Number of active coils {}".format(N_a))
    # print("Unmodified length of spring {}".format((N_a+6)*d))
    # print("mean diameter {}".format(D))
    # print("length of spring in mm {}".format(length_spring))
    # print("number of cycles {}".format(cycles))
    # return(d,length_spring,round(turns_total,2),cycles,d_wire)
    #
        ==========================================================


def main():
    # Use main for debugging
    drum = "FFVL11"
    height = 2900
    width = 3000
    weight = height/1000 * width/1000 * 12
    life = 20000
    no_springs = 2
    track = "Vertical Lift (VL)"
    count_60 = 1
    hi_lift = 0
    slope = 0

    spring_type = spring(height, width, weight, life, drum, track,
        count_60, no_springs, slope, hi_lift)
```

```
    print(spring_type)

if __name__ == '__main__': main()
```

Appendix B

Hardware database code

```
'''
Created on 13. mai 2016

@author: Magnus
'''

class Data:
    '''
    Stores all data for parts except price.
    '''
    def __init__(self):
        # Normal Lift Drums:
        self.drum_FFNL12 = {"extra height": 25.31, "max opening": 3680,
            "max door weight": 500, "max cable diameter": 5, "cable
            capacity of safety wraps": 740, "high moment arm" : 58.2,
                    "no of spiral wraps": 1.25, "cable capacity of
                        spiral wraps": 440, "rise per wrap": 4.3, "no of
                        flat wraps": 9.75, "cable capacity of flat wraps"
                        : 3240,
                    "flat torque arm": 52.8, "wire length from floor to
                        shaft center plus": 820,"Max. outside diameter":
                        124.0}

        self.drum_FFNL10 = {"extra height": 0, "max opening": 3000, "
            max door weight": 320, "max cable diameter": 3, "cable
            capacity of safety wraps": 124, "high moment arm" : 39.5,
                    "no of spiral wraps": 0, "cable capacity of spiral
                        wraps": 0, "rise per wrap": 0, "no of flat wraps"
                        : 12.6, "cable capacity of flat wraps": 3000,
                    "flat torque arm": 39.5, "wire length from floor to
                        shaft center plus": 525,"Max. outside diameter"
                        :84.0}

        self.drum_FFNL18 = {"extra height": 15.2, "max opening": 5570,
            "max door weight": 500, "max cable diameter": 5, "cable
            capacity of safety wraps": 940, "high moment arm" : 74.7,
```

```
            "no_of_spiral_wraps": 1, "cable_capacity_of_spiral_
                wraps": 450, "rise_per_wrap": 5.5, "no_of_flat_
                wraps": 11.75, "cable_capacity_of_flat_wraps":
                5120,
            "flat_torque_arm": 69.2, "wire_length_from_floor_to_
                shaft_center_plus": 1040,"Max._outside_diameter"
                :158.0}

    self.drum_FFNL32 = {"extra_height": 15.92, "max_opening":
        10175, "max_door_weight": 700, "max_cable_diameter": 6, "
        cable_capacity_of_safety_wraps": 1370, "high_moment_arm" :
        108.9,
            "no_of_spiral_wraps": 1, "cable_capacity_of_spiral_
                wraps": 670, "rise_per_wrap": 4.8, "no_of_flat_
                wraps": 14.5, "cable_capacity_of_flat_wraps":
                9505,
            "flat_torque_arm": 104.1, "wire_length_from_floor_to
                _shaft_center_plus": 1510,"Max._outside_diameter"
                :226.0}

    # High Lift Drums:
    self.drum_FFHL54 = {"extra_height": 0, "max_hi_lift": 1370, "
        max_opening": 4800, "max_door_weight": 500, "max_cable_
        diameter": 5, "cable_capacity_of_safety_wraps": 1150, "high_
        moment_arm" : 91.35,
            "no_of_spiral_wraps": 2.75, "cable_capacity_of_
                spiral_wraps": 1380, "rise_per_wrap": 8, "no_of_
                flat_wraps": 10.5, "cable_capacity_of_flat_wraps"
                : 4570,
            "flat_torque_arm": 69.7, "wire_length_from_floor_to_
                shaft_center_minus_HL_size_plus": 2650,"Max._
                outside_diameter":188.0}

    self.drum_FFHL120 = {"extra_height": 0, "max_hi_lift": 3050, "
        max_opening": 5050, "max_door_weight": 500, "max_cable_
        diameter": 5, "cable_capacity_of_safety_wraps": 1460, "high_
        moment_arm" : 115.8,
            "no_of_spiral_wraps": 5, "cable_capacity_of_spiral_
                wraps": 3060, "rise_per_wrap": 8, "no_of_flat_
                wraps": 7.75, "cable_capacity_of_flat_wraps":
                3680,
            "flat_torque_arm": 75.5, "wire_length_from_floor_to_
                shaft_center_minus_HL_size_plus": 4660,"Max._
                outside_diameter":238.0}
```

```
self.drum_FFHL164 = {"extra height": 0, "max hi lift": 4100, "
    max opening": 6000, "max door weight": 650, "max cable
    diameter": 6, "cable capacity of safety wraps": 1725, "high
    moment arm" : 136,
            "no of spiral wraps": 6, "cable capacity of spiral
                wraps": 4100, "rise per wrap": 8.76, "no of flat
                wraps": 11, "cable capacity of flat wraps": 5775,
            "flat torque arm": 83.5, "wire length from floor to
                shaft center minus HL size plus": 6015,"Max.
                outside diameter":280.0}

