



Institut für Thermische Turbomaschinen  
und Maschinendynamik



# **Simulation von Propellern für die Elektrosegelflugklasse F5B**

Christian Weiß

**Bachelor Arbeit**

Technische Universität Graz

Institut für Thermische Turbomaschinen und Maschinendynamik  
Institutsvorstand: Univ.-Prof. Dr.-Ing. Franz Heitmeir

Betreuer: Dr.-Ing. Oliver Borm

Graz, 2014-06-27



## Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz,

---

Date

Signature

## Eidesstattliche Erklärung\*

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am

---

Datum

Unterschrift

---

\* Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008



## Danksagung

Ich möchte mich bei meinem Betreuer Dr.-Ing Oliver Borm für seine umfangreiche Unterstützung bedanken.

Weiters möchte ich mich bei Dipl.-Päd. Georg Theuerkauf bedanken, der mir den 3D-Scanner der HTL-Steyr zur Verfügung stellte und mich beim Einscannen der Propeller unterstützte.

Außerdem will ich mich bei allen Anderen bedanken, die mich durch Korrekturlesen meiner Arbeit unterstützten.



## Kurzfassung

Diese Arbeit soll eine Möglichkeit zur Verbesserung und Anpassung von Modellflugzeugpropellern für Wettbewerbe bereitstellen. Dazu soll die bestehende Geometrie von zwei verwendeten Propellern digitalisiert und untersucht werden. Die Simulation soll im Programm QBlade erfolgen. Dieses muss aber für die Berechnung von Propellern adaptiert werden.

## Abstract

This Thesis should provide an opportunity to improve and adjust propellers of modelairplanes for competitions. Therefor the existing geometry of two propellers should be digitalized and investigated. The simulation should be done with the program QBlade. This has to be adapted to calculate propellers.



# Inhaltsverzeichnis

|  |             |
|--|-------------|
| <b>Abbildungsverzeichnis</b>                   | <b>xi</b>   |
| <b>Nomenklatur</b>                             | <b>xiii</b> |
| <b>1 Einleitung</b>                            | <b>1</b>    |
| 1.1 Aufgabenstellung . . . . .                 | 1           |
| 1.2 Ausgangslage . . . . .                     | 1           |
| <b>2 Theoretische Grundlagen</b>               | <b>3</b>    |
| 2.1 Propeller . . . . .                        | 3           |
| 2.2 Rotorblatt . . . . .                       | 4           |
| 2.3 Profil . . . . .                           | 4           |
| 2.3.1 ebene Schnitte . . . . .                 | 6           |
| 2.3.2 Zylinderschnitte . . . . .               | 7           |
| 2.4 BEM Theorie . . . . .                      | 8           |
| 2.4.1 Blade Element Momentum Theorie . . . . . | 8           |
| 2.4.2 BEM für Windturbinen . . . . .           | 9           |
| 2.4.3 BEM für Propeller . . . . .              | 14          |
| 2.4.4 Korrekturfaktoren . . . . .              | 15          |
| <b>3 Praktische Anwendung</b>                  | <b>19</b>   |
| 3.1 Vorgehensweise . . . . .                   | 19          |
| 3.2 Scannen der Geometrie . . . . .            | 21          |
| 3.3 Verarbeitung der Punkte in Catia . . . . . | 23          |
| 3.4 Verarbeitung mit python . . . . .          | 24          |
| 3.5 QBlade . . . . .                           | 27          |
| 3.5.1 Allgemeines . . . . .                    | 27          |
| 3.5.2 Quellcode der BEM Simulation . . . . .   | 27          |
| 3.5.3 Anwendung . . . . .                      | 40          |
| 3.6 Bewertung mit python . . . . .             | 45          |
| 3.7 Auswertung . . . . .                       | 47          |
| <b>4 Ausblick</b>                              | <b>55</b>   |
| <b>A Anhang</b>                                | <b>57</b>   |
| A.1 BData.cpp . . . . .                        | 57          |
| A.2 catiasort.py . . . . .                     | 71          |
| <b>Literaturverzeichnis</b>                    | <b>77</b>   |



# Abbildungsverzeichnis

|      |   |    |
|------|---|----|
| 2.1  | Ausgangspropeller   | 3  |
| 2.2  | Auf den Propeller wirkende Kräfte im Betrieb                        | 4  |
| 2.3  | Charakterisierung eines Profils                                     | 5  |
| 2.4  | Am Profil angreifende Kräfte  | 5  |
| 2.5  | Schnitt des Rotorblattes mit einer Ebene                            | 6  |
| 2.6  | Schneiden mit einer Zylindermantelfläche                            | 7  |
| 2.7  | Zustandsgrößen am Beispiel einer Turbine, in Anlehnung an [6, S 30] | 8  |
| 2.8  | Geschwindigkeiten im Vergleich von Propeller und Turbine            | 12 |
| 2.9  | Glauertkorrektur als Verlauf von $C_T$ über $a$                     | 17 |
| 3.1  | Grafische Darstellung des Arbeitsablaufes                           | 20 |
| 3.2  | Die beiden von Karl Waser zur Verfügung gestellten Propeller        | 21 |
| 3.3  | mit Kreidestaub bedeckter Propeller                                 | 22 |
| 3.4  | Reihenfolge der Punkte eines Xfoil Profils                          | 24 |
| 3.5  | Sortierung der Punkte eines Xfoil Profils (suchen des 8. Punktes)   | 25 |
| 3.6  | Systematische Darstellung der Abwicklung                            | 26 |
| 3.7  | Prinzipieller Ablauf der OnBEM Funktion für Propeller               | 28 |
| 3.8  | Diskretisierung des Rotors  | 29 |
| 3.9  | Vergleich der beiden Formulierungen von $a'$                        | 37 |
| 3.10 | Schärfen der Kante zur Verhinderung von Turbulenzen                 | 40 |
| 3.11 | Einstellungen für die Berechnung der Polare                         | 42 |
| 3.12 | Darstellung der berechneten Polare                                  | 42 |
| 3.13 | Extrapolation der Polare für einen Bereich von $360^\circ$          | 43 |
| 3.14 | Schubkoeffizient $C_n$ über den Radius bei verschiedenen $\lambda$  | 47 |
| 3.15 | Vergleich der Profile 90, 95 und 100                                | 48 |
| 3.16 | $C_L$ über $\alpha$   | 48 |
| 3.17 | $C_D$ über $\alpha$   | 49 |
| 3.18 | Verhältnis von $C_L/C_D$ mit verschiedenen Profilen                 | 49 |
| 3.19 | Vergleich des Schubes   | 50 |
| 3.20 | Vergleich der Leistung  | 50 |
| 3.21 | Vergleich der Auftriebsbeiwerte von verschiedenen Profilen          | 52 |
| 3.22 | Vergleich der Widerstandsbeiwerte von verschiedenen Profilen        | 52 |
| 3.23 | Vergleich der Schübe bei verschiedenen Profilen                     | 53 |
| 3.24 | Vergleich der Leistungen bei verschiedenen Profilen                 | 53 |



