1. Introduction

This paper is a brief documentation and is just preliminary and hence all feedback is very welcome. Main task is now to get the program as good as possible and add all wanted features by you guys. So use the report tab to report errors or features wanted, if you also can refer to line number it would be great.

Hope we all can gain better insight in whats going on under a launch, and that we can cooperate to achieve this.

Thanks to all that have helped out so far.(An acknowledgement will be added at some point)

2. Discussion

2.1. Introduction

As mention earlier the analytical solution will be a very simplified analysis and only considered a few effects during a launch. The path is considered to be ideal and therefore there is no need for looking at the moments of the plane and no regulation loop of inputs. This means that the acceleration can be integrated twice to find the path of the launch. The acceleration is given by the sum of the forces on the plane. The typical launch path is given in figure 2.1, however the different force involves a large number of parameters were several of them are dependent on each others. Typically for aeroplanes it would be solved in a airplane coordinate system and then transformed into a earth frame, but since the winch is fixed to ground the forces are transformed into vectors in x and z direction of the global coordinate system.

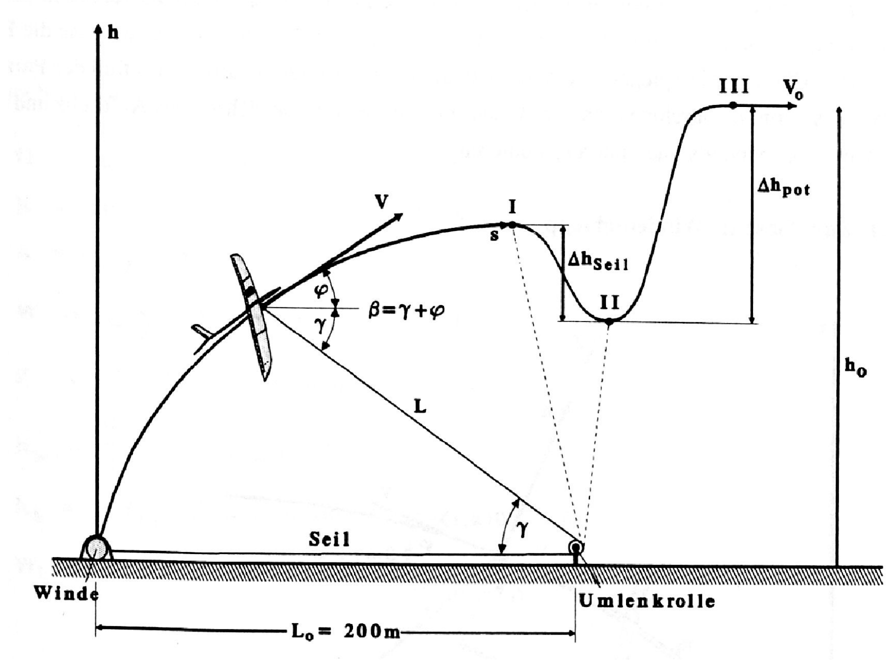

Figure 2.1.: Launch path.

2.2. Position

2.2.1. Acceleration

The position or the path can be found by integration of acceleration and velocities. As mentioned above this paper is only looking at the forces, and by using newtons second law, $F = m \cdot a$ the acceleration can be found. There exist several num,erical integration methods, however due to the fact that there is several phases and events a forward euler methos was chosen to be sure that the simulation will produce a result when the equations are altered. The derivation of the forces in x,z direction will be discussed later in the forces section.

2.2.1.1. Total velocity

When the acceleration is integrated the velocity is found. However to find the total velocity it is also neccesary to add the external disturbances such as thermic and wind.

The wind speed vary with the height and there is no certain data available and this needs to be calculated on the fly with the help of sensoric. However for the simulation a simplified model, will be used, the wind speed is estimated as a polynomial as shown in equation 2.1.

$$vw = vw_{ref} \cdot \left(\frac{z}{2}\right)^{\frac{1}{7}} \tag{2.1}$$

Here is to be noted that the reference speed, typicaly the wind speed is measured in the aproximity of the ground, and by using this value togehter with the height of the measurement the reference wind speed can be found see equation 2.2.

$$vw_{ref} = \frac{vw}{\left(\frac{z}{2}\right)^{\frac{1}{7}}} \tag{2.2}$$

This means that the velocity of the wind is zero at ground, which correspond to a no slip condition.

For the thermic a linear model is assumed, where the thermic is known at given height. Up to this height the thermic increases linearly, see equation 2.3.

$$vt = \frac{vt_{ceil} \cdot z}{z_{ceil}} \tag{2.3}$$

It is worth mentioning that it is only considered thermic and head wind, sink and tail wind can be found by setting in the reference speeds with negative signs. Crosswind and more complex windmodels are not implemented at this stage.

By integrating the velocity the position or the path can be found, in addition is also needed to know which phase the plane is in since this alters the force and angles equations.

2.3. Phases

As seen in figure 2.1, the launch path is continues but not homegenous. The path is therefore divided into several phases where a given configuration is valid.

Dependent of aircraft configuration and launch configuration not all of the phases needs to be used during a winch launchwhich is illustrated in figure ,.

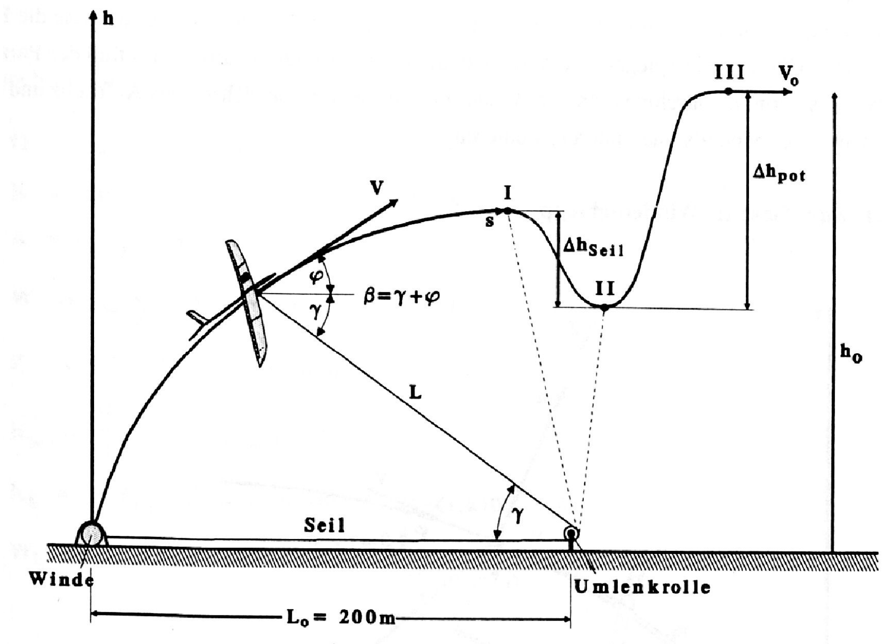

Figure 2.2.: Launch path.

As seen in figure , 3 different alternatives of launch teqniques is sketched :

1. full size gliders
2. a typically F3B launch
3. a radical launch for F3J

The effect of these launch teqniques will be evaluated later.In addition to launch alternative there aslo occur events which might initiate a phase or alter the launch configuration, however events will be explained later in this chapter.

2.3.1. Phase 0

In this phase the line will be pretension by starting the winch which the starts winding up the line, therefore the plane velocities is kept at zero.

2.3.1.1. Phase 1

Here the glider release the brakes and starts rolling on the ground until it takes off. The velocity is in this phase is only kept at zero for the z-axis. However for a F3B or similar glider, which does not have a undercarriage this phase will consist of the launch of the glider with a given angle and start speed, i.e only one timestep.

2.3.1.2. Phase 2

The climb phase is the second phase, this phase is characterized by a high lift configuration phase,

2.3.1.3. Phase 3

In this phase the elastic energy stored in the line will be transferred in to kinetic energy.

2.3.1.4. Phase 4

This is another climb phase, but this time the plane is not attached to winch line and the kinetic energy is transferred into additional height.

2.3.1.5. Phase 5

The winch is ended by the phase 5, this phase is dependent of the task that is going to be flown after the launch and the weather conditions.

2.4. Forces

The forces are differnt in each phase, for instance in phase 4 there is no lineforce, in phase 3 there is almost no line drag, in phase 1 the wheel friction and the lineforce is dominant. Thge forces in phase 2 is illustrated in figure 2.3, this is similar to the other other phase, where one or more forces is to be omitted or added.

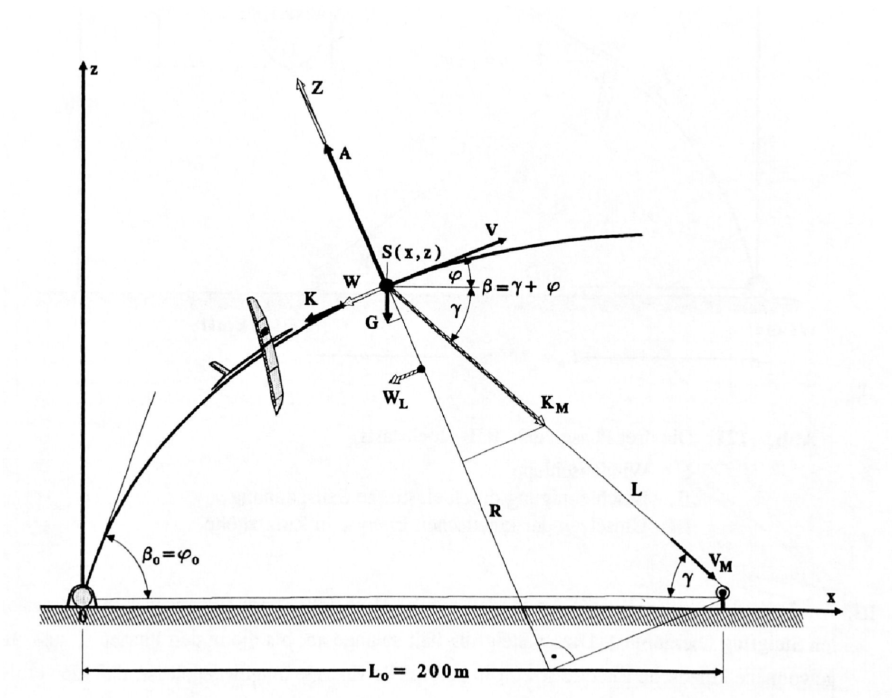

Figure 2.3.: Force in phase 1.