# Vertical Lift Drums:
self.drum_FFVL11 = {"extra height": 0, "max opening": 3300, "
    max door weight": 500, "max cable diameter": 5, "cable
    capacity of safety wraps": 1325, "high moment arm" : 105.65,
            "no of spiral wraps": 7.5, "cable capacity of spiral
                wraps": 3300, "rise per wrap": 9.525, "flat
                torque arm": 34.2,
            "Size from floor to shaft center , minus opening
                height plus": 4785,"Max. outside diameter":218.0}

self.drum_FFVL18 = {"extra height": 0, "max opening": 6000, "
    max door weight": 600, "max cable diameter": 5, "cable
    capacity of safety wraps": 1745, "high moment arm" : 139,
            "no of spiral wraps": 11, "cable capacity of spiral
                wraps": 6000, "rise per wrap": 9.525, "flat
                torque arm": 34.2,
            "Size from floor to shaft center , minus opening
                height plus": 7955,"Max. outside diameter":284.0}

self.drum_FFVL28 = {"extra height": 0, "max opening": 8500, "
    max door weight": 750, "max cable diameter": 6, "cable
    capacity of safety wraps": 510, "high moment arm" : 164.6,
            "no of spiral wraps": 14, "cable capacity of spiral
                wraps": 9010, "rise per wrap": 9.5, "flat torque
                arm": 34.9,
            "Size from floor to shaft center , minus opening
                height plus": 9280,"Max. outside diameter":348.0}

self.wire_strength = {3: 163, 4: 321, 5: 490, 6: 723}

# Springs:
self.spring_diam_avail = {4.5: 50.8, 5: 50.8, 5.5: 50.8, 6:
    50.8, 6.5: 66.7, 7: 66.7, 7.5: 95.3, 8: 95.3, 8.5: 95.3, 9:
    95.3, 9.5: 152.4, 10: 152.4}
```

```
self.spring_item_no = {4.5: "VL45", 5: "VL50", 5.5: "VL55", 6:
    "VL60", 6.5: "VL65", 7: "VL70", 7.5: "VL75", 8: "VL80", 8.5:
     "VL85", 9: "VL90 3", 9.5: "VL95 6", 10: "VL100 6"}
self.spring_fittings_item_no = {50.8: "FF 2.00TAI", 66.7: "FF
    2.63TAI", 95.3: "FF3.75LE", 152.4: "FF600"}
self.length_fittings = {50.8: 98, 66.7: 110, 95.3: 110.5,
    152.4: 103.5} # Add to spring length to get total length
self.spring_ut = {4.5: 1756, 5: 1758, 5.5: 1730, 6: 1697, 6.5:
    1665, 7: 1659, 7.5: 1628, 8: 1596, 8.5: 1574, 9: 1570, 9.5:
    1550, 10: 1523} # Stress at 10.000 openings. without K_i.
    Used as base line in fatigue calculations.

# Vertical and Horizontal tracks:
self.vertical_item_no = {"2250": "RSCV30Z", "2370": "RSCV40Z",
    "2500": "RSCV50Z", "2750": "RSCV60Z", "3000": "RSCV70Z"}
self.horizontal_item_no = {"2500": "resh250", "3000": "RS200H70
    "}
self.seal_track_item_no = {"2250": "1085 2260", "2370": "
    1085 2360", "2500": "1085 2510", "2750": "1085 3060", "3000"
    : "1085 3060"}


# Sectional Panel:
self.section61_kg_p_m = 6.4 # Weight per meter
self.section488_kg_p_m = 5.15 # Weight per meter
self.section61_height = 610
self.section488_height = 488

# Top/bottom Lists:
self.bottom_low = 40
self.bottom_med = 60
self.bottom_high = 70

# Bearing Plates
# Number of plates according to width of door:
self.width_for_3 = 3050
self.width_for_4 = 4050
self.width_for_5 = 5050
# Radius of the plates:
self.bearing_310_67 = 67.0 # 3.46 $
self.bearing_312R = 67.0   # 1.5  Centre holder
self.bearing_3086C = 86.0  # 4.57 $
self.bearing_3111C = 111.0 # 5.07 $
self.bearing_3127C = 127.0 # 5.41 $
self.bearing_3152C = 152.0 # 6.84 $
self.bearing_320_4 = 190.0 # 12.54 $
```

```
        # Top Roller Bracket
        self.top_bracket_415CZ = "adjustable"
        self.top_bracket_417 = "not adjustable"
        # Top roller bracket stainless steel
        self.top_bracket_415_304 = "adjustable"
        self.top_bracket_417_304 = "not adjustable"

        # Bottom Bracket
        self.bottom_bracket_421K = 300
        self.bottom_bracket_428TAI = 735

        # Intermediate Hinge
        self.max_space_intermediate_hinge = 1000
        #self.inter_hinge_450HZ = 1.12 # Price
        #self.inter_hinge_450H304 = 4.75 # Price

        # Hinge
        self.double_hinge = 5050

        # Spring Bumper
        self.spring_bumper_719 = 12.6
        self.spring_bumper_2100_15 = 0.283

        # Struts
        self.strut_screw = 300 # space between screws in struts

        # Mounting Plates
        # 3021HL = 1.66
        # 3022HD = 1.13

        # Paint the door in RAL
        self.paint_base = 6.25
        self.paint_base_price = 18000
        self.paint_m2_price = 1600
```

Appendix C

Graphical user interface code

```
class Feedback:

    def __init__(self, root):
        # Initialize spacing and title:
        root.title("Sectional Door Calculator v1.0")
```

```
pad_x = 12  # internal: ipadx, ipady
pat_y = 5
ipad_x = 7

# Label fields:
# Column 0
width_label = ttk.Label(root,text="Daylight Width (mm)")
width_label.grid(row=0,column=0,padx=pad_x,pady=pat_y,sticky="W
    ")
height_label = ttk.Label(root,text="Daylight Height (mm)")
height_label.grid(row=1,column=0,padx=pad_x,pady=pat_y,sticky="
    W")
life_label = ttk.Label(root,text="Life cycles")
life_label.grid(row=2,column=0,padx=pad_x,pady=pat_y,sticky="W"
    )
track_label = ttk.Label(root,text="Track type")
track_label.grid(row=3,column=0,padx=pad_x,pady=pat_y,sticky="W
    ")
hr_label = ttk.Label(root,text="Head room (mm)")
hr_label.grid(row=4,column=0,padx=pad_x,pady=pat_y,sticky="W")
hl_label = ttk.Label(root,text="High lift (mm)")
hl_label.grid(row=5,column=0,padx=pad_x,pady=pat_y,sticky="W")
pitch_label = ttk.Label(root,text="Pitch (Degrees)")
pitch_label.grid(row=6,column=0,padx=pad_x,pady=pat_y,sticky="W
    ")
manual_drum_label = ttk.Label(root,text="Manual drum selection"
    )
manual_drum_label.grid(row=7,column=0,padx=pad_x,pady=pat_y,
    sticky="W")
auto_drum_label = ttk.Label(root,text="Automatic drum selection
    ")
auto_drum_label.grid(row=8,column=0,padx=pad_x,pady=pat_y,
    sticky="W")
paint_label = ttk.Label(root,text="Color RAL")
paint_label.grid(row=9,column=0,padx=pad_x,pady=pat_y,sticky="W
    ")