# Nomenklatur

## Abkürzungen

|           |   |
|-----------|---|
| $\dot{m}$ | Massenstrom   |
| $\lambda$ | lokale Schnelllaufzahl (tipspeedratio)              |
| $\alpha$  | Angriffswinkel                                      |
| $\Omega$  | Winkelgeschwindigkeit des Rotors                    |
| $\omega$  | Winkelgeschwindigkeit der Abströmung nach dem Rotor |
| $\Phi$    | Anströmwinkel                                       |
| $\rho$    | Dichte des Fluids                                   |
| $\sigma$  | Rotorsteifigkeit                                    |
| $\Theta$  | Neigungswinkel                                      |
| $\varphi$ | Nebenwinkel zum Anströmwinkel $\Phi$                |
| $a$       | axial induction factor                              |
| $a'$      | tangential induction factor                         |
| $C_D$     | Koeffizient der Widerstandskraft                    |
| $C_L$     | Koeffizient der Auftriebskraft                      |
| $C_n$     | Koeffizient der Normalkraft                         |
| $C_T$     | Schubbeiwert des Rotors                             |
| $C_t$     | Koeffizient der Tangentialkraft                     |
| $F_D$     | Widerstandskraft eines Profils (drag)               |
| $F_L$     | Auftriebskraft eines Profils (lift)                 |
| $F_N$     | Normalkraft   |
| $F_S$     | Schubkraft (thrust)                                 |
| $F_T$     | Tangentialkraft                                     |
| $a\_a$    | C++ Variable für axial induction factor             |
| $a\_r$    | C++ Variable für tangential induction factor        |
| BEM       | Blade Element Momentum Theorie/Methode              |
| $c$       | relative Anströmgeschwindigkeit                     |
| $dr$      | Radiusänderung, Wandstärke des Elements             |
| $E$       | Energie   |
| $F$       | Prandtl Tip Loss Faktor                             |
| $I$       | Impuls  |

|    |   |
|----|---|
| L  | Länge der Profilsehne (chordlength)       |
| M  | Drehmoment an der Rotorwelle              |
| NB | Anzahl der Rotorblätter(number of blades) |
| P  | Leistung                                  |
| p  | Druck                                     |
| R  | äußerster Radius                          |
| r  | mittlerer Radius eines Elements           |
| Re | Reynoldszahl                              |
| u  | Geschwindigkeit in Umfangsrichtung        |
| v  | Geschwindigkeit in axialer Richtung       |
| w  | Geschwindigkeit in radialer Richtung      |

**Hoch- und Tiefgestellte Indizes**

|   |                             |
|---|-----------------------------|
| 0 | Zustand weit vor dem Rotor  |
| 1 | Zustand weit nach dem Rotor |

# Kapitel 1

## Einleitung

### 1.1 Aufgabenstellung

Karl Waser nimmt mit seinem Motorsegelflieger an Wettbewerben der Elektrosegelflugklasse F5B teil und stellte zwei Propeller für diese Arbeit zur Verfügung. Bei den Wettbewerben wird versucht mit einer begrenzten Menge an Energie so lange wie möglich in der Luft zu bleiben und bestimmte Flugmanöver auszuführen.

Im ersten Abschnitt wird nach dem Start eine 150m lange Strecke abgeflogen, außerhalb der Strecke eine Wende ausgeführt und dann die selbe Strecke wieder retour geflogen. Je nach Flughöhe wird die Strecke ein weiteres mal in beiden Richtungen überflogen, bis die Flughöhe nicht mehr ausreicht und eine 2 bis 3 Sekunden lange Steigphase in die Wende eingebaut wird. Während dieser Phase wird der Motor mit Strom aus den Akkus versorgt und die dabei verwendete Energie gemessen. Im zweiten Abschnitt gilt es, das Modell so lange wie möglich mit der verbleibenden Energie unter Ausnutzung aller Gegebenheiten, wie zum Beispiel Thermik, in der Luft zu halten.

Für den Wettbewerb darf jeder sein eigenes ferngesteuertes Flugmodell mitbringen, in das ein Regler eingebaut wird, der dann im Betrieb Strom und Spannung, die dem Antriebsmotor zugeführt werden, misst und ab dem vorgegebenen Wert an verwandelter Energie den Strom unterbricht.

Aufgrund der Erfahrung sind die Komponenten des Modells, welche die Flugeigenschaften beeinflussen, schon sehr ausgereift. Rumpf und Flügel sind ausgiebig getestet worden. Bei den elektrischen Komponenten besteht relativ wenig Spielraum für Verbesserungen, da es sich um Standardbauteile handelt, die bis an die physikalischen Grenzen ausgereizt wurden. Also bleibt nur noch der Propeller, um Verbesserungen des Modells zu bewirken.

### 1.2 Ausgangslage

Die bisher verwendeten Propeller wurden empirisch entwickelt oder zugekauft. Diese Propeller sind schon sehr gut an die Bedürfnisse des Wettbewerbs angepasst. Es ist nicht bekannt, ob oder wie die Propeller noch verbessert werden können. Zu diesem Zweck soll eine Möglichkeit

geboten werden, die bestehenden Propeller zu berechnen und zu analysieren. Ausgehend von der bestehenden Geometrie kann dann versucht werden die Propellereigenschaften zu verbessern.

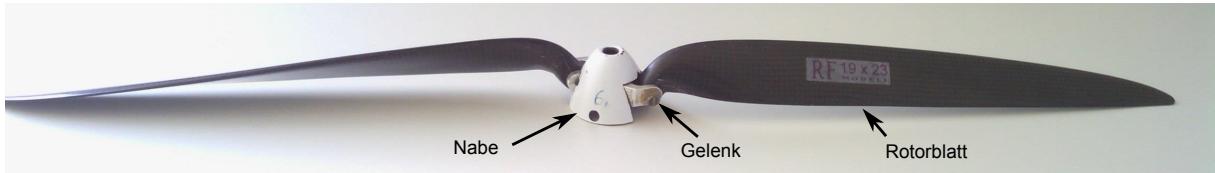
Laut im Betrieb durchgeföhrten Messungen soll ein aktueller Propeller bei einer Drehzahl zwischen 8500 und 9200 rpm eine Leistung zwischen 5,5 und 6 kW verbrauchen. Das Flugzeug fliegt beim Einschalten des Motors mit einer Geschwindigkeit von 80 bis 100 km/h und am Ende der Beschleunigungsphase mit 250 bis 280 km/h.

# Kapitel 2

## Theoretische Grundlagen

### 2.1 Propeller

Ein Propeller besteht typischerweise aus mindestens zwei Rotorblättern, die um eine Rotationsachse rotieren. Die Winkelgeschwindigkeit dieser Rotation wird als  $\Omega$  und die Anzahl der Rotorblätter als NB bezeichnet. Die Rotorblätter sind ähnlich wie Flugzeugflügel aerodynamisch geformt und erzeugen durch Umlenkung der anströmenden Luft eine Kraft in Flugrichtung auf den Propeller. Die Größe eines Propellers wird häufig mittels Durchmesser angegeben. Für die Berechnung ist es jedoch praktischer den Radius R zu verwenden. R ist der Abstand der Rotorblattspitze von der Rotationsachse.