The forces in the figure 2.3 is listed in equation and consist of four forces, gravity,line,drag and lift force. However these forces are not independent of each other and have several parameters which influence them.

$$G = m \cdot g \tag{2.4}$$

$$L = \frac{C_{LA} \cdot \rho \cdot A \cdot v^2}{2} \tag{2.5}$$

$$D = (C_{DP} \cdot S_P + C_{DL} \cdot S_L) \cdot \frac{\rho \cdot v^2}{2} \tag{2.6}$$

$$F_L = F_M + F_E \tag{2.7}$$

2.4.1. Gravity force

The gravity force always works in negative global z axis, however the mass is dependent of several parameters, which are shown in equation

$$G = A \cdot WL \cdot g \tag{2.8}$$

$$G = \frac{b^2 \cdot WL \cdot g}{\Lambda} \tag{2.9}$$

2.4.2. Lift force

The lift coefficient can be estimated by a linear approximation for different angle of attacks, see equation 2.10. Close to stall this linear approximation is not valid, this means in a launch configuration when the pilot is pushing the limit this might yield in inaccurate results. For a typical F3B profile, HQW2/8 the curve can been seen in figure 2.4, with 5 degree positive flap.

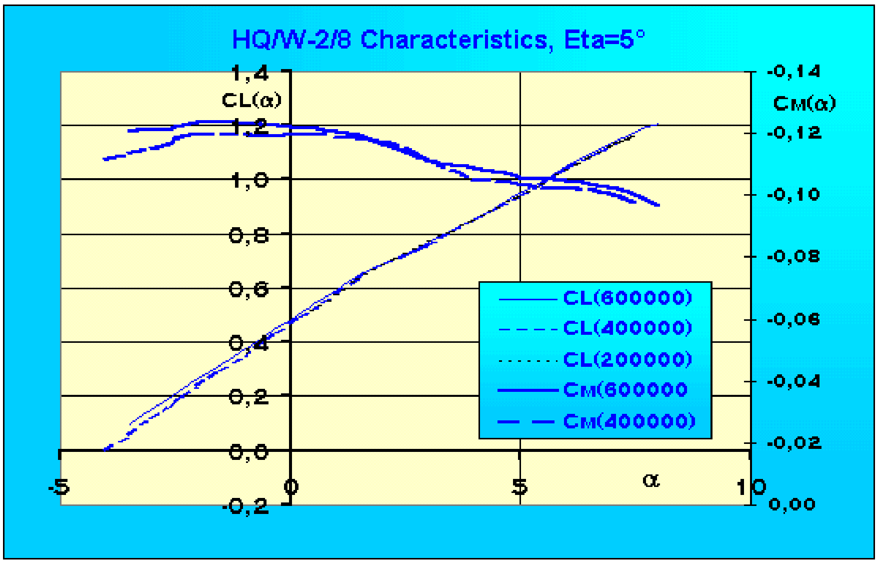

Figure 2.4.: Polars for HQW2/8 with start flap setting .

$$C_L = C_L' \cdot \alpha \tag{2.10}$$

For the case of flat plate $C_L' = 2 \cdot \pi$, so as seen by figure 2.4 this coeffsient will be reduced for a profile, this yields to a lower lift, see equation 2.11, where $e < 1$.

$$C_L = 2 \cdot \pi \cdot e \cdot \alpha \tag{2.11}$$

However the lift is altered when using differnt flap setting, as seen in figure 2.5, where the curve is parellell to the one in figure 2.4, but where the crossing point with the Cl axis is shifted downwards. In addition the stall point is shifted to a higher angle of attack.

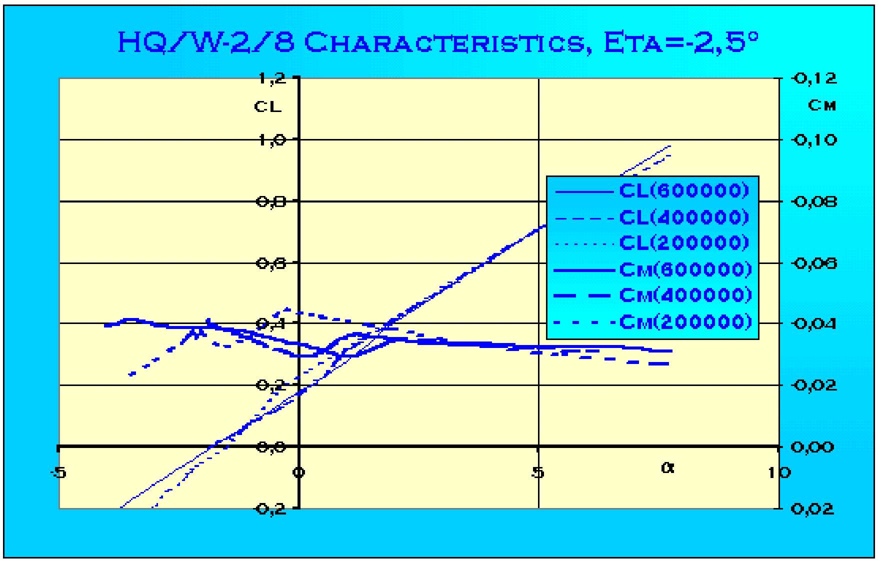

Figure 2.5.: Polars for HQW2/8 with speed flap setting .

So by combing all this information a very accurate linear approximation for the lift coeffcient for different flap position can be found, see equation

For the analysis it is necessary to to find the equivalent total lift coefficient for the wing, see equation

$$C_{LA} = C_L \cdot \frac{\Lambda}{\Lambda + 2} \tag{2.12}$$

This compensates for a non elliptical wing shape, however for gliders the aspect ratio is large and the total lift coefficient is therefore more than 90% of the local lift coefficient, as shown in figure 2.6.

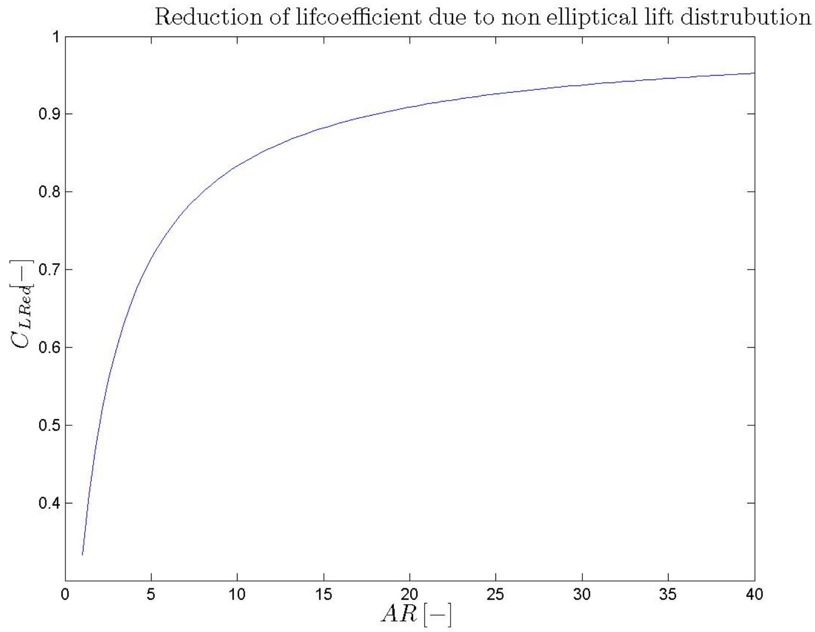

Figure 2.6.: Lift coefficient reduction.

2.4.3. Drag

The drag is consist of two parts, one for the plane and one for the line. As for the lift it changes with the velocity,angle of attack and geometry. However there is greater uncertainty with the drag estimate, and below some effects will be discussed.

2.4.3.1. Plane drag

2.4.3.2. Lift induced drag

Induced drag is drag which occurs as the result of the creation of lift in 3D. Induced drag consists of two primary components, including drag due to the creation of vortices (vortex drag) and the presence of additional viscous drag (lift-induced viscous drag).

With other parameters remaining the same, as the lift generated by the wing increases, so does the lift-induced drag. The induced drag can be estimated by equation 2.13.

$$C_{DI} = \frac{C_L^2}{\pi \cdot e \cdot \Lambda} \tag{2.13}$$

The Oswald factor e , is for a glider with high aspect ratio normally greater than 0.9, and will therefore not influence the induced drag dramatically. The induced drag is therefore important in the phases where high lift is needed, which typically will be in phase 2.

2.4.3.3. Parasitic drag

Parasite drag is drag caused by moving a solid object through the air. Parasitic drag is made up of multiple components including viscous pressure drag (form drag), and drag due to surface roughness (skin friction drag).

Parasitic drag increases with the speed because the fluid is flowing faster around protruding objects which is then increasing the friction or drag. Therefore parasitic drag will be dominant in the phases with high velocity, such as phase 3 and 4.