# Column 2
operator_label = ttk.Label(root,text="Type of operator")
operator_label.grid(row=0,column=2,padx=pad_x,pady=pat_y,sticky
    ="W")
windows_label = ttk.Label(root,text="Windows")
windows_label.grid(row=1,column=2,padx=pad_x,pady=pat_y,sticky=
    "W")
springs_label = ttk.Label(root,text="Number of springs")
```

```
springs_label.grid(row=2,column=2,padx=pad_x,pady=pat_y,sticky=
    "W")
weight2_label = ttk.Label(root,text="Weight (kg/m2)")
weight2_label.grid(row=3,column=2,padx=pad_x,pady=pat_y,sticky=
    "W")
weight_label = ttk.Label(root,text="Weight (kg)")
weight_label.grid(row=4,column=2,padx=pad_x,pady=pat_y,sticky="
    W")
passdoor_label = ttk.Label(root,text="Passdoor")
passdoor_label.grid(row=5,column=2,padx=pad_x,pady=pat_y,sticky
    ="W")
stainless_label = ttk.Label(root,text="Stainless steel")
stainless_label.grid(row=6,column=2,padx=pad_x,pady=pat_y,
    sticky="W")
bottomlist_med_label = ttk.Label(root,text="Medium bottom list
    saves one section:")
#bottomlist_med_label.grid(row=7,column=2,padx=pad_x,pady=pat_y
    ,sticky="W")
bottomlist_hi_label = ttk.Label(root,text="High bottom list
    saves one section:")
#bottomlist_hi_label.grid(row=8,column=2,padx=pad_x,pady=pat_y,
    sticky="W")
passdoorLocation_label = ttk.Label(root,text="Pass door
    location:")
passdoorLocation_label.grid(row=7,column=2,padx=pad_x,pady=
    pat_y,sticky="W")
passdoorOpening_label = ttk.Label(root,text="Pass door opening:
    ")
passdoorOpening_label.grid(row=8,column=2,padx=pad_x,pady=pat_y
    ,sticky="W")
calc_label = ttk.Label(root,text="Calculate")
calc_label.grid(row=9,column=2,padx=pad_x,pady=pat_y,sticky="W"
    )

# Input fields:
# Column 1
self.width_input = ttk.Entry(root)
self.width_input.grid(row=0,column=1,padx=pad_x,pady=pat_y,
    sticky="W")
self.width_input.bind("<FocusOut>",self.CalcWeight)

self.height_input = ttk.Entry(root,text="Daylight Height input"
    )
self.height_input.grid(row=1,column=1,padx=pad_x,pady=pat_y,
    sticky="W")
self.height_input.bind("<FocusOut>",self.CalcWeight, add="+")
```

```
self.height_input.bind("<FocusOut>",self.AutoDrumSelection, add
    ="+")
self.height_input.bind("<FocusOut>",self.Sections, add="+")

self.cycles = StringVar()
self.life_input = ttk.Combobox(root,textvariable = self.cycles,
    state="readonly")
self.life_input.grid(row=2,column=1,padx=pad_x,pady=pat_y,
    sticky="W",ipadx=ipad_x)
self.life_input.config(values = ("10.000", "15.000", "25.000",
    "50.000", "100.000"))
self.life_input.current(1)

self.tracks = StringVar()
self.track_input = ttk.Combobox(root,textvariable = self.tracks
    ,state="readonly")
self.track_input.grid(row=3,column=1,padx=pad_x,pady=pat_y,
    sticky="W",ipadx=ipad_x)
self.track_input.config(values = ("Normal␣Lift␣(NL)", "Low␣Head
    ␣Room␣(LHR)", "High␣Lift␣(HL)", "Vertical␣Lift␣(VL)",
                            "Follow␣the␣Roof␣NL␣(FNL)", "Follow␣
                                the␣roof␣LHR␣(FLHR)", "Follow␣the␣
                                Roof␣HL␣(FHL)"))
self.track_input.current(0)
self.track_input.bind("<<ComboboxSelected>>",self.SetFromTrack,
    add="+")
self.track_input.bind("<FocusOut>",self.Sections, add="+")
self.track_input.bind("<FocusOut>",self.AutoDrumSelection, add=
    "+") # Change 17.08.16

self.hr_var = StringVar()
self.hr_var.set("0")
self.hr_input = ttk.Entry(root,textvariable = self.hr_var)
self.hr_input.grid(row=4,column=1,padx=pad_x,pady=pat_y,sticky=
    "W")
self.hr_input.bind("<FocusOut>",self.MinHR, add="+")
self.hr_input.bind("<FocusOut>",self.Sections, add="+")


self.hl_var = StringVar()
self.hl_var.set("0")
self.hl_input = ttk.Entry(root,textvariable = self.hl_var,state
    ="disabled")
self.hl_input.grid(row=5,column=1,padx=pad_x,pady=pat_y,sticky=
    "W")
self.hl_input.bind("<FocusOut>",self.SetFromTrack)
```

```
self.pitch_var = StringVar()
self.pitch_var.set("0")
self.pitch_input = ttk.Entry(root,textvariable = self.pitch_var
    ,state="disabled")
self.pitch_input.grid(row=6,column=1,padx=pad_x,pady=pat_y,
    sticky="W")

self.manual_drum = StringVar()
self.manual_drum_input = ttk.Combobox(root,textvariable = self.
    manual_drum,state="disabled")
self.manual_drum_input.grid(row=7,column=1,padx=pad_x,pady=
    pat_y,sticky="W",ipadx=ipad_x)
self.manual_drum_input.config(values = ("FFNL10","FFNL12","
    FFNL18","FFNL32"))
self.manual_drum_input.current(0)

self.auto_drum_var = IntVar()
self.auto_drum_var.set(1)
self.auto_drum_input = ttk.Checkbutton(root,variable=self.
    auto_drum_var,onvalue =1,comman=self.SetStateAutoDrum)
self.auto_drum_input.grid(row=8,column=1,padx=pad_x,pady=pat_y,
    sticky="W")

self.paint_var = StringVar()
self.paint_var.set("9002")
self.paint_input = ttk.Entry(root,textvariable = self.paint_var
    )
self.paint_input.grid(row=9,column=1,padx=pad_x,pady=pat_y,
    sticky="W")