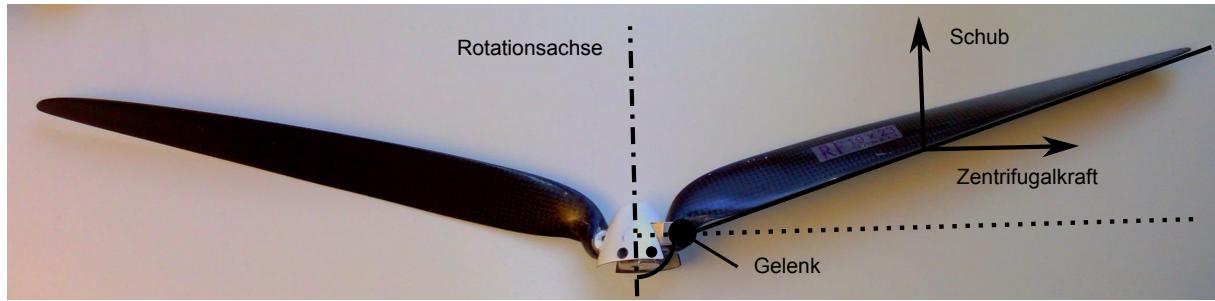


**Abbildung 2.1:** Ausgangspropeller

Wie in Abb. 2.1 zu sehen ist besteht der Propeller aus zwei Komponenten: der Nabe und den Rotorblättern. Die Nabe ist auf einer Welle montiert, die vom Motor angetrieben wird. Die Rotorblätter sind mit einem Gelenk an der Nabe befestigt. Sie werden häufig in Handarbeit aus Karbon gefertigt.

Für den Elektrosegelflug gibt es eigene Propeller, bei denen die Rotorblätter gelenkig an der Nabe montiert sind. Im Segelbetrieb liegen die Rotorblätter am Flugzeuggrumpf an. Dadurch haben sie nur wenig Luftwiderstand. Wird der Motor eingeschaltet beginnt sich der Propeller zu drehen und die Rotorblätter werden durch die Zentrifugalkraft nach außen gezogen. Dadurch klappen sie nach außen. Wenn es keinen Anschlag gibt, der die Position der Rotorblätter festlegt, stellt sich ein Kräftegleichgewicht ein. Die Zentrifugalkraft zieht die Rotorblätter nach außen. Ohne weitere Kräfte würden sie eine Position normal zur Rotationsachse einnehmen. Durch die Interaktion des Propellers mit der Luft wird Schub  $F_S$  erzeugt, eine Kraft die in Flugrichtung auf den Propeller wirkt. Der Schub zieht die Rotorblätter nach vorne, über den rechten Winkel hinaus. Die beiden Kräfte bewirken mit dem jeweiligen Abstand zum Gelenk ein Moment. In

der Gleichgewichtslage sind diese beiden Momente gleich groß und die Rotorblätter sind, ähnlich wie in Abb. 2.2 gezeigt, leicht nach vorne geneigt.



**Abbildung 2.2:** Auf den Propeller wirkende Kräfte im Betrieb

## 2.2 Rotorblatt

Das Rotorblatt kann als Stapel von einzelnen Profilen angesehen werden. Jedem Profil kann ein eigener Radius  $r$  zugeordnet werden. Weiters wird die Definition der Rotorsteifigkeit ( $\sigma$ ), einer von der Rotorblatt abhängige Geometriegröße, verwendet:

$$\sigma(r) = \frac{L(r)NB}{2\pi r} \quad (2.1)$$

## 2.3 Profil

Als Profil wird hier eine geschlossene Kurve, die den Querschnitt eines Rotorblattes darstellt, verstanden. In den NACA und Eppler Serien sind Profile für verschiedenste Anwendungsfälle zusammengefasst worden. Für die Erzeugung von Auftrieb ist es notwendig, die anströmende Luft umzulenken. Dazu ist ein asymmetrisches Profil besonders gut geeignet. Die Asymmetrie wird durch die Krümmung charakterisiert. Für einen Propeller kommen nur Profile in Frage, die einen Auftrieb erzeugen und somit eine Krümmung besitzen. In Abb. 2.3 sind die Begriffe zur Charakterisierung eines Profils eingetragen. Die Profilsehne ist die direkte Verbindung zwischen Vorder- und Hinterkante des Profils. Ihre Länge wird auch als Profillänge  $L$  angegeben. Wird die Profillänge horizontal angeordnet und auf die dimensionslose Länge 1 normiert, können verschiedene Profile gut miteinander verglichen werden. Wird für den Einsatz eine bestimmte Größe des Profils gefordert, werden alle Längen des Profils mit der geordneten Profillänge multipliziert. Die Skelettlinie verläuft mittig zwischen der Ober- und Unterseite des Profils. Die Abweichung der Skelettlinie von der Profilsehne ist ein Maß für die Krümmung des Profils. Die Krümmung wird angegeben, indem der maximale Abstand der Skelettlinie von der Profilsehne durch die Profillänge dividiert wird und als Prozentzahl angegeben wird. Auch die Dicke des Profils wird auf die Profillänge bezogen in Prozent angegeben. Sowohl die Position der maximalen Dicke als auch der maximalen Krümmung kann als Abstand von der Vorderkante angegeben werden. Diese Abstände werden ebenfalls auf die Profillänge bezogen und in Prozent angegeben. Wird ein Profil von einem Fluid, wie in Abb. 2.4 gezeigt, angeströmt, wirkt eine Widerstandskraft (*Drag*)  $F_D$  in

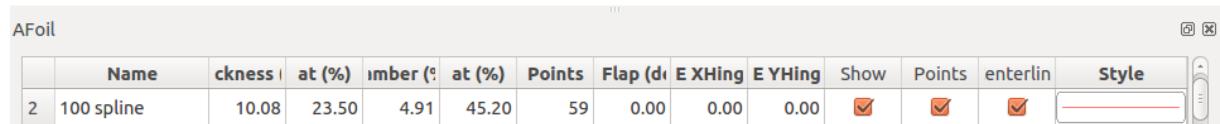
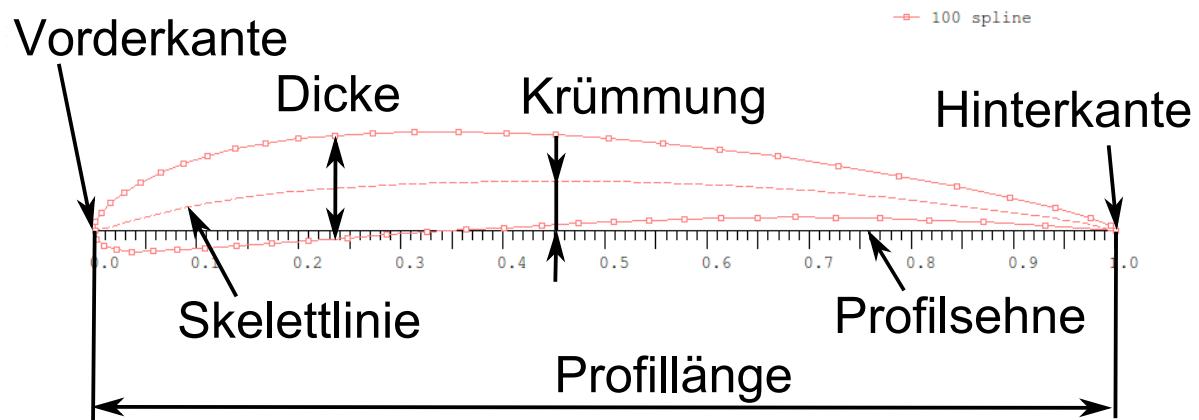


Abbildung 2.3: Charakterisierung eines Profils

Richtung der Fluidbewegung auf das Profil. Gleichzeitig wird durch die Umlenkung des Fluids eine Kraft quer zur Bewegungsrichtung erzeugt. Sie wird als Auftriebskraft (Lift)  $F_L$  bezeichnet.