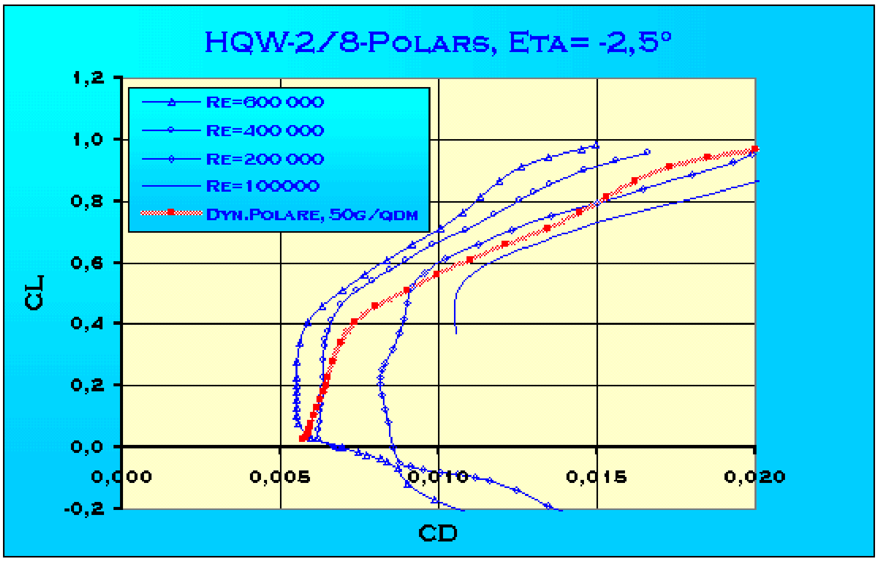

Figure 2.7.: Polars for HQW2/8 with speed flap setting .

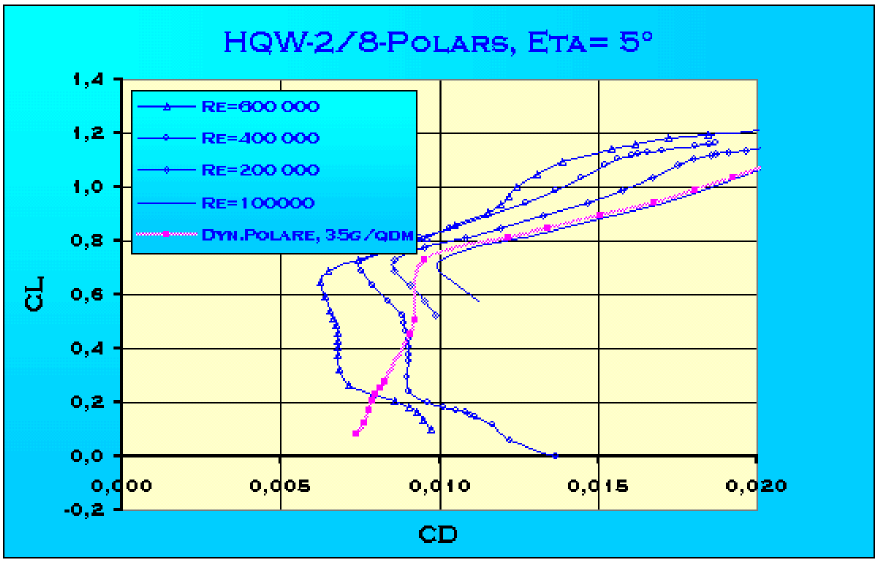

Figure 2.8.: Polars for HQW2/8 with start flap setting .

2.4.3.4. Interference drag

Additionally, the presence of multiple bodies in relative proximity it incur so called interference drag. The interference drag is hard to estimate and should ideally be measured. As a rule of thumb there is one glide ratio point lost for every interconnection, for a modern F3B plane this means 4. The interference drag is small compared to induced and parasitic drag, but it is present at every phase.

2.4.3.5. Rolling drag

When the glider is at the ground there will also be friction drag between the wheel and the ground. This drag is reduced with increasing lift since then the normal force is reduced, see equation 2.14.

$$C_{DR} = \mu \cdot (G - L) \tag{2.14}$$

2.4.3.6. Total drag

The total drag is the sum of all the drags, this will be high for a low speed then decrease and then increase as the speed goes higher.

$$C_D = C_{Ds} + C_{Di} + C_{Dp} + C_{Dr} \tag{2.15}$$

2.4.4. Line force

The line force consist of two parts, the force from the winch motor characteristic and the line characteristic.

2.4.4.1. Winch force

The force from the winch is given by its moment and drum radius, this means that the motor characteristic and the drum geometry are input parameters to the launch. In F3B the maximum allowed power is limited as well as the motor needs to be a standard car starter engine. In equation is the line force given as a function of the motor moment and the drum radius.

However the radius will change if the drum is not wide enough to hold the complete line in one single layer, which is normally the case, this means that the drum diameter will increase during the launch. This is illustrated in figure

$$F_W = \frac{M_W}{r_w} \tag{2.16}$$

The power that the winch has is given in equation , this is important since it in F3B is set a maximum allowed power.

$$P_W = F_W \cdot v_L \tag{2.17}$$

Equation can be rearranged so that the winch force can be found for any given winch output power,

$$F_W = \frac{60 \cdot P_W}{2 \cdot \pi \cdot r_{drum} \cdot N} \tag{2.18}$$

There are two events of a standard motor which needs to be taken into account, what that happens when the motor is stalled, the line force has become so large that the available torque is not sufficient to still wind in the line T_{WS}, and what happens when there is no line force, speed where the winch has no torque N_{TZ}. These two case can been seen in figure 2.9.

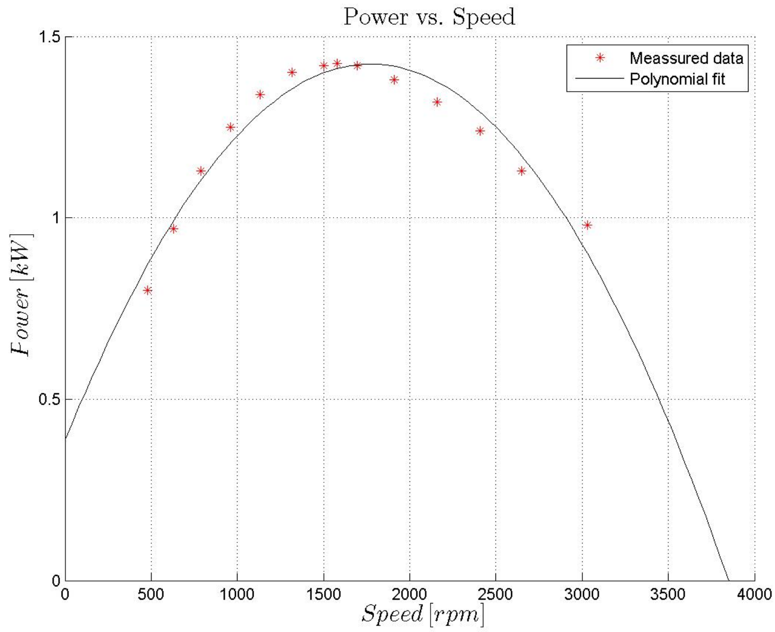

Figure 2.9.: Winch characteristic for an FRB winch.

2.4.4.2. Elastic line force

Every line is elastic, even a steel wire, however the elongation of the line and reversible of the line is very dependent of the material used.

The stress in the cable is a function of the line force and the cross section area of the line, equation

$$\sigma = \frac{F}{A} \tag{2.19}$$

The line has a circular cross section, even though a symmetrical airfoil would be ideal, this has to do with manufacturing of the line and price.

$$A = \frac{\pi \cdot d^2}{4} \tag{2.20}$$

The stress can also be expressed from the young's modulus as shown in equation

$$\sigma = E \cdot \epsilon \tag{2.21}$$

the elongation of the line is dependent of the total length of the line and the deflection as shown in equation

$$\epsilon = \frac{\Delta l}{l} \tag{2.22}$$

If equations 2.19-2.22 are combined the elastic force can be found, as shown in equation

$$F_E = \frac{E \cdot \Delta l \cdot \pi \cdot d^2}{4 \cdot l} \tag{2.23}$$

This equation can be reshaped so that the standard equation for a spring can be used, see equation

$$F_E = k \cdot \epsilon \tag{2.24}$$

The spring constant is then found by combining equation 2.23and 2.24, equation

$$k = \frac{E \cdot \pi \cdot d^2}{4} \tag{2.25}$$

The problem is that the total line is winded up and therefore the total unstressed line length length needs to be adjusted. The line length that is winded up is given by the motor and the drum diameter as shown in equation 2.26.

$$\Delta l_s = \int \frac{2 \cdot N \cdot \pi \cdot r_{drum}}{60} dt \tag{2.26}$$

The total elastic line force is then given by combing equation 2.23-2.26, which is shown in equation

$$F_E = \frac{E \cdot \Delta l \cdot \pi \cdot d^2}{4 \cdot \left(l_0 - \int \frac{2 \cdot N \cdot \pi \cdot r_{drum}}{60} dt \right)} \tag{2.27}$$

2.4.4.3. Total line force

If the line is perfectly elastic and reversible the total line force can be found by adding them, the total force can then be found by using equation 2.28.

$$F_L = \frac{E \cdot \Delta l \cdot \pi \cdot d^2}{4 \cdot \left(l_0 - \int \frac{2 \cdot N \cdot \pi \cdot r_{drum}}{60} dt \right)} + \frac{60 \cdot P_W}{2 \cdot \pi \cdot r_{drum} \cdot N} \tag{2.28}$$

The formula in equation 2.28 is implicit due to the fact that the speed and the radius of the drum is related to the other parameters in the formula, this adds complexity to the system and only a numerical solution can be found.

$$F_L = max \left(0, F_L \left(i - 1 \right) + \frac{\Delta l \cdot k}{l} \right) \tag{2.29}$$

2.4.5. Sum of forces

The total forces can be found summing up all the force in XZ direction, for X direction is given in equation

$$
\begin{aligned}
F_X &= \cos(\theta) \cdot F_L - \cos(\varphi) \cdot D - \sin(\varphi) \cdot L \\
F_Z &= -\sin(\theta) \cdot F_L - \sin(\varphi) \cdot D + \cos(\varphi) \cdot L - G
\end{aligned}
$$

When all the forces are set into these two equations becomes dependent of several parameters.