# Column 3
self.operator = StringVar()
self.operator_input = ttk.Combobox(root,textvariable = self.
    operator,state="readonly")
self.operator_input.grid(row=0,column=3,padx=pad_x,pady=pat_y,
    sticky="W")
self.operator_input.config(values = ("Manual","Track/Pull","
    Axle"))
self.operator_input.current(0)

self.windows_var = StringVar()
self.windows_var.set("0")
self.windows_input = ttk.Entry(root,textvariable = self.
    windows_var)
```

```
self.windows_input.grid(row=1,column=3,padx=pad_x,pady=pat_y,
    sticky="W")

self.no_sprins_var = StringVar()
self.springs_input = ttk.Combobox(root,textvariable = self.
    no_sprins_var)
self.springs_input.grid(row=2,column=3,padx=pad_x,pady=pat_y,
    sticky="W")
self.springs_input.config(values = ("1","2","4"))
self.springs_input.current(1)

self.weight_var = StringVar()
self.weight_input = ttk.Entry(root,textvariable=self.weight_var
    )
self.weight_input.grid(row=4,column=3,padx=pad_x,pady=pat_y,
    sticky="W")


self.weight2_var = StringVar()
self.weight2_input = ttk.Combobox(root,textvariable = self.
    weight2_var)
self.weight2_input.grid(row=3,column=3,padx=pad_x,pady=pat_y,
    sticky="W")
self.weight2_input.config(values = ("10","10.5","11","11.5","12
    ","12.5","13","13.5","14"))
self.weight2_input.current(4)
self.weight2_input.bind("<FocusOut>",self.CalcWeight)

self.passdoor_var = IntVar()
self.passdoor_var.set(0)
self.passdoor_input = ttk.Checkbutton(root,onvalue=1,offvalue
    =0,variable = self.passdoor_var,comman=self.SetStatePassdoor
    )
self.passdoor_input.grid(row=5,column=3,padx=pad_x,pady=pat_y,
    sticky="W")

self.stainless_var = IntVar()
self.stainless_var.set(0)
self.stainless_input = ttk.Checkbutton(root,onvalue=1,offvalue
    =0,variable = self.stainless_var)
self.stainless_input.grid(row=6,column=3,padx=pad_x,pady=pat_y,
    sticky="W")

self.bottomlist_med_var = IntVar()
self.bottomlist_med_var.set(0)
```

```
        self.bottomlist_med_input = ttk.Checkbutton(root,onvalue=1,
            offvalue=0,variable = self.bottomlist_med_var)
        #self.bottomlist_med_input.grid(row=7,column=3,padx=pad_x,pady=
            pat_y,sticky="W")

        self.bottomlist_hi_var = IntVar()
        self.bottomlist_hi_var.set(0)
        self.bottomlist_hi_input = ttk.Checkbutton(root,onvalue=1,
            offvalue=0,variable = self.bottomlist_hi_var)
        #self.bottomlist_hi_input.grid(row=8,column=3,padx=pad_x,pady=
            pat_y,sticky="W")

        self.passdoorLocation_var = StringVar()
        self.passdoorLocation_input = ttk.Combobox(root,textvariable =
            self.passdoorLocation_var,state="disabled")
        self.passdoorLocation_input.grid(row=7,column=3,padx=pad_x,pady
            =pat_y,sticky="W")
        self.passdoorLocation_input.config(values = ("Left","Right"))

        self.passdoorOpening_var = StringVar()
        self.passdoorOpening_input = ttk.Combobox(root,textvariable =
            self.passdoorOpening_var,state="disabled")
        self.passdoorOpening_input.grid(row=8,column=3,padx=pad_x,pady=
            pat_y,sticky="W")
        self.passdoorOpening_input.config(values = ("Right␣hand","Left␣
            hand"))


        self.calc_input = ttk.Button(root,text="Click␣to␣calculate",
            command=self.Calculate)
        self.calc_input.grid(row=9,column=3,padx=pad_x,pady=pat_y,
            sticky="W")



# Calculates the weight of the door from weight per square meter.
def CalcWeight(self,FocusOut):
    if self.width_input.get() == "" or self.height_input.get() == "
        ":
        return()
    elif float(self.width_input.get()) <= 0 or float(self.
        height_input.get()) <= 0:
        raise ValueError("Width␣and␣height␣must␣be␣positive␣and␣
            larger␣then␣0")
    else:
```

```
        while True:
            try:
                float(self.width_input.get())
                float(self.height_input.get())
                break
            except ValueError:
                print("Height and width need positive numbers")

        width = float(self.width_input.get())
        height = float(self.height_input.get())
        kg_2 = float(self.weight2_var.get())
        weight = width * height * kg_2 / 10**6
        self.weight_var.set(weight)

# Set state of passdoor location and opening if passdoor is checked
def SetStatePassdoor(self):
    if self.passdoor_var.get() == 0:
        self.passdoorLocation_input.configure(state="disabled")
        self.passdoorOpening_input.configure(state="disabled")
    else:
        self.passdoorLocation_input.configure(state="readonly")
        self.passdoorOpening_input.configure(state="readonly")


# Set state of manual drum selection if automatic drum selection is
    active
def SetStateAutoDrum(self):
    if self.auto_drum_var.get() == 1:
        self.manual_drum_input.configure(state="disabled")
    else:
        self.manual_drum_input.configure(state="readonly")

# Set values based on track type
def SetFromTrack(self,ComboboxSelected):
    if self.track_input.get() == "High Lift (HL)" or self.
       track_input.get() == "Follow the Roof HL (FHL)":
        self.manual_drum_input.config(values = ("FFHL54","FFHL120",
            "FFHL164"))
        self.hl_input.configure(state="normal")
        self.manual_drum_input.current(0)
        self.hr_input.configure(state = "disabled")
        self.hr_var.set(self.hl_input.get())
        self.minhr = 0
        self.pitch_var.set("0")
        if self.track_input.get() == "Follow the Roof HL (FHL)":
            self.pitch_input.configure(state = "normal")
```