### anströmendes Fluid

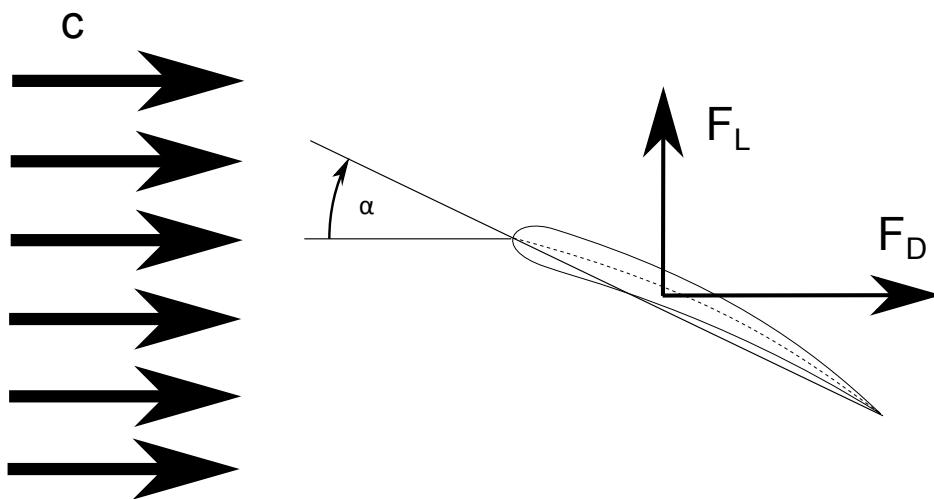


Abbildung 2.4: Am Profil angreifende Kräfte

Die Größe der Kräfte ist abhängig von der Dichte des Fluids  $\rho$ , der Profillänge L, der Anströmgeschwindigkeit c, dem Angriffswinkel  $\alpha$  und der Profilform. Profilform und Angriffswinkel werden in Koeffizienten für Auftrieb  $C_L$  und Widerstand  $C_D$  zusammengefasst. Für jedes  $\alpha$  müssen eigene Werte für die Koeffizienten aus Windkanalmessungen oder Simulationen errechnet werden. Die Kräfte können mit den Formeln

$$F_L = \frac{1}{2} \rho C_L c^2 L \quad (2.2)$$

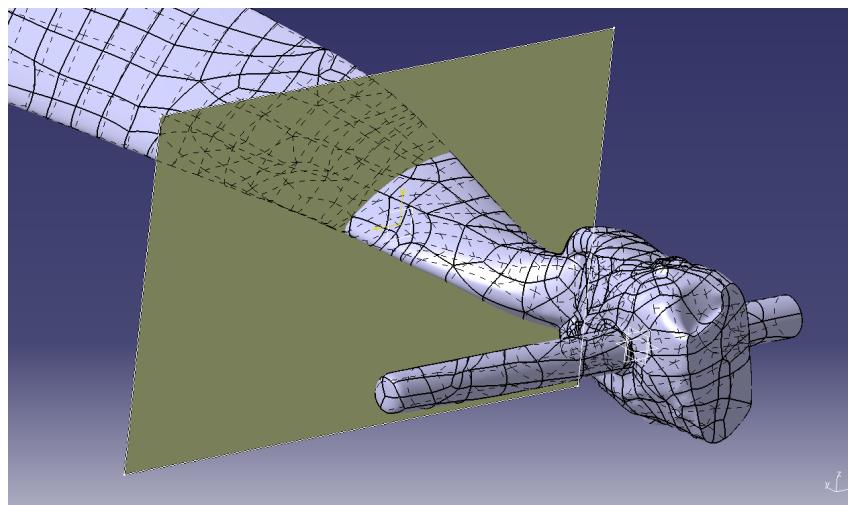
$$F_D = \frac{1}{2} \rho C_D c^2 L \quad (2.3)$$

berechnet werden. Es ist zu bedenken, dass die Widerstandskraft unabhängig von der Lage des Profils in Richtung der Anströmgeschwindigkeit wirkt und die Auftriebskraft normal darauf. Bei der in Abb. 2.4 gezeigten Anströmung handelt es sich um eine homogene wirbelfreie Anströmung. Verschiebungen in der Bildebene, in und quer zur Anströmungsrichtung, beeinflussen das Strömungsverhalten nicht. Rein der Angriffswinkel  $\alpha$  beeinflusst die Strömung. Somit ist es auch unerheblich um welchen Punkt das Profil gedreht wird. In Polaren werden die Verläufe der Auftriebs und Widerstandsbeiwerte für einen bestimmten Bereich von  $\alpha$  zusammengefasst.

Um den Querschnitt eines Rotorblattes zu erstellen und damit ein Profil zu erzeugen gibt es zwei Möglichkeiten. Zum einen kann man das Rotorblatt mit einer Ebene schneiden oder mit einem Zylinder.

### 2.3.1 ebene Schnitte

Schneidet man den Rotor, wie in Abb. 2.5, mit einer Ebene, so erhält man direkt ein zweidimensionales Profil. Dieses Profil kann direkt für die zweidimensionale Berechnung der Auftriebs und Widerstandsbeiwerte verwendet werden. Durch ein einfaches Stapeln der Profile kann die dreidimensionale Ausgangsgeometrie des Rotorblattes wiederhergestellt werden.



**Abbildung 2.5:** Schnitt des Rotorblattes mit einer Ebene

### 2.3.2 Zylinderschnitte

Schneidet man den Rotor, wie in Abb. 2.6, mit einem Zylinder, dessen Rotationsachse der Rotationsachse des Rotors entspricht, erhält man eine dreidimensionale Kurve. Um diese Kurve mit zweidimensionalen Methoden verarbeiten zu können, muss die Kurve erst wie eine Zylindermantelfläche abgewickelt werden. Dadurch wird der Abstand zwischen Vorder- und Hinterkante länger dargestellt als bei ebenen Schnitten. Werden nun die abgewickelten Profile wie ebene Profile übereinander gestapelt, so entsteht eine verzerrte Rotorblattgeometrie, deren Tiefe größer ist als die ursprüngliche. Um Geometrie realitätsgtreu wiederzugeben, müssen die abgewickelten Profile wieder auf einen Zylinder aufgewickelt werden, bevor sie gestapelt werden können. Betrachtet man in einem, mit dem Rotor mitdrehendem Koordinatensystem ein Fluidteilchen auf seine Bahn so bewegt es sich näherungsweise auf einem Zylinder. Auch die Distanz die es in Kontakt mit dem Rotor zurücklegt ist länger als es bei einem ebenen Schnitt und geradliniger Anströmung der Fall wäre. Bei einer Simulation die der in Abb. 2.4 dargestellten entspricht liefern die Zylinderschnitte ein Ergebnis, das eher der Wirklichkeit entspricht.