2.5. Angles

The angles is used to determine the forces in X and Z direction as well as events which will be described later. It is important to pay attention that the angles are given in the correct quadrant since some of the forces changes sign dependent on which quadrant the plane is in.

The angle between the path tangent and the horizontal plane is called φ and can be found in two different ways, either by using the velocity, equation or by using the relation to the point to the pulley, equation

$$
\begin{aligned}
\varphi &= \arctan\left(\frac{w}{u}\right) \\
\varphi &= \beta - \gamma
\end{aligned}
$$

If there is no drag or the line force is large the angle that the line is pulling is

$$
\begin{aligned}
\theta &= \gamma \, for \, L_D = 0 \\
\theta &< \gamma \, for \, L_D > 0
\end{aligned}
$$

2.6. Events

Under a normal winch start there is several events happening at certain angles, velocities and heights.

2.6.1. Pre tension

Before release from the launcher the winch is started and at a given line force the model is released with a start angle and velocity. This means that the line is winded up on the drum with increasing force and different line diameter.

2.6.2. Motor stall

If the line force is becoming larger then the maximum moment that the winch can handle the one way bearing on the winch kick in preventing the winch drum from spinning in the wrong direction.

2.6.3. Zooming flap

The launch is started with a high flap setting, then at a certain point, a γ_{SF} angle a switch is activated which then decreases the positive flap to negative flap with a time constant,

$$\begin{aligned} \kappa &= \kappa_{PF} - \kappa_{kp} \cdot t \, for \, t \leq t_\kappa \\ \kappa &= \kappa_{NF} \, for \, t > t_\kappa \end{aligned}$$

2.6.4. Zooming start

When the plane is getting close to $\gamma = 90^o$ of phase 1 the dive to zooming phase is initiated, γ_z and then the elastic energy in the line is transferred into kinetic energy.

2.6.5. Zooming arc

The radius of the arc that the zoom is initiated with is important for the final height.

2.6.6. Kick flap

At a certain height the dive is ended by applying elevator, however this elevator is connected to the flap with , kick flap, equation

$$\kappa = \kappa_{kpe} \cdot \eta$$

2.6.7. Bottom arc

Shortly before the line is released an arc is introduced together with the kick-flap mentioned above. The question is how large the radius of this arc should be to reach maximum final height. A to narrow radius will lead to high g-forces which may destroy the plane and a to large radius will lead to large energy losses.

2.6.8. Line release

At a certain β_L the line is released and the drag and gravity force is the only force working on the plane (lift is close to zero in climb phase). The angle can be defined in two ways, either by when the plane has reached a certain dive height or when the rest elongation of the line has reached a desired minimum level.

2.6.9. End of launch

The last event is when the vertical climb is ended and the winch start is considered over. The ideal path flown at the end of the launch is dependent of the task that are later flown, if there i long distance flight then it is probably most desirable to end up with a flap setting for the highest glide ratio. On the other hand if there is there is an endurance task to be flown, then a flap setting for minimum sink is desirable. Often also a change of direction is desirable, like flying back to start, then a half turn will be flown at the end of the launch. As a summary there can be said that the ideally path is dependent of task and the conditions, however for this thesis only the final height will be considered, and the flight path will therefore be simplified. In the climb phase the plane is typically flown with a negative flap to minimize drag, however this also yields that a lower C_{Lmax} so that the stall speed increases, see formula

$$v_{min} = \sqrt{\frac{2 \cdot m \cdot g}{C_L \cdot \rho \cdot S_P}} \tag{2.30}$$

How the stall speed varies for different lift coefficient can be shown in figure 2.10.

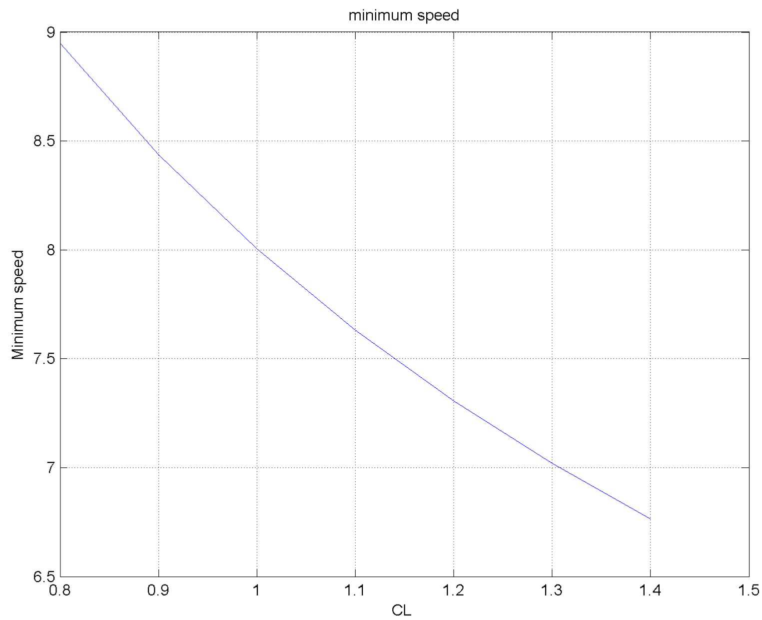

Figure 2.10.: End speed for different lift coefficients.

As seen the end velocity can typically be reduced with 2m/s by changing the flap setting at the end of the launch phase, so will this extra pilot load have a large influence of the total height. So the question is how many meters difference there will be in a small change of the end velocity.

2.6.10. Line failure

If the line force is becoming to great, the line will simply break. Before this happens it will go into a plastic deformation and the formulas for the line force are not valid anymore. The breaking point is given by equation

2.6.11. Over loading

In worst case if the G-forces is becoming to large the wing will break. Therefore it is important to keep the loading below the allowed maximums.

3. Results

3.1. Introduction

The equations in the discussion chapter has been implemented in a python simulation tool, this is due to the fact that python is an open source platform with easy and equivalent plotting syntax as matlab. The simulations software has several input and output parameters and the effects of changing these will be discussed here.

3.2. Input parameters

Several of the formulas share the same parameters and since it is so many parameters it was decided to group them into four different groups:

- plane parameters
- flight parameters
- winch parameters
- line parameters
- flight conditions parameters

The groups of parameters with allowed and standard values were derived in corporation with some of the best F3B and F3J pilots in the world, Jo Grini and Roman Vojtech, based on flight polars and values used by Helmut Quabeck.

3.2.1. Plane parameters

The values for lift and drag coefficient where chosen by using Javafoil simulation software made by Martin Hepperle and cross checking these against polars given by Helmut Quabeck. The values used in the simulatioan can be found in table 3.1 together with the rest of the plane parameters. However there are large uncertainty in these data, especially for the drag coefficient. All the coefficient are linearized for the whole Reynolds number and angle of attack range, which makes it even more inaccurate. A table look up method would most likely be more accurate, however this computational time consuming and for an parameter influence study not absolute necessary.

Description	Value
speedFlapCl0	0.111
startFlapCl0	0.9
wingSpan	3
cdParasiticStartFlap	0.035
wingLoading	3.5
cdInducedFactor	0.9
refRe	150000
ReCoeff	0.5
clAlphaCoeff	0.9
cdInference	0.01
aspectRatio	15
maxLoadFactor	50
cdParasiticSpeedFlap	0.025

Table 3.1.: Plane parameters.

3.2.2. Flight parameters

Unlike the plane parameters, the flight parameters can easily be changed by the pilot. This means that some of these parameters can easily be changed to test the effect of them by the pilot. The only flight parameters that is known from test result is the start and speed flap setting as well as the pretension in the line. This makes it hard to truly evaluate the flight parameters to real test results.

Description	Value
setpointAOA	8
takeOffSpeed	10
startFlapPos	10
preTensionOfLine	150
diveStartAngle	75
gammaR2	550
gammaR3	550
gammaR0	200
gammaR1	200
climbAngle	80
launchAngle	70
speedFlapPos	0
thermicFlapPos	5

Table 3.2.: Flight parameters.

3.2.3. Line parameters

Description	Value
totalLineLength	400
parachuteDragCoeffcient	3
lineDragCoeffsient	0.69
parachuteArea	0.2
lineDiameter	0.0014
EModule	2000000000.0

Table 3.3.: Line parameters.

3.2.4. Winch parameters

The winch parameters are based upon the result and a diagram given by Helmut Quabeck,

Description	Value
distanceToPulley	200
winchZeroSpeed	3800
drumDiameter	0.05
drumLength	0.3
winchStallTorque	9.8

Table 3.4.: Winch parameters.

3.2.5. Flight conditions parameters

The winch parameters are based upon the result and a diagram given by Helmut Quabeck,

Description	Value
windSpeed	0
humidity	0
thermicCeil	600
temperatureAtGround	25
thermic	0
pressureAtGround	101325

Table 3.5.: Winch parameters.

3.3. Output parameters

The simulation software have several output parameters and dependent of the input values varying output wil be created, table shows which parameters that the simulation can produce.

3.4. Different launch methods

As it can been seen by the output from the simulation in figure 3.1, there is quite a bit advantage of using an elastic line and a zooming procedure for an F3X glider as it was excepted.