```
        elif self.track_input.get() == "Vertical Lift (VL)":
            self.hl_var.set("0")
            self.pitch_var.set("0")
            self.manual_drum_input.config(values = ("FFVL11","FFVL18","
                FFVL28"))
            self.manual_drum_input.current(0)
            self.hl_input.configure(state = "disabled")
            self.pitch_input.configure(state = "disabled")
            self.hr_input.configure(state = "disabled")
            self.hr_var.set(self.height_input.get())
            self.minhr = 0
        else:
            self.hl_var.set("0")
            self.pitch_var.set("0")
            self.manual_drum_input.config(values = ("FFNL10","FFNL12","
                FFNL18","FFNL32"))
            self.manual_drum_input.current(0)
            self.hl_input.configure(state="disabled")
            self.hr_input.configure(state = "normal")

            if self.track_input.get() == "Follow the Roof NL (FNL)" or
                self.track_input.get() == "Follow the roof LHR (FLHR)":
                 self.pitch_input.configure(state = "normal")
            else:
                 self.pitch_input.configure(state = "disabled")
                 self.manual_drum_input.current(0)

    # Select the correct drum based on track type, height and hi lift.
    def AutoDrumSelection(self,FocusOut):
        if self.height_input.get() == "":
            return()
        else:
            if self.track_input.get() == "High Lift (HL)" or self.
                track_input.get() == "Follow the Roof HL (FHL)":
                 if Data().drum_FFHL54["max opening"] >= float(self.
                     height_input.get()) and Data().drum_FFHL54["max hi
                     lift"] >= float(self.hl_var.get()):
                      self.manual_drum_input.config(values = ("FFHL54","
                          FFHL120","FFHL164"))
                      self.manual_drum_input.current(0)
                      return
                 elif Data().drum_FFHL120["max opening"] >= float(self.
                     height_input.get()) and Data().drum_FFHL120["max hi
                     lift"] >= float(self.hl_var.get()):
                      self.manual_drum_input.config(values = ("FFHL120","
                          FFHL164"))
```

```
            self.manual_drum_input.current(0)
            return
        elif Data().drum_FFHL164["max opening"] >= float(self.
            height_input.get()) and Data().drum_FFHL164["max hi
            lift"] >= float(self.hl_input.get()):
            self.manual_drum_input.config(values = ("FFHL164"))
            self.manual_drum_input.current(0)
            return
        else:
            print("High lift or height is to high for drums")
            raise ValueError("No drum in database for the
                specified height or hi lift size (GUI)")

    if self.track_input.get() == "Vertical Lift (VL)":
        if Data().drum_FFVL11["max opening"] >= float(self.
            height_input.get()):
            self.manual_drum_input.config(values = ("FFVL11","
                FFVL18","FFVL28"))
            self.manual_drum_input.current(0)
            return
        elif Data().drum_FFVL18["max opening"] >= float(self.
            height_input.get()):
            self.manual_drum_input.config(values = ("FFVL18","
                FFVL28"))
            self.manual_drum_input.current(0)
            return
        elif Data().drum_FFVL28["max opening"] >= float(self.
            height_input.get()):
            self.manual_drum_input.config(values = ("FFVL28"))
            self.manual_drum_input.current(0)
            return
        else:
            print("Height over range of drums")
            raise ValueError("Height over range of drums (GUI)"
                )
    else:
        if  Data().drum_FFNL10["max opening"] >= float(self.
            height_input.get()):
            self.manual_drum_input.config(values = ("FFNL10","
                FFNL12","FFNL18","FFNL32"))
            self.manual_drum_input.current(0)
        elif Data().drum_FFNL12["max opening"] >= float(self.
            height_input.get()):
            self.manual_drum_input.config(values = ("FFNL12","
                FFNL18","FFNL32"))
            self.manual_drum_input.current(0)
```

```
            elif Data().drum_FFNL18["max opening"] >= float(self.
                height_input.get()):
                 self.manual_drum_input.config(values = ("FFNL18","
                     FFNL32"))
                 self.manual_drum_input.current(0)
            elif Data().drum_FFNL32["max opening"] >= float(self.
                height_input.get()):
                 self.manual_drum_input.config(values = ("FFNL32"))
                 self.manual_drum_input.current(0)
            else:
                 print("Height over range of drums")
                 raise ValueError("Height over range of drums (GUI)"
                     )

def MinHR(self,FocusOut):
    if self.height_input.get() == "" or self.hr_input.get() == "0"
        or self.width_input.get() == "":
         return()
    elif float(self.height_input.get()) <= Data().drum_FFNL12["max
        opening"]:
         self.minhr = 200
    elif self.track_input.get() == "Low Head Room (LHR)" or self.
        track_input.get() == "Follow the roof LHR (FLHR)":
         self.minhr = 100
    else:
         self.minhr = 380

def Sections(self,FocusOut):
    if self.height_input.get() == "" or self.hr_input.get() == "0"
        or self.width_input.get() == "":
         return()
    Section_610 = Data().section61_height
    Section_488 = Data().section488_height
    bottom_list_low = Data().bottom_low
    bottom_list_medium = Data().bottom_med
    bottom_list_high = Data().bottom_high

    if self.track_input.get() == "Low Head Room (LHR)" or self.
        track_input.get() == "Follow the roof LHR (FLHR)":
         width_add_to_day = 3.5
    else:
         width_add_to_day = 5

    # correct to working sizes
    self.height_working = float(self.height_input.get())
        bottom_list_low
```

```
        self.width_working = float(self.width_input.get()) +
            width_add_to_day
        # initiate for calculating number of sections
        self.count_section = ceil(self.height_working / Section_610)
        count_section_floor = floor(self.height_working / Section_610)

        # ATH
        if (Data().bottom_med    Data().bottom_low) / Section_610 >= (
            self.height_working / Section_610)    count_section_floor:
             self.bottomlist_med_var.set(1)
        elif (Data().bottom_high    Data().bottom_low) / Section_610 >=
            (self.height_working / Section_610)    count_section_floor:
             self.bottomlist_hi_var.set(1)


        calc_height = self.count_section * Section_610
        self.count_610 = self.count_section
        self.count_488 = 0
        section_difference = Section_610    Section_488
        # Calculate the number of sections
        while(calc_height    section_difference > self.height_working):
             calc_height = calc_height    section_difference
             self.count_488 += 1
             self.count_610  = 1

        self.height_total = self.count_488*Section_488 + self.count_610
            *Section_610 + bottom_list_low
        # Check to see if top sections needs to be cut
        self.saw_off_top_section = 0
        if self.height_total + self.minhr > float(self.hr_input.get())
            + float(self.height_input.get()):
             self.saw_off_top_section = self.height_total    float(self.
                 height_input.get())
             #print("saga af efsta fleka {} mm".format(self.
                 saw_off_top_section)) ####################  print
                 """""""""""""""
             self.height_total = self.height_total    self.
                 saw_off_top_section