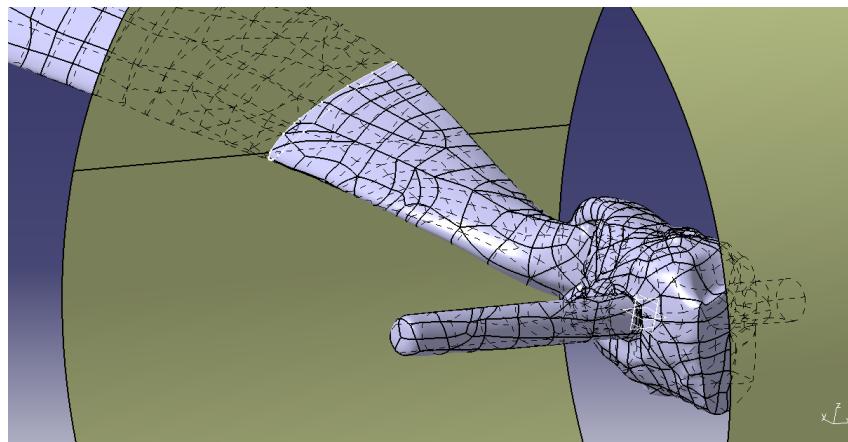


Abbildung 2.6: Schneiden mit einer Zylindermantelfläche

## 2.4 BEM Theorie

Das Grundprinzip der Berechnungen in Qblade ist die Blade Element Momentum Methode (BEM), welche in einer iterativen Berechnung die Interaktion zwischen einem strömenden Fluid und einem Rotor errechnet. Das Prinzip der BEM Methode ist in den beiden Artikeln [6] und [5] gut beschrieben, soll hier aber noch einmal zusammengefasst werden, da dies sehr wichtig für das weitere Vorgehen ist.

### 2.4.1 Blade Element Momentum Theorie

In der BEM Theorie wird der Rotor in Ringe mit der Dicke  $dr$  zerlegt. Für jeden dieser Ringe wird nun ein konstantes Profil angenommen und auch die Strömung wird in diesem Bereich gleichmäßig angenommen. Betrachtet man das Fluid, das durch diese Ringe strömt erhält man ein rotationssymmetrisches Kontrollvolumen. Dieses Kontrollvolumen wird in zwei Bereiche unterteilt. Der Anströmung vor dem Rotor und der Abströmung nach dem Rotor. Zwischen den beiden Bereichen findet die Interaktion mit dem Rotor statt. Die BEM Theorie beinhaltet die Annahme, dass sich die einzelnen Kontrollvolumina nicht gegenseitig beeinflussen und so voneinander unabhängige Elemente entstehen. Abb. 2.7 zeigt wie sich die Elemente entlang der Strömungsrichtung erstrecken. Aufgrund der Geschwindigkeitsunterschiede kann die Dicke des Elements nicht als konstant angenommen werden. Der Index 0 bezeichnet einen Punkt, der so weit vor dem Rotor liegt, dass er von ihm nicht mehr beeinflusst wird. Der Index 1 hingegen bezeichnet einen Punkt weit nach dem Rotor. Durch Aufstellen von Impuls und Drallsätzen werden die Zu-

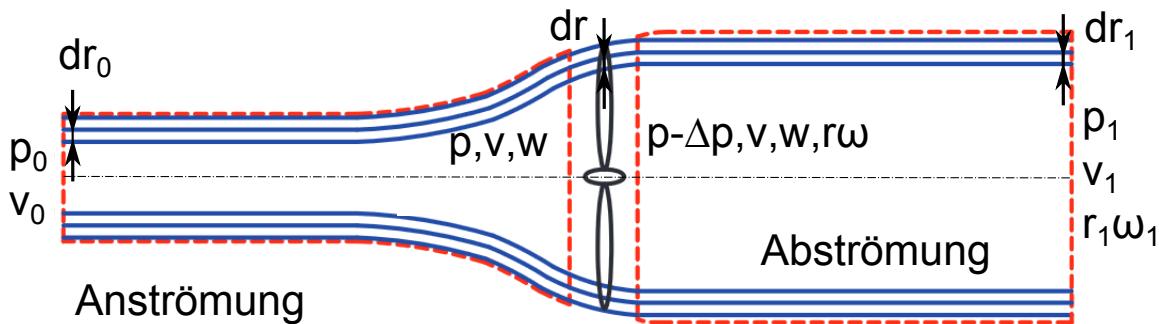


Abbildung 2.7: Zustandsgrößen am Beispiel einer Turbine, in Anlehnung an [6, S 30]

stände an den beiden Punkten und am Rotor verglichen. Da ein stationärer Vorgang untersucht wird ändert sich der Impuls innerhalb des Kontrollvolumens nicht über die Zeit. Nach [7, S37] muss, unter dieser Bedingung, der austretende Impulsfluss minus dem eintretendem Impulsfluss gleich der Summe der auf das System wirkenden Kräfte sein.

Wie bereits erwähnt, erfolgt in jedem dieser Ringe die Erzeugung des Schubes durch eine Umlenkung des Fluids. Da sich die Ringe gegenseitig nicht beeinflussen, darf keine Geschwindigkeit in radialer Richtung auftreten. Also muss die Umlenkung in Umfangsrichtung erfolgen und führt zu einem Drall des abströmenden Fluids. Analog zum Impuls führt eine Differenz der Drallströme zu einem auf das Fluid wirkenden Moment.

Weiters wird in der BEM Theorie davon ausgegangen, dass Geschwindigkeitsdifferenz zwischen der Geschwindigkeit der Durchströmung des Rotors  $v$  und der Anströmgeschwindigkeit  $v_0$  gleich groß ist wie die Differenz zwischen der Geschwindigkeit weit hinter dem Rotor  $v_1$  und der Durchströmungsgeschwindigkeit  $v$ . Der axial inductionfactor  $a$  beschreibt wie die Fluidbewegung an der Stelle des Rotors von dessen Schub beeinflusst wird und ist bei Turbinen positiv.[5]

Dabei gilt:

$$v = v_0(1 - a) \quad (2.4)$$

und :

$$v_1 = v_0(1 - 2a) \quad (2.5)$$

Für die Umfangskomponente wird der tangential induction factor  $a'$  definiert. Dieser Faktor erzeugt eine Beziehung zwischen der Winkelgeschwindigkeit des Rotors  $\Omega$  und der Winkelgeschwindigkeit der Abströmung direkt nach dem Rotor  $\omega$ . So dass gilt:

$$\omega = 2a'\Omega \quad (2.6)$$

Für die Umströmung der Profile direkt in der Rotorebene wird, bei der Ermittlung der Umfangsgeschwindigkeit des Fluids, der Mittelwert der Umfangsgeschwindigkeiten vor und nach der Rotorebene gebildet. Die Anströmung erfolgt rotationsfrei. Somit wird  $\frac{\omega}{2} = a'\Omega$  verwendet.

#### 2.4.2 BEM für Windturbinen

Abedi [6] liefert in seiner Masterarbeit eine sehr genaue Herleitung des Formelwerkes für die BEM Berechnung einer Windturbine. Die Ergebnisse dieser Herleitung (Gl. (2.48) und Gl. (2.52)) sind exakt die Formeln, die in der QBlade Dokumentation [11] angegeben wurden und bieten daher eine gute Grundlage für die Abänderungen zur Berechnung von Propellern. Diese Herleitung soll nun kurz erklärt werden. Die verwendeten Größen sind in Abb. 2.7 dargestellt.