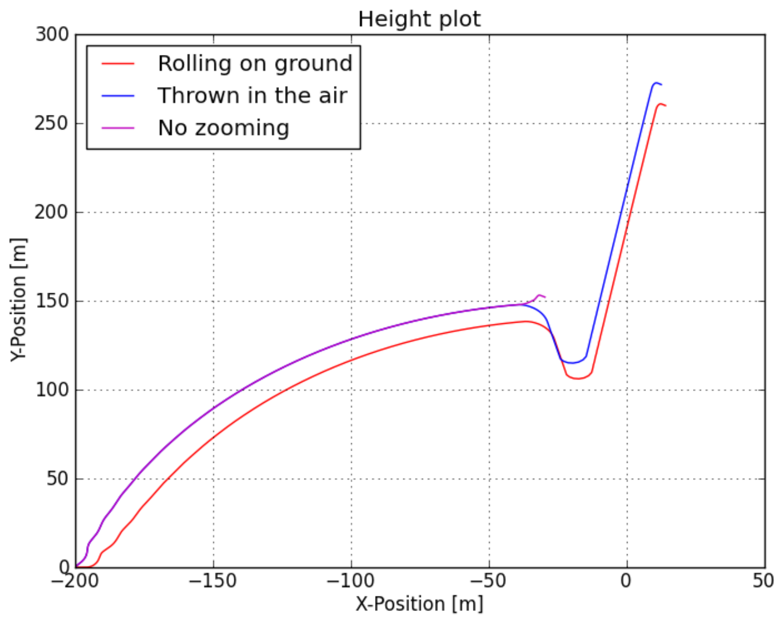

Figure 3.1.: XY-plot for different launch techniques.

The advantage of using a zooming technique can clearly been seen by looking at the energy plot for the plane, see figure 3.2.

Description	Value
cdTotal	[0.0]
totalLineLength	[400.0]
drumDiameter	[0.05]
kLine	[3078.7608005179973]
energy	[0.0]
ay	[0.0]
ax	[0.0]
deltaLineLength	[0.0]
attAng	[0.0]
psi	[0.0]
fx	[0.0]
fy	[0.0]
clTotal	[0.0]
time	[0.0]
lineDiameter	[0.0014]
velAng	[0.0]
lineForce	[0.0]
rho	[1.1838921585324385]
x	[-200.0]
lineOnWinch	[0.0]
fDrag	[0.0]
fLift	[0.0]
momentOnWinchDrum	[0]
u	[0.0]
flapPos	[0.0]
v	[0.0]
y	[0.0]
velocity	[0.0]
gamma	[0.0]

Table 3.6.: Simulations parameters

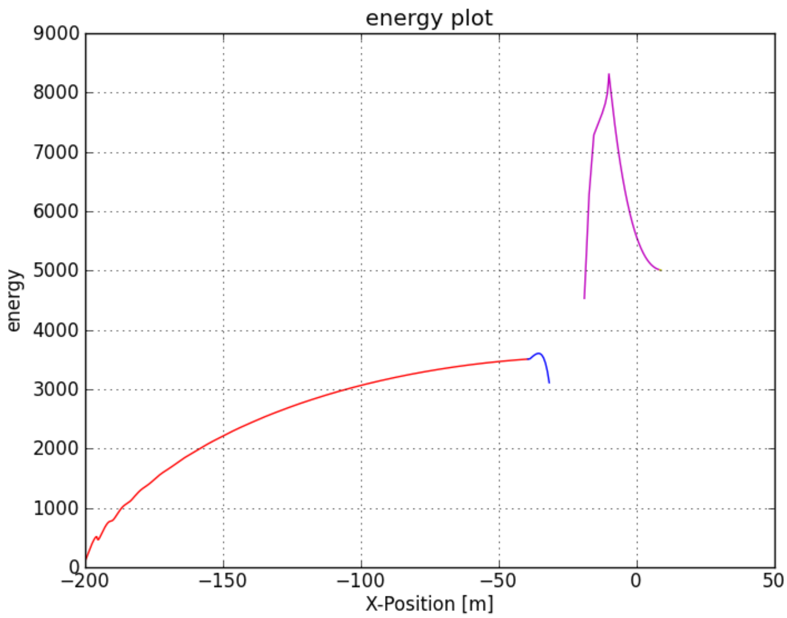

Figure 3.2.: XE-plot for a launch with standard settings.

3.4.1. Sensitivity analysis

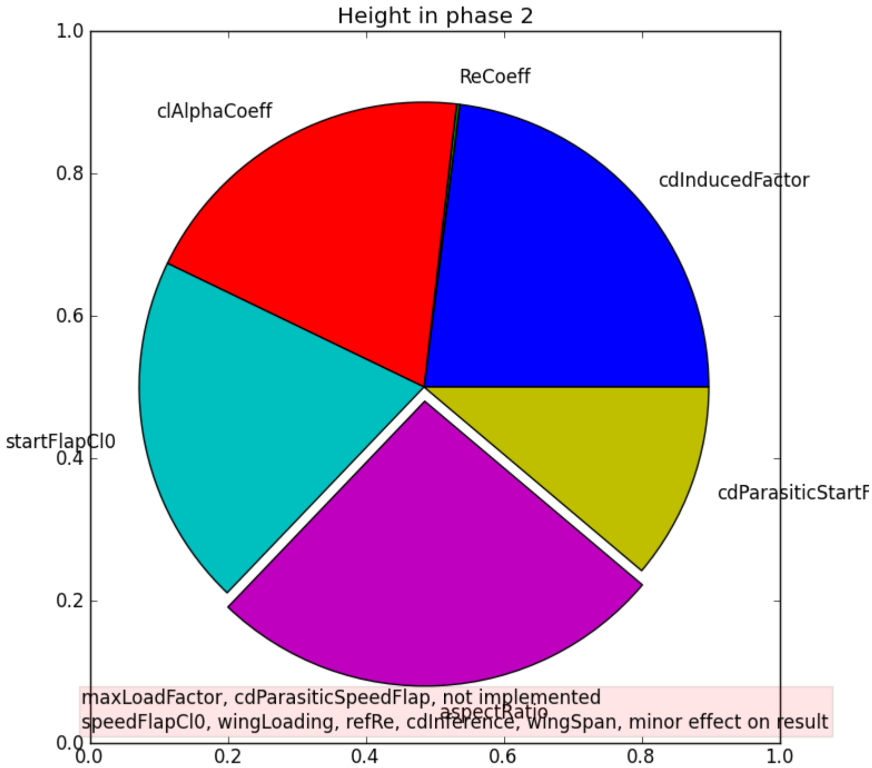

Figure 3.3.: XE-plot for a launch with standard settings.

3.4.2. Energy conservation

Since there is little practical test data available it is a good check to see if the energy balances matches up. The energy in the plane consist of potential and kinetic energy, into the system is the energy by the winch, the difference between these energy is the energy stored in the line and the energy losses due to drag forces.

The forces is dependent of where on the path of the launch the glider is, in phase 1 the glider is climbing with a high lift configuration until the height start to flatten out, then the energy in the line is transferred into kinetic energy in phase two before this is transferred into potential energy in phase 3.

The energy that the plane has in any given point is given in equation 3.1. The energy going into the system comes from the battery which is operating the winch and is given in equation

$$E = m \cdot g \cdot h + 1/2m \cdot v^2 \tag{3.1}$$

$$E = M \cdot \omega + \frac{k_{line} \cdot x^2}{2} \tag{3.2}$$

3.5. Practical test

To evaluate the result of the simulation some test results have been provided by Jo Grini and Roman Vojtech. However these flights were only logged with 3 parameters, speed,height and line force.

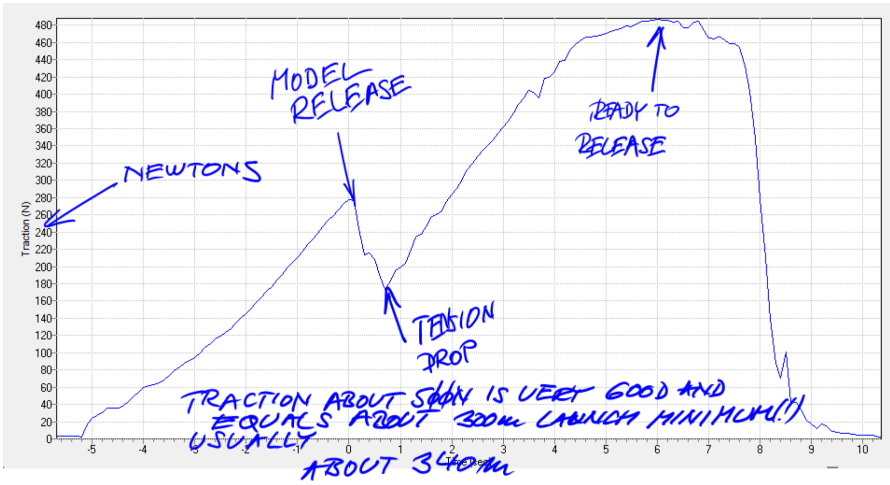

Figure 3.4.: End speed for different lift coefficients.

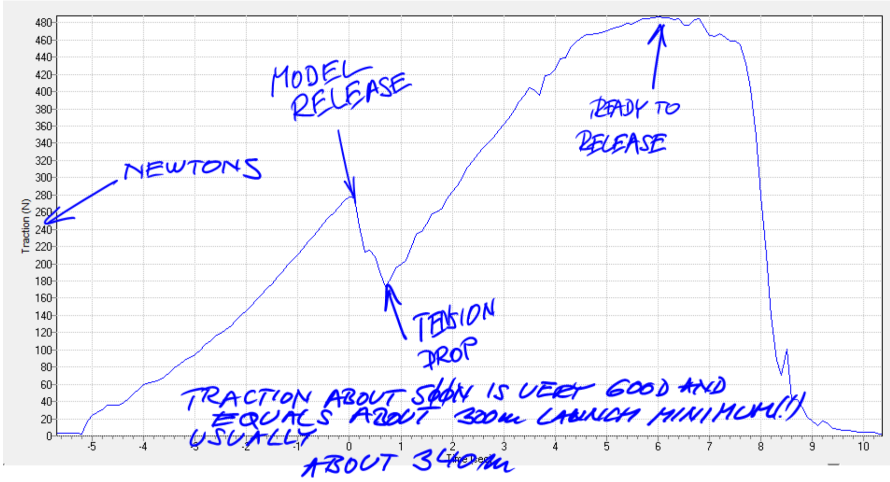

Figure 3.5.: End speed for different lift coefficients.