# Calculates all components
def Calculate(self):
    width = float(self.width_input.get())
    height = float(self.height_input.get())
    weight = float(self.weight_input.get())
```

```
        spring_info = spring(height, width, weight,
                float(self.life_input.get()) * 10**3, self.
                    manual_drum_input.get(), self.track_input.get(),
                    self.count_610, float(self.springs_input.get()),
```

Appendix D

Component calculation code

```
# Find correct struts
def Strut(width):
    if width <= 3500:
        strut = ""
    elif width <= 4020:
        strut = "65S4020"
    elif width <= 4520:
        strut = "65S4520"
    elif width <= 6020:
        strut = "65S6020"
    elif width <= 6520:
        strut = "110S6520"
    else:
        strut = "special order"
    return(strut)


# Find the correct vertical track
def Vertical_track(height_day,track_type,high_lift):
    if track_type == "Normal Lift (NL)" or track_type == "Low Head Room
       (LHR)" or track_type == "Follow the Roof NL (FNL)" or
       track_type == "Follow the roof LHR (FLHR)":
        if height_day <= 2250:
            vertical_track = "2250"
        elif height_day <= 2370:
            vertical_track = "2370"
        elif height_day <= 2500:
            vertical_track = "2500"
        elif height_day <= 2750:
            vertical_track = "2750"
        elif height_day <= 3000:
            vertical_track = "3000"
        else:
            vertical_track = "industrial"
    elif track_type == "High Lift (HL)" or track_type == "Follow the
       Roof HL (FHL)":
```

```
        if high_lift + height_day <= 2250:
            vertical_track = "2250"
        elif high_lift + height_day <= 2370:
            vertical_track = "2370"
        elif high_lift + height_day <= 2500:
            vertical_track = "2500"
        elif high_lift + height_day <= 2750:
            vertical_track = "2750"
        elif high_lift + height_day <= 3000:
            vertical_track = "3000"
        else:
            vertical_track = "industrial"
    else:
        vertical_track = "industrial"
    return(vertical_track)


# Fix size of vertical track for head room
def VerticalTrackSaw(vertical_track,height,hr,min_hr):
    if vertical_track == "industrial":
        return(0)
    vertical_track = float(vertical_track)
    if vertical_track    height + min_hr <= hr:
        saw = 0
    else:
        saw = vertical_track    height
    return(saw)

# Find the correct horizontal track
def Horizontal_track(height_day,track_type,hi_lift):
    if track_type == "High Lift (HL)"  or track_type == "Follow the
       Roof HL (FHL)":
        height_day = height_day    hi_lift
    if track_type == "Vertical Lift (VL)":
        horizontal_track = ""
    else:
        if height_day <= 2500:
            horizontal_track = "2500"
        elif height_day <= 3000:
            horizontal_track = "3000"
        else:
            horizontal_track = "industrial"
    return(horizontal_track)

# Find correct shaft
def Shaft(width,weight):
```

```
    if width <= 2500 and weight <= 240:
        shaft = "701 2750Z"
        number = 1
    elif width <= 3250 and weight <= 240:
        shaft = "701 3500Z"
        number = 1
    elif width <= 4250:
        shaft = "705GB4500"
        number = 1
    elif width <= 5000 and weight <= 240:
        shaft = "701 2750Z"
        number = 2
    elif width <= 6500 and weight <= 240:
        shaft = "701 3500Z"
        number = 2
    elif width <= 8500:
        shaft = "705GB4500"
        number = 2
    elif width <= 12750:
        shaft = "705GB4500"
        number = 3

    return(shaft,number)

# Find the bearing plate type and ammount
def BearingPlates(spring_inner,width,drum):
    drum = Drum(drum)
    # Number of plates
    if Data().width_for_3 >= width:
        plate_no = 2
    elif Data().width_for_4 >= width:
        plate_no = 3
    elif Data().width_for_5 >= width:
        plate_no = 4
    else:
        plate_no = 5
    # Type of plate
    if spring_inner <= 50.8 and drum["Max. outside diameter"] <= Data()
       .bearing_310_67*2:
        plate_type = "310LH RH" # 310 67
        hr_min = 200
    elif spring_inner <= 66.7 and drum["Max. outside diameter"] <= Data
       ().bearing_3086C*2:
        plate_type = "3086C"
        hr_min = 260
```

```
    elif spring_inner <= 95.3 and drum["Max. outside diameter"] <= Data
        ().bearing_3111C*2:
        plate_type = "316 4B" # 3111C
        hr_min = 311
    elif spring_inner <= 95.3 and drum["Max. outside diameter"] <= Data
        ().bearing_3127C*2:
        plate_type = "3127C"
        hr_min = 312.7
    elif spring_inner <= 95.3 and drum["Max. outside diameter"] <= Data
        ().bearing_3152C*2:
        plate_type = "3152C"
        hr_min = 315.2
    elif spring_inner <= 95.3 and drum["Max. outside diameter"] <= Data
        ().bearing_320_4*2:
        plate_type = "320 4"
        hr_min = 360
    return(plate_no,plate_type,hr_min)


def TopRollerBracket(track_type,height_door,vertical_track,saw,
    stainless):
    if vertical_track == "industrial":  # Always adjustable
        if stainless == 1:
            top_roller = "415 304"
        else:
            top_roller = "415"
        return(top_roller)
    vertical_track = float(vertical_track)   saw
    if track_type == "Normal Lift (NL)" or track_type == "Follow the
        Roof NL (FNL)":
        if height_door < vertical_track: # Adjustable bracket
            if stainless == 1:
                top_roller = "415 304"
            else:
                top_roller = "415"
        else: # Not adjustable
            if stainless == 1:
                top_roller = "417 304"
            else:
                top_roller = "417"
    elif track_type == "Low Head Room (LHR)" or "Follow the roof LHR (
        FLHR)": # Always not adjustable
        if stainless == 1:
            top_roller = "417 304"
        else:
            top_roller = "417"
```