Stellt man die Kontinuitätsgleichung zwischen dem Punkt direkt nach der Propellerebene und einem Punkt weit nach dem Propeller in einem Ringelement auf, so erhält man:

$$v_1 r_1 dr_1 = vrdr \quad (2.7)$$

Wobei  $v$  die Geschwindigkeit des Fluids in axialer Richtung ist,  $r$  der mittlere Radius des Elements und  $dr$  die Dicke des Elements. Da sich, laut Blade Element Momentum Theorie, die ringförmigen Elemente im Nachlauf nicht gegenseitig beeinflussen können, können auch keine Momente darauf wirken und der Drall muss gleich bleiben:

$$\omega_1 r_1^2 = \omega r^2 = ur \quad (2.8)$$

Da wir davon ausgehen, dass die Anströmung drallfrei erfolgt und wir aber gerade eben den Drall der Abströmung berechnet haben, besteht eine Differenz zwischen Eintritts und Austrittsdrall des Systems. Diese Differenz ruft ein Moment  $M$  hervor.

$$dM = \rho v \omega r^2 dA \quad (2.9)$$

Stellt man die Bernoulligleichungen für die An- und Abströmung auf, erhält man die beiden Gleichungen Gl. (2.10) und Gl. (2.11). Dabei ist  $v$  die Geschwindigkeit in axialer Richtung,  $w$  in radialer Richtung,  $\rho$  die Dichte des Fluids und  $p$  der im Fluid herrschende Druck.

$$H_0 = p_0 + \frac{1}{2}\rho v_0^2 = p + \frac{1}{2}\rho(v^2 + w^2) \quad (2.10)$$

In der Abströmung bewegt sich das Fluid mit konstanter Dichte bei konstantem Druck. Um der Massenerhaltung gerecht zu werden, muss der Querschnitt bei konstanter axialer Geschwindigkeit ebenfalls konstant bleiben. Damit kann weit nach dem Rotor keine Geschwindigkeit in radialer Richtung auftreten und man erhält:

$$H_1 = p - \Delta p + \frac{1}{2}\rho(v^2 + w^2 + \omega^2 r^2) = p_1 + \frac{1}{2}\rho(v_1^2 + \omega_1^2 r_1^2) \quad (2.11)$$

Bildet man daraus die Differenz so erhält man:

$$\Delta H = H_1 - H_0 = -\Delta p + \frac{1}{2}\rho\omega^2 r^2 \quad (2.12)$$

oder:

$$\Delta H = H_1 - H_0 = p_1 + \frac{1}{2}\rho(v_1^2 + \omega_1^2 r_1^2) - p_0 - \frac{1}{2}\rho v_0^2 \quad (2.13)$$

Dies kann umgeformt werden auf:

$$p_0 - p_1 = \frac{1}{2}\rho(v_1^2 - v_0^2) + \frac{1}{2}\rho\omega_1^2 r_1^2 - (H_1 - H_0) \quad (2.14)$$

Setzt man nun Gl. (2.12) ein, kommt man auf

$$p_0 - p_1 = \frac{1}{2}\rho(v_1^2 - v_0^2) + \frac{1}{2}\rho(\omega_1^2 r_1^2 - \omega^2 r^2) + \Delta p \quad (2.15)$$

Bei einer Turbine kann man sagen, dass sich der Nachlauf um die Winkelgeschwindigkeit  $\omega$  schneller dreht als die Anströmung, wodurch sich der Druck um  $\Delta p$  verringert. Bei einem Propeller ist dies genau umgekehrt, aber die Bernoulligleichung ist in der gleichen Form zulässig. Hier ist aber eine Druckerhöhung zu erwarten. Stellt man nun die Bernoulligleichung für ein mit dem Rotor mitdrehendes Koordinatensystem auf, so erhält man:

$$\Delta p = \frac{1}{2}\rho[(\Omega + \omega)^2 - \Omega^2]r^2 = \rho(\Omega + \frac{1}{2}\omega)\omega r^2 \quad (2.16)$$

Verbindet man nun Gl. (2.8), Gl. (2.15) und Gl. (2.16), so erhält man:

$$p_0 - p_1 = \frac{1}{2}\rho(v_1^2 - v_0^2) + \rho\omega_1 r_1^2 (\Omega + \frac{1}{2}\omega) \quad (2.17)$$

Für die weitere Rechnung sind einige Annahmen zu treffen. Die Rotationsgeschwindigkeit  $\omega$  der Abströmung ist um ein Vielfaches kleiner als die Rotationsgeschwindigkeit  $\Omega$  des Rotors. Daher können alle Terme mit  $\omega^2$  vernachlässigt werden. Da sich sowohl An- als auch Abströmung in freier Umgebung befinden, kann man davon ausgehen, dass die Drücke weit vor und weit nach dem Rotor gleich groß sind  $p_0 = p_1$ . Außerdem wird angenommen, dass der Druckabfall  $\Delta p$  dem Druckabfall durch  $(H_1 - H_0)$  entspricht.

Gemäß der Impulsgleichung der axialen Strömung entspricht die Differenz der Impulsströme der auf das Fluid wirkenden Kraft  $F_S$ . Stellt man nun die allgemeine Impulsgleichung auf erhält man:

$$dF_S = (v_1 - v_0)dm = (v_1 - v_0)\rho v dA \quad (2.18)$$

Setzt man Gl. (2.4) und Gl. (2.5) ein, so erhält man:

$$dF_S = v_0(1 - 2a - 1)\rho v_0(1 - a)dA \quad (2.19)$$

Will man aber nicht die Kraft auf das Fluid, sondern auf den Rotor berechnen, muss das Vorzeichen einmal geändert werden:

$$dF_S = 2av_0^2\rho(1 - a)dA \quad (2.20)$$

Die Querschnittsfläche des Elements ist dabei ein Kreisring mit:

$$dA = 2\pi r dr \quad (2.21)$$

Damit wird Gl. (2.20) zu:

$$dF_S = 4\pi\rho v_0^2(1 - a)ardr \quad (2.22)$$

Betrachtet man die Druckdifferenz am Rotor und berücksichtigt Gl. (2.16) und Gl. (2.21), kann man alternativ auch sagen:

$$dF_S = \Delta pdA = 2\pi\rho(\Omega + \frac{1}{2}\omega)\omega r^3 dr \quad (2.23)$$

Durch Einsetzen von Gl. (2.6) wird daraus:

$$dF_S = 4\pi\rho\Omega^2(1 + a')a'r^3 dr \quad (2.24)$$

Durch Gleichsetzen von Gl. (2.22) und Gl. (2.24), kann man nun eine Beziehung zwischen  $a$  und  $a'$  aufstellen:

$$v_0^2(1 - a) = \Omega^2 r^2(1 + a')a' \quad (2.25)$$

Bringt man alle Ausdrücke mit  $a$  und  $a'$  auf die linke Seite kann die rechte Seite durch die lokale Schnelllaufzahl  $\lambda$  ersetzt werden. Die lokale Schnelllaufzahl ist das Verhältnis von Umfangsgeschwindigkeit des Rotors an der Stelle des aktuellen Radius  $r$  zur Anströmgeschwindigkeit weit vor dem Rotor:

$$\lambda = \frac{\Omega r}{v_0} \quad (2.26)$$

Damit wird aus Gl. (2.25):

$$\frac{(1 - a)a}{(1 + a')a'} = \frac{\Omega^2 r^2}{v_0^2} = \lambda^2 \quad (2.27)$$