4. Conclusion

The glider should during the launch have the highest possible lif coeffcient, because this parameter is signficantly for the launch height h_2, this however means that the pilot needs to be able to use the flpas correctly.

The drum radius should ideal decrease during the launch, therfor it is important to have wide drum so that the drum radius only slightly increases during the launch. The drumradius should ideally be adjusted to the wind conditions aswell as the glider.

It is benefitical to use a thin and highly elastic line.

In the acceleration phase the drag should be kept at a minimum, which is achieved by a highly steep dive and correct flap position.

It is important that the transition from the accelaeration phase to the climbing phase does not follow a to small radius, since then to high drag occurs.

With increasing headwind it is possible that the force on the glider is becoming so large that the winch motor is stalling, this means that the lift needs to be reduced in order to move agaisnt the wind. In such a case a tremendous amount of energy can be stored in an elastic line, which will result in very large final height.

Termic has only a small influence on the final heigth, but is important in case of a line break.

Other sizes such as wing loading and wing area only has small impact on the final height.

5. Future work

1. Find the bug in climbangle
2. Use values from xfoil
3. Fix this documentation
4. Add references to documentation
5. Import log data
6. Export pictures
7. Make a logging pulley

Part I.

Code

```
ï»¿Imports System.Threading

Module Simulate
    Public online As Boolean

    Public debug As Boolean
    Private debugger As Debugger
    Public diveAngle As Double
    Public dt As Object



    Public Sub simulate(_debug As Boolean, _debugger As
        Debugger)

        Dim CounterKim As Integer
        Dim bugFix As Boolean
        bugFix = False
        CounterKim = 0
        debug = _debug
        debugger = _debugger
        online = True
        lengthSoFar = 0
        'alt = 2
        '""""
        'Runs the simulation
        '""""
        Dim iterations As Integer = 0

        dt = val("dt")
        While last("time") <= val("maxT") And last("y") >=
            -10.0 And last("phase") < 5 And last("y") <= 500.0


            '""""
```

```
        'Each last("phase") of the launch determines how the plane should behave
        '    0 Preload the wire. The plane is stationary and the winch starts to tention the wire
        '    1 Takeoff. The plane is released and starts to accelerate along the ground. This last("phase") starts when the line reaches the wanted preforce
        '    2 Liftoff and climb. The plane increases the angle of attack and leaves the ground. This last("phase") starts when the takeoffspeed (v0) is reached
        '    3 Dive. At the peak height the plane starts to dive against the pulley to increase its energy
        '    4 Climb. The winch is released and the plane starts to climb.
        '    5 Climb is ended and plane is flying straight ahead with thermic setting.

        '"""

        ' Change phases


        ' Check if Pre tensioning of line is completed
        If last("lineForce") > val("preTensionOfLine") And last("phase") = 0 Then
            'dt = val("dt") / 100
            If alt >= 2 Then ' Alt 2 does not contain any takeoff along the ground
                last("phase", 2)
                last("u", Math.Cos(val("launchAngle")) * val("takeOffSpeed")) ' Plane velocity in x direction [m/s]
                last("v", Math.Sin(val("launchAngle")) *
```

```
                    val("takeOffSpeed"))    ' Plane
                    velocity in y direction [m/s]
                last("gamma", val("launchAngle"))   '
                    Angle between plane and ground [deg]

            Else
                last("phase", 1)
            End If
        End If


        ' Check if we reached take of speed
        If last("airSpeed") > val("takeOffSpeed") And
           last("phase") = 1 Then
            last("phase", 2)
        End If


        'CHeck if we reached angle to start zooming
        If last("psi") > val("diveStartAngle") And last("
           phase") = 2 Then
            last("phase", 3)
            diveAngle = -last("psi")

            'dt = val("dt")
        End If
        'val("dt") = val("dt")/5


        ' Reached bootom of zoom
        If (last("y") < 100 Or last("lineForce") < 0) And
            last("phase") = 3 Then
            add("phase", 4)
            online = False

            last("u", Math.Cos(val("climbAngle")) * last(
```

```
"airSpeed")) ' Plane velocity in x direction [m/s]
            last("v", Math.Sin(val("climbAngle")) * last("airSpeed"))   ' Plane velocity in y direction [m/s]
            bugFix = False
        End If


        'Reached top of launch
        '
        If (last("airSpeed") < velocityMin(last("flapPos")) * 1.1 Or ((last("y") - last2("y")) <= 0 And bugFix)) And last("phase") >= 4 And last("phase") < 5 Then
            last("phase", 5)
        End If

        If (last("gamma") <= 0 And last("phase") >= 5) Then
            last("phase", 6)
        End If


        euler() ' call the desired solver
        bugFix = True

        'Debugging mode
        If (debug) Then
            debugger.iterationMade()
            If debugger.isPause Then
                debugger.paused(outputs.getAllNames, inputs.getAll)
            End If
            While (debugger.isPause)
                Thread.Sleep(500)
```

```
                End While


            End If

            iterations = iterations + 1
            If iterations Mod 1000 = 0 Then
                Console.WriteLine("iterations " & iterations)
            End If

        End While

        If (debug) Then
            debugger.done()
        End If



    End Sub


    'E.append(_y[-1]*g()*planemass+0.5*pm*_velocity[-1]^2) add some energy calculations aswell??Kinetic potensinal, winch and whats stored in line..



End Module
```

Listing 1: Simulate

```
ï»¿Module Euler
    '"""
    'Calculates the new position of the plane using forward Euler iteration
    'Does not allow the plane to go below y=0 as this is
```

```
        ground level
    '"""
    Function Euler()

        If phase = 0 Then
            sumForces()
            _u.Add(0)
            _v.Add(0)
            _ax.Add(0)
            _ay.Add(0)
        ElseIf phase = 1 And alt = 1 Then
            sumForces()
            _ax.Add(_fx(_fx.Count - 1) / pm)
            _ay.Add(_fy(_fy.Count - 1) / pm)
            _u.Add(_u(_u.Count - 1) + _ax(_ax.Count - 1) * dt)
            _v.Add(0)

        Else
            sumForces()
            _ax.Add(_fx(_fx.Count - 1) / pm)
            _ay.Add(_fy(_fy.Count - 1) / pm)
            _u.Add(_u(_u.Count - 1) + _ax(_ax.Count - 1) * dt)
            _v.Add(_v(_v.Count - 1) + _ay(_ay.Count - 1) * dt)
        End If

        _x.Add(_x(_x.Count - 1) + _u(_u.Count - 1) * dt)
        _y.add(Math.Max(_y(_y.Count - 1) + _v(_v.Count - 1) * dt),0)) ' To make sure y always is positive

    End Function
End Module
```

Listing 2: Euler

```
ï»¿
Module Angles

    Public integral As Double
    Dim tempKim As Object

    Public Function calcVelAng()
        '"""
        'Returns the angle of the velocity vector of the
           plane
        '"""
        If last("phase") = 2 Then
            calcVelAng = Math.Atan2(last("v"), last("u"))
        ElseIf last("phase") = 3 Then
            calcVelAng = diveAngle
        ElseIf last("phase") = 4 Then
            calcVelAng = val("climbAngle")
        Else
            calcVelAng = 0
        End If

    End Function

    Public Function calcLoadFactor()
        Dim r, zentAcc, loadVector As Double
        loadVector = 50
        r = Math.Sqrt(last("x") ^ 2 + last("y") ^ 2)
        zentAcc = last("velocity") ^ 2 / r ' Should this be
           ground velocity?
        calcLoadFactor = (last("Flift") + loadVector *
           zentAcc) / g() * planeMass

    End Function

    Public Function calcAttAng()
```

```
        '"""
        'Returns the angle of attack for the plane
        '"""

        If last("phase") = 2 Then
            calcAttAng = rad(8)
        Else
            calcAttAng = 0
        End If
        'If last("u") = 0 And last("v") = 0 Then ' If the plane is stationary the attack angle should be zero
        '    calcAttAng = 0
        'Else
        '    calcAttAng = last("gamma") - last("velAng")
        'End If
    End Function

    Public Function calcGamma()
        '"""
        'Returns the plane angle.
        'Assumes the plane flies with gamma0 degrees towards the line all the time
        '"""

        calcGamma = last("psi") +last("velAng")+last("attAng")
        'Dim gammaMyR, goal As Double

        'tempKim = deg(last("gamma"))


        'If last("phase") = 2 Then
        '    gammaMyR = val("gammaR0") ' radius of bottom of launch
        'ElseIf last("phase") = 3 Then
```

```
        '    gammaMyR = val("gammaR1") ' radius of top of zoom
        'ElseIf (last("phase") = 4) Then
        '    gammaMyR = val("gammaR2") ' radius of bottom of zoom
        'Else
        '    gammaMyR = val("gammaR3") ' radius of flatten
        'End If


        'goal = 0

        'If last("phase") = 0 Then
        '    If alt >= 2 Then
        '        goal = val("gamma0") + val("setpointAOA") ' Included to simplify things for the governor
        '    Else
        '        goal = 0 'gamma0
        '    End If
        'End If

        '' rolling on ground
        'If last("phase") = 1 Then
        '    goal = 0 'gamma0
        'End If

        ''
        'If last("phase") = 2 Then

        '    goal = gammaDesired() - last("psi")


        'End If

        'If last("phase") = 3 Then
        '    'If alt = 3 Then
```

```
        '     'goal = val("climbAngle")
        '     'Else
        '     goal = -last("psi")
        '     'End If
        'End If

        'If last("phase") = 4 Then
        '     goal = val("climbAngle")
        'End If

        'If last("phase") = 5 Then
        '     goal = 0
        'End If

        'If goal < last("gamma") Then

        '     calcGamma = Math.Max(goal, last("gamma") -
            gammaMyR * val("dt"))
        'ElseIf (goal > last("gamma")) Then

        '     calcGamma = Math.Min(goal, last("gamma") +
            gammaMyR * val("dt"))

        'Else
        '     calcGamma = last("gamma")
        'End If

    End Function

    Public Function calcPsi()
        '"""
        'Returns the line angle
        '"""
        calcPsi = Math.Atan2(last("y"), (-last("x")))
    End Function