```
    else: # Always adjustable
        if stainless == 1:
            top_roller = "415 304"
        else:
            top_roller = "415"
    return(top_roller)

def BottomBracket(width,weight,stainless):
    if stainless == 1 or width >= 3000 or weight >= Data().
       bottom_bracket_421K:
        bottom_bracket = "428TAI"
    else:
        bottom_bracket = "421K"
        if weight > Data().bottom_bracket_428TAI:
            print("Weight of door exceeds bottom bracket needs special
                order")
    return(bottom_bracket)

def DoorLock(motor): # Also applies to door handle
    if motor == "Manual":
        lock = True
    else:
        lock = False
    return(lock)

def IntermediateHinge(width,sections_no):
    intermediate_hinge = (ceil(width/Data().
       max_space_intermediate_hinge)   1) * (sections_no   1)
    return(intermediate_hinge)

def Hinge(width,sections_no):
    hinge = 4 * (sections_no   1) if width > Data().double_hinge else 2
        * (sections_no   1)
    return(hinge)

def Roller(hinge,width):
    if width > Data().double_hinge:
        roller_size = "long"
        roller = (hinge/2) + 4
    else:
        roller_size = "short"
        roller = hinge + 4
    return(roller,roller_size)

def SpringBumper(motor,track_type):
```

```
    if track_type == "Normal Lift (NL)" or track_type == "Low Head Room
        (LHR)":
         if motor == "Axle":
              spring_bumper = True
              return(spring_bumper)

    spring_bumper = False
    return(spring_bumper)

def Screw(hinge,intermediatehinge,strut,sections_no,width):
    if strut == "":
        screw = 6*hinge + 4*intermediatehinge + 20
    else:
        screw = 6*hinge + 4*intermediatehinge + width/Data().
            strut_screw * 2 * sections_no + 20
    return(screw)

def Bolt6mm(hinge):
    bolt = hinge * 2
    return(bolt)

def Bolt8mm(spring_no):
    bolt = spring_no * 2
    return(bolt)

def BoltTracks(hr,weight):
    if hr <= 380: # Able to use Mounting plates 3021HL and 3022HD
        bolt = 4
    else:
        if weight <= 100: # use one angle on each horizontal track
            bolt = 6
        else:
            bolt = 8 # use two angle on each horizontal track
    return(bolt)

def StopRing(spring_no):
    if spring_no == "1":
        stop_ring = 1
    else:
        stop_ring = 0
    return(stop_ring)

def Paint(width,height,paint):
    if paint == "9002":
        return(0)
    w = width/1000
```

```
    h = height/1000
    if w * h <= Data().paint_base:
        price = Data().paint_base_price
    else:
        price = Data().paint_base_price + ((w*h)    Data().paint_base) *
            Data().paint_m2_price
    return(price)

def ImportCSV():
    ###Imports prices and item numbers from data.csv###
    with open("voruskra17.csv",encoding="utf 8",errors="ignore") as
        csvfile:
        reader = csv.reader(csvfile,delimiter=";")
        items = []
        prices = []
        for row in reader:
            item = row[0]
            price = float(row[1].replace(",","."))
            items.append(item[3:])
            prices.append(price)
        return(items,prices)

def WriteCSVInventory(width, working_width, height, height_total, hr,
    saw_top, windows, count_610, count_488, paint, spring, no_springs,
            wire_length, strut, shaft, bearing_plate, track_vertical,
                track_horizontal, drum, top_bracket, bottom_bracket,
            intermediate_hinge, hinge, stainless, roller, door_lock,
                spring_bumper, screw, bolt6mm, bolt8mm, bolt_tracks,
                name, passdoor,
            passdoor_location, passdoor_opening):
    ### Writes results in csv Manufacturing sheet ###
    with open("C:\Doors\ " + name + "_INV" + ".csv", "w", newline="")
        as csvfile:
        writer = csv.writer(csvfile,delimiter=";")
        writer.writerow(["Name"] + [""])
        writer.writerow(["Address"] + [""])
        writer.writerow(["Contact"] + [""])
        writer.writerow(["Phone"] + [""])
        writer.writerow([""] + [""])
        writer.writerow([""] + ["Width"] + ["Height"])
        writer.writerow(["Daylight size of opening"] + [width] + [
            height])
        writer.writerow(["Cut down size"] + [working_width])
        writer.writerow(["Number of 610mm sections"] + [count_610])
        writer.writerow(["Number of 488mm sections"] + [count_488])
        if passdoor == 1:
```

```
        writer.writerow(["Pass door in door. "] + ["Location: " +
            passdoor_location] + ["Opening: " + passdoor_opening])
    writer.writerow(["Head room"] + [hr])
    if saw_top != 0:
        writer.writerow(["Saw of top section"] + [saw_top])
    writer.writerow(["Total height"] + [height_total])
    writer.writerow(["Color of door RAL"] + [paint])
    if windows != "0":
        writer.writerow(["Windows"] + [windows])
    writer.writerow(["Springs"] + [no_springs] + [spring[0:4]])
    writer.writerow(["Wire"] + [str(spring[5]) + "mm"] + [
        wire_length])
    if strut != "":
        writer.writerow(["Strut type"] + [strut] + [str(count_488+
            count_610 1)])
    if no_springs == 1:
        writer.writerow(["Stop ring on shaft"] + [1])
    writer.writerow(["Shaft"] + [shaft[0]] + [str(shaft[1]) + " PCS
        "])
    writer.writerow(["Bearing plate"] + [bearing_plate[1]] + [
        bearing_plate[0]])
    writer.writerow(["Wire drum"] + [drum])
    writer.writerow(["Vertical track"] + [track_vertical])
    writer.writerow(["Horizontal track"] + [track_horizontal])
    writer.writerow(["Seals on vertical track"] + [2])
    writer.writerow(["Top roller bracket"] + [top_bracket])
    writer.writerow(["Bottom roller bracket"] + [bottom_bracket])
    writer.writerow(["Intermediate hinge"] + [intermediate_hinge] +
        ["Stainless" if stainless == 1 else ""])
    writer.writerow(["Hinge"] + [hinge] + ["Stainless" if stainless
        == 1 else ""])
    writer.writerow(["Roller bracket on hinge"] + [hinge] + ["
        Stainless" if stainless == 1 else ""])
    writer.writerow(["Roller"] + [roller[0]] + [roller[1]] + ["
        Stainless" if stainless == 1 else ""])
    if door_lock == True:
        writer.writerow(["Lock on door"] + ["1"])
    writer.writerow(["Door handle"] + ["1"])
    if spring_bumper == True:
        writer.writerow(["Spring bumper"] + ["pair"] + ["Brackets
            for bumper"])
    else:
        writer.writerow(["Rubber end stop"] + ["2"])
    writer.writerow(["Screw 6,3x35"] + [screw] + ["Stainless" if
        stainless == 1 else ""] )
```