Durch Umformen auf  $a'$  erhält man:

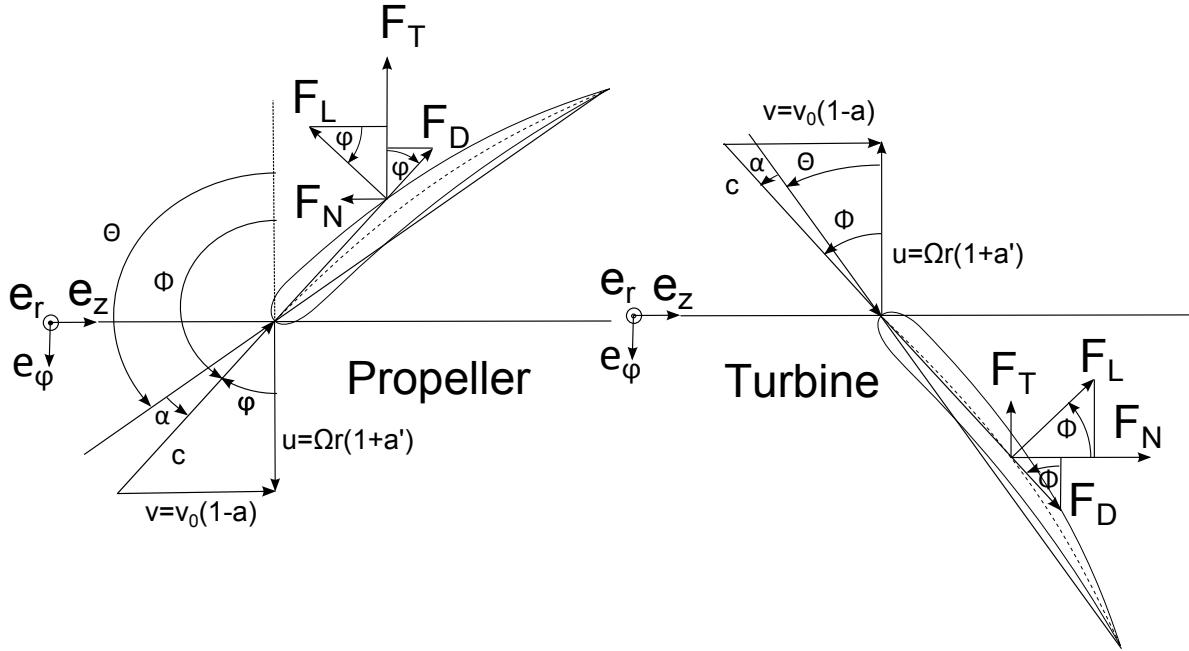
$$a'^2 + a' - \frac{(1 - a)a}{\lambda^2} = 0 \quad (2.28)$$

Löst man die quadratische Gleichung und betrachtet das physikalisch relevante Ergebnis mit der positiven Lösung der Wurzel kann man  $a'$  ausdrücken als:

$$a' = \frac{1}{2} \left( \sqrt{1 + \frac{4a(1 - a)}{\lambda^2}} - 1 \right) \quad (2.29)$$

Wenn man nun Gl. (2.4), Gl. (2.6) und Gl. (2.21) in Gl. (2.9) einsetzt, kommt man auf

$$dM = \rho v \omega r^2 dA = 4\pi \rho v_0 \Omega (1 - a) a' r^3 dr \quad (2.30)$$



**Abbildung 2.8:** Geschwindigkeiten im Vergleich von Propeller und Turbine

Die Gleichungen Gl. (2.22) und Gl. (2.30) sind im Prinzip simple Impuls- und Drallsätze, die allgemein gültig sind und im Folgenden Grundlage für die eigentlichen Formeln zur Berechnung von Turbinen und Propellern sind. Betrachtet man das Geschwindigkeitsdreieck am Profil, wie es in Abb. 2.8 dargestellt ist, kann man bei der Turbine folgende Gleichungen aufstellen.

$$\alpha = \Phi - \Theta \quad (2.31)$$

$$\tan(\Phi) = \frac{(1-a)v_0}{(1+a')\Omega r} \quad (2.32)$$

$$\sin(\Phi)c = v_0(1-a) \quad (2.33)$$

$$\cos(\Phi)c = \Omega r(1+a') \quad (2.34)$$

Wie in Abb. 2.8 zu sehen ist können die in Gl. (2.2) und Gl. (2.3) berechneten Kräfte nicht direkt zur Berechnung von Antriebsmoment und Widerstandskraft der Windturbine verwendet werden, stattdessen muss man die beiden Kräfte gemäß den Geometriebedingungen folgendermaßen umrechnen. So ergeben sich Normalkraft und Tangentialkraft zu:

$$F_N = F_L \cos(\Phi) + F_D \sin(\Phi) \quad (2.35)$$

$$F_T = F_L \sin(\Phi) - F_D \cos(\Phi) \quad (2.36)$$

Normalisiert man die beiden Kräfte zu dimensionslosen Größen erhält man:

$$C_n = C_L \cos(\Phi) + C_D \sin(\Phi) \quad (2.37)$$

$$C_t = C_L \sin(\Phi) - C_D \cos(\Phi) \quad (2.38)$$

Hierbei gilt:

$$F_N = C_n \frac{1}{2} \rho c^2 L \quad (2.39)$$

$$F_T = C_t \frac{1}{2} \rho c^2 L \quad (2.40)$$

Multipliziert man die Normalkraft ( $F_N$ ), die pro Längeneinheit angegeben ist, mit der Dicke des Elements und mit der Anzahl der Rotorblätter (NB) so erhält man den gesamten Schub des Elements:

$$dF_S = NB F_N dr \quad (2.41)$$

Gleiches gilt für die Tangentialkraft. Multipliziert man diese noch mit dem aktuellen Radius erhält man das Drehmoment:

$$dM = r N B F_T dr \quad (2.42)$$

Setzt man nun Gl. (2.39) und c aus Gl. (2.33) in Gl. (2.41) ein, kommt man auf:

$$dF_S = \frac{1}{2} \rho N B \frac{v_0^2 (1-a)^2}{\sin^2(\Phi)} C_n L dr \quad (2.43)$$

aus Gl. (2.40), Gl. (2.34) und Gl. (2.42) gilt analog dazu für das Drehmoment:

$$dM = \frac{1}{2} \rho L N B \frac{v_0 (1-a) \Omega r (1+a')}{\sin(\Phi) \cos(\Phi)} C_t r dr \quad (2.44)$$

Setzt man nun die Gleichungen Gl. (2.22) und Gl. (2.43) gleich erhält man:

$$4\pi \rho v_0^2 (1-a) ar dr = \frac{1}{2} \rho N B \frac{v_0^2 (1-a)^2}{\sin^2(\Phi)} C_n L dr \quad (2.45)$$