```

```
    Public Function gammaDesired()
        Dim errorAng, derivative, gammaDesiredAngle, previousErrorAng As Double
        Dim Kp, Ki, Kd As Double

        Kp = 1.1
        Ki = 0
        Kd = 0

        tempKim = last("gamma")
        errorAng = val("setpointAOA") - calcAttAng()

        tempKim = deg(errorAng)
        integral = integral + (deg(errorAng) * val("dt"))
        derivative = deg(errorAng - previousErrorAng) / val("dt")

        gammaDesiredAngle = gammaDesiredAngle + rad(Kp * deg(errorAng) + Ki * integral + Kd * derivative)

        previousErrorAng = errorAng
        If gammaDesiredAngle > rad(95) Then
            gammaDesiredAngle = rad(95)
        End If
        If gammaDesiredAngle < rad(50) Then
            gammaDesiredAngle = rad(50)
        End If


        gammaDesired = gammaDesiredAngle


    End Function

End Module
```

Listing 3: Angles

```
ï»¿Module Forces

    Dim s As Double
    Dim deltaLineForceMy As Double
    Dim gravityForce As Object

    Private Property tempKim As Object

    Sub sumForces()
        '"""
        'Calculates the resulting forces acting on the plane
        '"""


        add("horisontalWindSpeed", horisontalSpeed(val("windSpeed"), last("y")))
        add("verticalWindSpeed", verticalSpeed(val("thermic"), val("thermicCeil"), last("y")))

        add("airSpeedX", last("u") + last("horisontalWindSpeed"))
        add("airSpeedY", last("v") + last("verticalWindSpeed"))

        add("groundSpeed", Math.Sqrt(last("u") ^ 2 + last("v") ^ 2))
        add("airSpeed", Math.Sqrt(last("airSpeedX") ^ 2 + last("airSpeedY") ^ 2))



        If last("phase") > 1 Then
            add("psi", calcPsi())
```

```
        add("velAng", calcVelAng())
        add("attAng", calcAttAng())
        add("gamma", calcGamma())

        ' on ground all angles can be set to 0
    Else
        add("psi", 0)
        add("velAng", 0)
        add("attAng", 0)
        add("gamma", 0)

    End If


    add("rho", densityWithHumidity(val("humidity"), val("pressureAtGround"), val("tempAtGround"), last("y")))
    add("temperature", temperature(last("y"), val("tempAtGround")))
    add("nu", dynamicViscosity(last("temperature")))
    add("Re", reynoldsNumber(last("nu"), last("airSpeed"), meanChoord))


    add("clTotal", calcCl(last("attAng"), val("clAlphaCoeff"), val("speedFlapPos"), val("startFlapPos"), val("speedFlapCl0"), val("startFlapCl0"), last("flapPos")))
    add("cdTotal", calcCd(val("AR"), val("cdInducedFactor"), last("clTotal"), val("cdParasiticSpeedFlap"), val("cdParasiticStartFlap"), val("speedFlapPos"), val("startFlapPos"), last("flapPos"), last("Re"), val("refRe"), val("reCoeff"), val("cdInference")))
    add("cdLine", cdLine())

    add("lengthToPlaneFromPulley",
```

```
            lengthToPlaneFromPulley(last("x"), last("y")))
        If online Then
            add("totalLineLength", lineLength(last("x"), last
                ("y"), val("distanceToPulley")))
            'add("drumDiameter", drumDiameter(last("
                drumDiameter")))
            add("drumDiameter", drumDiameter2(last("
                lineOnWinch"), last("drumDiameter"), val("
                drumLength"), last("lineDiameter")))
            add("momentOnWinchDrum", momentOnWinchDrum(last("
                lineForce"), last("drumDiameter")))
            add("s", Swinch(last("lineForce"), last("
                drumDiameter"), val("wst"), val("wzs"), last("
                momentOnWinchDrum"), val("dt")))
            add("lineOnWinch", last("lineOnWinch") + last("s"
                ))
            add("deltaLineLength", last("totalLineLength") -
                last2("totalLineLength") + last("s"))
            'add("lineDiameter", lineDiameter(last("phase"),
                val("lineDiameter"), last("totalLineLength"),
                last("deltaLineLength")))
            add("lineDiameter", lineDiameter2(last("phase"),
                val("lineDiameter"), val("poissonRatio"), val(
                "EModule"), last("lineForce")))

            add("kLine", kLine(val("EModule"), last("
                lineDiameter")))
            deltaLineForceMy = deltaLineForce(last("
                totalLineLength"), last("deltaLineLength"),
                last("kLine"))
            add("lineForce", lineForce(last("phase"), last("
                lineForce"), deltaLineForceMy))

            add("lineArea", lineArea(last("lineDiameter"),
                last("lengthToPlaneFromPulley"), last("gamma")
                ))
```

```
        Else
            add("totalLineLength", last("totalLineLength"))
            add("momentOnWinchDrum", 0.0)
            add("lineOnWinch", last("lineOnWinch"))
            add("deltaLineLength", 0.0)
            add("lineDiameter", val("lineDiameter"))
            add("kLine", last("kLine"))
            add("lineForce", 0)
            add("drumDiameter", last("drumDiameter"))

            add("lineArea", last("lineArea"))
        End If


        If last("phase") >= 1 Then
            gravityForce = g() * planeMass
            add("fDrag", Fdrag(last("cdTotal"), last("
               airSpeed"), last("rho"), wingArea))
            add("fLift", Flift(last("clTotal"), last("
               airSpeed"), last("rho"), wingArea))
            add("lineDrag", FlineDrag(val("
               lineDragCoefficient"), last("airSpeed"), last(
               "rho"), last("lineArea"), last("phase")))


        Else
            add("fDrag", 0)
            add("fLift", 0)
            add("lineDrag", 0)
            gravityForce = 0
        End If

        add("fTotalDrag", last("fDrag") + last("lineDrag"))
        add("fx", -last("fTotalDrag") * Math.Cos(last("velAng
           ")) + last("lineForce") * Math.Cos(last("psi")) -
           last("fLift") * Math.Sin(last("velAng")))
```

```
        add("fy", -last("fTotalDrag") * Math.Sin(last("velAng")) - last("lineForce") * Math.Sin(last("psi")) + last("fLift") * Math.Cos(last("velAng")) - gravityForce)

    End Sub


End Module
```

Listing 4: Forces

```
ï»¿Module LnD


    Public Function Flift(ByVal cl As Double, ByVal velocity As Double, ByVal rho As Double, ByVal wingArea As Double)
        '"""
        'Returns the lift force of the plane based on the input velocity
        '    Flift = cl * v ^ 2 * rho * wingArea / 2
        '"""
        If last("phase") <= 2 Then
            Flift = cl * velocity * Math.Abs(velocity) * rho * wingArea / 2
        Else
            Flift = 0
        End If
    End Function


    Public Function Fdrag(ByVal cd As Double, ByVal velocity As Double, ByVal rho As Double, ByVal wingArea As Double)
        '"""
```

```
        'Returns the grad force of the plane based on the
            input velocity
        '    Flift = cd * v ^ 2 * rho * wingArea / 2
        '""

        Fdrag = cd * velocity * Math.Abs(velocity) * rho *
            wingArea / 2

    End Function
    Public Function lineArea(ByVal diameter As Double, ByVal
        length As Double, ByVal beta As Double)
        lineArea = diameter * length / 3 '*np.sin(rad(_beta))
            /3

    End Function
    Public Function FlineDrag(ByVal cd As Double, ByVal
        velocity As Double, ByVal rho As Double, ByVal
        lineArea As Double, ByVal phase As Double)
        '""
        'Returns the drag force of the line based on the
            input velocity
        '    Flift = cd * v ^ 2 * rho * lineArea / 2
        '""
        'No drag when flying towards pulley
        If phase >= 3 Then
            FlineDrag = 0
        Else
            FlineDrag = cd * velocity * Math.Abs(velocity) *
                rho * lineArea / 2
        End If

    End Function
    Public Function velocityMin(ByVal flapPos)
        velocityMin = 9

        ' Drag formulas
```

```
    End Function
    Public Function calcCd(ByVal AR As Double, ByVal
       cdInducedFactor As Double, ByVal cl As Double, ByVal
       cdParasiticSpeedFlap As Double, ByVal
       cdParasiticStartFlap As Double, ByVal speedFlapPos As
       Double, ByVal startFlapPos As Double, ByVal flapPos As
        Double, ByVal Re As Double, ByVal refRe As Double,
       ByVal ReCoeff As Double, ByVal cdInference As Double)
        '"""
        'Returns the total drag coefficient
        '"""
        Dim cli, cdp, cdf As Double
        cli = cdInduced(AR, cdInducedFactor, cl)
        cdp = cdParasitic(cdParasiticSpeedFlap,
           cdParasiticStartFlap, speedFlapPos, startFlapPos,
           flapPos, Re, refRe, ReCoeff)
        cdf = cdInterference(cdInference, Re, refRe, ReCoeff)

        calcCd = cli + cdp + cdf

    End Function
    Public Function cdParasitic(ByVal cdParasiticSpeedFlap As
        Double, ByVal cdParasiticStartFlap As Double, ByVal
       speedFlapPos As Double, ByVal startFlapPos As Double,
       ByVal flapPos As Double, ByVal Re As Double, ByVal
       refRe As Double, ByVal ReCoeff As Double)
        '"""
        'Returns the parasitic drag coefficient
        '"""
        '-2.5 0.025 drag coeff speed flap
        '  10 0.04  drag coeff start flap
        Dim x1, x2, y1, y2 As Double
        y1 = cdParasiticSpeedFlap
        x1 = speedFlapPos
        y2 = cdParasiticStartFlap
        x2 = startFlapPos
```

```
        cdParasitic = (y2 - y1) / (x2 - x1) * (flapPos - x1) + y1

    End Function
    Public Function cdInterference(ByVal cdInference As Double, ByVal Re As Double, ByVal refRe As Double, ByVal ReCoeff As Double)
        '"""
        'Returns the interference drag coefficient
        '"""
        cdInterference = cdInference * (refRe / Re) ^ ReCoeff

    End Function
    Public Function cdInduced(ByVal AR As Double, ByVal cdInducedFactor As Double, ByVal cl As Double)
        '"""
        'Returns the induced drag coefficient
        '"""
        ' 0.95 < cdInducedFactor < 1
        cdInduced = cl ^ 2 / (Math.PI * AR * cdInducedFactor)

    End Function
    Public Function cdLine()
        cdLine = 0.69

        ' Lift formulas
    End Function
    Public Function calcCl(ByVal attAng As Double, ByVal clAlphaCoeff As Double, ByVal speedFlapPos As Double, ByVal startFlapPos As Double, ByVal speedFlapCl0 As Double, ByVal startFlapCl0 As Double, ByVal flapPos As Double)
        '"""
        'Returns the lift coefficient
        '"""
        calcCl = clAlpha(clAlphaCoeff) * attAng + calcCl0(
```

```
            speedFlapPos, startFlapPos, speedFlapCl0,
            startFlapCl0, flapPos)

    End Function
    Public Function clAlpha(ByVal clAlphaCoeff As Double)
        Return 2 * Math.PI * clAlphaCoeff

    End Function
    Public Function calcCl0(ByVal speedFlapPos As Double,
       ByVal startFlapPos As Double, ByVal speedFlapCl0 As
       Double, ByVal startFlapCl0 As Double, ByVal flapPos As
        Double)
        '-2.5 0.111 stall occurs at 10deg
        '10 0.9 stall occurs at 5deg
        Dim x1, x2, y1, y2 As Double
        y1 = speedFlapCl0
        x1 = speedFlapPos
        y2 = startFlapCl0
        x2 = startFlapPos
        calcCl0 = (y2 - y1) / (x2 - x1) * (flapPos - x1) + y1

        ' These are just for control or future uses
    End Function
    Public Function clAlpha2()
        '0deg 0.278
        '3.5deg 0.615
        Dim x1, x2, y1, y2 As Double
        y1 = 0.278
        x1 = 0
        y2 = 0.615
        x2 = 3.3
        clAlpha2 = (y2 - y1) / (x2 - x1) * 180 / Math.PI
    End Function

    Public Function clCD()
        ''    cl[0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 ]
```

```
        ''    cdp_n5_Re150000 = [.0127 .0113 .0108 .0108
            .0107 0.0106 .0113 .0123 .0153 .0215 .0320]
        ''    cdp_n0_Re200000 = [.0113 0.0098 .0088 .0085
            .0084  .0088 .0098 .0115 .0145  .0185 .00265]
        ''    cdp_n_25_Re300000 = [.0098 0.0078 .0073 .0078
            0.0090 .0113 0.0170 0.0220]

        ' at 5deg flap cwTot~0.049 at 0 deg 0.031
        clCD = 3
    End Function




End Module
```