```
        writer.writerow(["Bolt 6mm"] + [bolt6mm] + ["Stainless" if
            stainless == 1 else ""])
        writer.writerow(["Bolt 8mm"] + [bolt8mm])
        writer.writerow(["Track bolts"] + [bolt_tracks])
        writer.writerow(["6mm nut"] + [bolt6mm+bolt_tracks] + ["
            Stainless" if stainless == 1 else ""])
        writer.writerow(["8mm nut"] + [bolt8mm])
        writer.writerow(["Optional manual"] + [""])

def WriteCSVQuote(width, height, hr, color, stainless, track, motor,
    windows, passdoor, name, price,passdoor_location,passdoor_opening):
    ### Writes results in csv Quote sheet ###
    with open("C:\Doors\ " + name + "_Quote" + ".csv", "w", newline="")
        as csvfile:
        writer = csv.writer(csvfile,delimiter=";")
        writer.writerow(["Invoice for a sectional overhead door"])
        writer.writerow(["Size of daylight opening " + "Width: " + str(
            width) + "mm" + " Height: " + str(height) + "mm"])
        writer.writerow(["Size from highest point of daylight opening
            to lowest point on roof: " + str(hr) + "mm" + "(head room)"
            ])
        writer.writerow(["Track opening: " + track])
        writer.writerow(["Color of door RAL: " + color])
        if stainless == 1:
            writer.writerow(["Stainless steel fittings"])
        if windows != "0":
            writer.writerow(["Number of windows: " + windows])
        if passdoor == 1:
            writer.writerow(["Pass door in door. Location: " +
                passdoor_location + ". Opening: " + passdoor_opening])
        if motor == "Manual":
            writer.writerow(["Manual opening"])
        elif motor == "Track/Pull":
            writer.writerow(["Automatic pull operator"])
        elif motor == "Axle":
            writer.writerow(["Automatic axle/shaft operator"])
        writer.writerow(["Price of the door: " + str(int(price)) + " ,
            kr"])



def CalculatePrice(width,count_488,count_610,spring,no_spring,strut,
    vertical_track,horizontal_track,bearing_plate,vertical_saw,shaft,
                    top_roller_bracket, bottom_bracket, door_lock,
                        intermediate_hinge, hinge, spring_bumper, screw,
```

```
                    bolt6mm, bolt8mm,
                bolt_tracks, stop_ring, stainless, paint,height,
                    passdoor):
###Calculates the price of all components###
item_prices = ImportCSV()
items = item_prices[0]
prices = item_prices[1]

# Sections:
section_488P = prices[items.index("1002")] * width/1000 * count_488
section_610P = prices[items.index("1000")] * width/1000 * count_610
if width < Data().double_hinge:
    endcap_488P =  prices[items.index("40E488")] * count_488
    endcap_610P =  prices[items.index("40E610")] * count_610
else:
    endcap_488P =  prices[items.index("40ED500")] * count_488
    endcap_610P =  prices[items.index("40ED610")] * count_610

top_bottom_listP = prices[items.index("1038 6090")] * width/1000 *
    2 # 1038H6090 for 55mm    1040 6090 for high
bottom_rubberP = prices[items.index("1037")] * width/1000 # 1035
    for double rubber soft
top_rubberP = prices[items.index("1036 36")] * width/1000
price_sections = section_488P + section_610P + endcap_488P +
    endcap_610P + top_bottom_listP + bottom_rubberP + top_rubberP

# Springs:
springP = prices[items.index(Data().spring_item_no[spring[0]])] *
    spring[2]/1000 * no_spring
spring_fittingsP = prices[items.index(Data().
    spring_fittings_item_no[spring[1]])] * no_spring/2
price_springs = springP + spring_fittingsP

# Struts:
if strut == "":
    strutP = 0
elif strut == "special_order":
    strutP = 0
else:
    strutP = prices[items.index(strut)] * (count_488 + count_610
        1)

# Vertical and Horizontal tracks:
if vertical_track == "industrial":
    verticalP = 0
    sealP = 0
```

```
else:
    verticalP = prices[items.index(Data().vertical_item_no[
        vertical_track])]
    sealP = prices[items.index(Data().seal_track_item_no[
        vertical_track])]
if horizontal_track == "industrial":
    horizontalP = 0
elif horizontal_track == "":
    horizontalP = 0
else:
    horizontalP = prices[items.index(Data().horizontal_item_no[
        horizontal_track])]
if vertical_saw == 0:
    vertical_sawP = 0
else:
    vertical_sawP = prices[items.index("vinna")] * 1/6 # 10 minutes
price_tracks = verticalP + horizontalP + sealP + vertical_sawP

#print("horizontal track {}".format(horizontal_track))
#print("Horizontal price {}".format(horizontalP))
#print("Vertical price {}".format(verticalP))

# Bearing plates:
bearing_plateP = prices[items.index(bearing_plate[1])] *
    bearing_plate[0]

# Shaft:
shaftP = prices[items.index(shaft[0])] * shaft[1]
couplerP = prices[items.index("708 90")] * (shaft[1]    1) # 708S90
    Flexiforce
price_shaft = shaftP + couplerP

# Roller brackets and hinges:
top_rollerP = prices[items.index(top_roller_bracket)]
bottom_bracketP = prices[items.index(bottom_bracket)]
if stainless == 1:
    hingesP = prices[items.index("450C304")] * hinge
    intermediateP = prices[items.index("450H304")] *
        intermediate_hinge
else:
    hingesP = prices[items.index("450CZ")] * hinge
    intermediateP = prices[items.index("450HZ")] *
        intermediate_hinge
price_brackets = top_rollerP + bottom_bracketP + hingesP +
    intermediateP
```

```
# Misc:
if door_lock == True:
    door_lockP = prices[items.index("629VER")]
else:
    door_lockP = 0

if spring_bumper == True:
    spring_bumperP = prices[items.index("718")] * 2
else:
    spring_bumperP = 0

if passdoor == 1:
    passdoorP = prices[items.index("005")]
else:
    passdoorP = 0
#print(passdoorP)

stop_ringP = prices[items.index("1065")] * stop_ring
price_misc = door_lockP + spring_bumperP + stop_ringP + passdoorP
```