Listing 5: LnD

```
ï»¿Module Solver

    Dim tempKim As Double

    Sub euler()
        '""
        'Calculates the new position of the plane using
            forward Euler iteration
        'Does not allow the plane to go below y=0 as this is
            ground level
        '""
        sumForces()

        ' Pretension
        If last("phase") = 0 Then
            add("u", 0)
            add("v", 0)
            add("ax", 0)
```

```
            add("ay", 0)

            'Rolling on ground
        ElseIf last("phase") = 1 And alt = 1 Then
            add("ax", last("fx") / planeMass)
            add("ay", last("fy") / planeMass)
            add("u", last("u") + last("ax") * dt)
            add("v", 0)

            ' All other cases
        Else
            add("ax", last("fx") / planeMass)
            add("ay", last("fy") / planeMass)
            add("u", last("u") + last("ax") * dt)
            add("v", last("v") + last("ay") * dt)
        End If


        add("x", last("x") + last("u") * dt)
        add("y", Math.Max(last("y") + last("v") * dt, 0)) 'To
            make sure y always is positive
        add("time", last("time") + val("dt")) ' MOve this to
           the solver??

    End Sub
End Module
```

Listing 6: Solver

```
ï»¿Module Winch
    Dim layersOnDrum As Double
    Dim momentOnDrum As Double
    Dim _dt As Double
    Dim omega As Double
    Public lengthSoFar As Double
    'Dim drumDiameter As Double
```

```
    Sub init()
        layersOnDrum = 1
    End Sub

    'Returns the length of wire which the winch collects during 1 dt
    'If the torque is bigger than the stall torque, It is assumed that
    'the winch stops and does not reverse
    Function Swinch(ByVal _lineForce As Double, ByVal _drumDiameter As Double, ByVal _winchStallTorque As Double, ByVal _winchZeroSpeed As Double, _
                    ByVal _momentOnWinchDrum As Double, ByVal _dt As Double) As Double

        momentOnDrum = momentOnWinchDrum(_lineForce, _drumDiameter)  ' Torque acting on the cylinder [Nm]

        omega = radPerS(speedOfWinchDrum(_winchStallTorque, _winchZeroSpeed, momentOnDrum)) ' Rotational speed of the winch [rad/s]

        Swinch = omega * (_drumDiameter / 2) * _dt ' The amount of line the winch collects [m]

        'lw.append(lw[-1]+S as Double ) as double

    End Function

    Function momentOnWinchDrum(ByVal _lineForce As Double, ByVal _drumDiameter As Double) As Double
        momentOnWinchDrum = _lineForce * _drumDiameter / 2  ' Torque acting on the cylinder [Nm]
```

```
    End Function

    ' Rotational speed of the winch [rpm]
    Function speedOfWinchDrum(ByVal _winchStallTorque As
       Double, ByVal _winchZeroSpeed As Double, ByVal
       _momentOnWinchDrum As Double) As Double
        speedOfWinchDrum = rpm(Math.Min(Math.Max(0, (1 -
           _momentOnWinchDrum / _winchStallTorque)), 1) *
           _winchZeroSpeed)
    End Function

    'Returns the spring constant of the line
    'As the line is shortened will the springconstant
       increase
    'Assumes the constant is reduced inverse to the relative
       length
    ' Springcoefficient of the line [N] E*pi*d^2/4
    Function kLine(ByVal E As Double, ByVal lineDiameter As
       Double) As Double
        kLine = E * Math.PI * lineDiameter ^ 2 / 4
    End Function

    'Returns the actual length of the line
    Function lineLength(ByVal x As Double, ByVal y As Double,
        ByVal L0 As Double) As Double

        lineLength = Math.Sqrt(x ^ 2 + y ^ 2) + L0
    End Function

    Function deltaLineForce(ByVal _lineLength As Double,
       ByVal _deltaLineLength As Double, ByVal _kLine As
       Double) As Double
        deltaLineForce = _deltaLineLength / _lineLength *
           _kLine
    End Function

```

```
'Returns the force in the line
'    dF = (dL + winchspeed ) / k
Function lineForce(ByVal _phase As Double, ByVal
   _lineForceOld As Double, ByVal _deltaLineForce As
   Double) As Double

    If _phase >= 4 Then
        lineForce = 0
    Else
        lineForce = Math.Max(0, (_lineForceOld +
           _deltaLineForce))
    End If
End Function

Function deltaLineLength(ByVal lineLengthOld As Double,
   ByVal x As Double, ByVal y As Double, ByVal L0 As
   Double) As Double
    deltaLineLength = lineLength(x, y, L0) -
       lineLengthOld
End Function

Function lineDiameter(ByVal phase As Integer, ByVal D0 As
    Double, ByVal l0 As Double, ByVal deltaL As Double)
   As Double
    If phase >= 4 Then
        lineDiameter = D0
    Else
        lineDiameter = D0 'np.sqrt(l0/(l0+deltaL as
           Double ) as double as Double ) as double*D0
    End If
End Function

Function lineDiameter2(ByVal phase As Integer, ByVal D0
   As Double, ByVal poissonRatio As Double, ByVal
   youngsModulus As Double, ByVal lineForce As Double) As
    Double
```

```
        If phase >= 4 Then
            lineDiameter2 = D0
        Else
            lineDiameter2 = (D0 + Math.Sqrt(D0 ^ 2 - 16 *
               poissonRatio * lineForce / youngsModulus /
               Math.PI)) / 2
        End If
    End Function

    Function drumDiameter(ByVal _drumDiameter As Double) As
       Double
        'global drumDiameter,layersOnDrum
        Return _drumDiameter
    End Function
    Function drumDiameter2(ByVal lineOnDrum As Double, ByVal
       _drumDiameter As Double, ByVal _drumWidth As Double,
       ByVal _lineDiameter As Double) As Double
        'global drumDiameter,layersOnDrum

        If lineOnDrum > (Math.PI * _drumDiameter * _drumWidth
            / _lineDiameter + lengthSoFar) Then
            'layersOnDrum = layersOnDrum + 1


            lengthSoFar = lineOnDrum - lengthSoFar
            drumDiameter2 = _drumDiameter + _lineDiameter
        Else
            drumDiameter2 = _drumDiameter
        End If
    End Function
End Module
```

Listing 7: Winch

```
ï»¿Module Weather
    Function horisontalSpeed(ByVal _windspeed As Double,
```

```
        ByVal _y As Double) As Double
        Return _windspeed * (_y / 2) ^ (1 / 7)
    End Function

    Function verticalSpeed(ByVal _thermicSpeed As Double,
        ByVal _thermicCeil As Double, ByVal _y As Double) As
        Double
        If _y > _thermicCeil Then
            Return _thermicSpeed
        Else
            Return _y / _thermicCeil * _thermicSpeed
        End If

    End Function
End Module
```

Listing 8: Weather