Durch Kürzen erhält man:

$$4\pi ar = \frac{1}{2} N B \frac{(1-a)}{\sin^2(\Phi)} C_n c \quad (2.46)$$

Bringt man alle Ausdrücke mit a auf die linke Seite und verwendet die Definition von Gl. (2.1) erhält man:

$$\frac{a}{1-a} = \frac{C_n \sigma}{4 \sin^2(\Phi)} \quad (2.47)$$

Diese Gleichung kann auf a umgeformt werden:

$$a = \frac{1}{\frac{4 \sin^2(\Phi)}{C_n \sigma} + 1} \quad (2.48)$$

Exakt diese Gleichung wird auch in QBlade für die Berechnung des axial induction factors verwendet. Für den radial inductionfactor a' kann man ähnlich vorgehen und die Gleichungen Gl. (2.30) und Gl. (2.44) gleichsetzen

$$\rho v \omega r^2 dA = 4\pi \rho v_0 \Omega (1-a) a' r^3 dr = \frac{1}{2} \rho L N B \frac{v_0 (1-a) \Omega r (1+a')}{\sin(\Phi) \cos(\Phi)} C_t r dr \quad (2.49)$$

Durch Kürzen wird daraus:

$$4\pi \Omega a' r = \frac{1}{2} c N B \frac{\Omega (1+a')}{\sin(\Phi) \cos(\Phi)} C_t \quad (2.50)$$

Bringt man wieder alle Ausdrücke mit  $a'$  nach links und verwendet die Definition von *Gl. (2.1)*, erhält man:

$$\frac{a'}{1+a'} = \frac{\sigma C_t}{4\sin(\Phi)\cos(\Phi)} \quad (2.51)$$

Diese Gleichung kann auf  $a'$  umgeformt werden:

$$a' = \frac{1}{\frac{4\sin(\Phi)\cos(\Phi)}{\sigma C_t} - 1} \quad (2.52)$$

Diese Formulierung ist in der Literatur üblich und findet sich auch in der Dokumentation von QBlade [11] wieder. Im Quellcode von QBlade ist jedoch die Formulierung von *Gl. (2.29)* zu finden.

### 2.4.3 BEM für Propeller

Nun soll aber keine Turbine sondern ein Propeller gerechnet werden. Die in *Gl. (2.9)* und *Gl. (2.18)* verwendeten Impulssätze sind allgemein gültig, unabhängig davon, ob die auftretenden Geschwindigkeiten größer oder kleiner werden. Betrachtet man die Geschwindigkeitsdreiecke vor und nach dem Profil eines Propellers und bedenkt die Aufgabe, dann ist davon auszugehen, dass die axiale Geschwindigkeit zunehmen wird. Die tangentiale Komponente ist zum einen räumlich umgedreht und zum anderen wird ihr Betrag kleiner werden, da sich die Abströmung in die gleiche Richtung drehen wird wie der Propeller selbst. Um nicht den ganzen Formelapparat umzuschreiben, sollen die Induktionsfaktoren  $a$  und  $a'$  negativ angenommen werden und auch die Rotationsgeschwindigkeit soll mit negativen Werten angegeben werden. Betrachtet man nun *Gl. (2.22)*, so wird der Ausdruck in der Klammer größer als 1 werden und das  $a$  direkt dahinter bewirkt, dass sich das Vorzeichen des ganzen Terms ändert. Dies passt auch weiterhin zu unserem Gedankenmodell. Durch die Klammer in *Gl. (2.30)* kann man auch hier davon ausgehen, dass der Betrag des Terms tendenziell größer als bei der Turbine ausfallen wird. Da aber sowohl  $\Omega$  als auch  $a'$  negative Vorzeichen besitzen, wird die Richtung des Moments gleich der Turbine bleiben. Dies passt ebenfalls zu unserem Gedankenmodell, da, wie in *Abb. 2.8* zu sehen, die Tangentialkraft  $F_T$  unabhängig von der Rotationsrichtung des Rotors in die gleiche Richtung zeigt. Das größte Problem sind die Winkel. Wie in *Abb. 2.8* zu erkennen ist, sind bei der Turbine alle Winkel kleiner als  $90^\circ$ . Beim Propeller sind jedoch die Winkel  $\Phi$  und  $\Theta$  zwischen  $90^\circ$  und  $180^\circ$ , wenn man die Definition beibehalten möchte.

Die Formel für  $\alpha$  bleibt gleich wie in *Gl. (2.31)*. *Gl. (2.32)* zur Berechnung von  $\Phi$  ist in der Form aber nicht mehr zulässig. Es kann aber stattdessen der Nebenwinkel  $\varphi$  verwendet werden.

$$\varphi = \pi - \Phi \quad (2.53)$$

Bei Turbinen kann  $\Phi$  einfach aus den geometrischen Beziehungen errechnet werden.

$$\tan(\Phi) = \frac{(1-a)v_0}{(1+a')\Omega r} \quad (2.54)$$

Verwendet man die Definition der lokalen Schnellaufzahl aus *Gl. (2.26)* so kann dies zusammengefasst werden zu:

$$\Phi = \text{atan}\left(\frac{(1-a)}{(1+a')\lambda}\right) \quad (2.55)$$

Bei Propellern ergibt der tan dieser Formel aber der Nebenwinkel  $\varphi$ :

$$\varphi = \text{atan}\left(\frac{(1-a)}{(1+a')\lambda}\right) \quad (2.56)$$

Mit Gl. (2.53) kann wieder  $\Phi$  berechnet werden:

$$\Phi = \pi - \text{atan}\left(\frac{(1-a)}{(1+a')\lambda}\right) \quad (2.57)$$

Für beide Fälle kann der Angriffswinkel aus der Differenz von Anströmwinkel und Neigungswinkel bestimmt werden:

$$\alpha = \Phi - \Theta \quad (2.58)$$

Die Gl. (2.2) und Gl. (2.65) sind nur auf das Profil bezogen und bleiben weiterhin so definiert. Für die Gl. (2.35) und Gl. (2.36) ist die veränderte Geometrie zu berücksichtigen.

$$F_N = -F_L \cos(\varphi) + F_D \sin(\varphi) \quad (2.59)$$

$$F_T = F_L \sin(\varphi) + F_D \cos(\varphi) \quad (2.60)$$

Da jedoch  $\sin(\pi - x) = \sin(x)$  und  $\cos(\pi - x) = -\cos(x)$  gilt, kann auch hier weiter mit den Gleichungen Gl. (2.35) und Gl. (2.36) gearbeitet werden. Gl. (2.39) und Gl. (2.40) bleiben definitionsgemäß weiter gleich. Also gelten auch beim Propeller die Gleichungen Gl. (2.37) und Gl. (2.38). Die in Gl. (2.33) und Gl. (2.34) berechnete Relativgeschwindigkeit ist jene Geschwindigkeit, mit der das Profil angeströmt wird, um die Auftriebs- und Widerstandsbeiwerte zu ermitteln. Diese hat sowohl bei der Turbine als auch beim Propeller positiv zu sein. Mit den gleichen Gedanken wie bei Gl. (2.35) und Gl. (2.36) kann man auch hier wieder die Gleichungen für die Turbine beibehalten. Die in Gl. (2.1) berechnete Rotorsteifigkeit ist eine vom Rotorblatt abhängige Größe, die unabhängig von der Orientierung positiv zu sein hat.

Die in Gl. (2.41) und Gl. (2.42) auftretenden Kräfte haben sich bereits zuvor als richtig erwiesen. Also besteht auch hier kein Unterschied. Alle weiteren Gleichungen sind rein aus den Umformungen der bisher besprochenen Gleichungen entstanden und sind demnach ebenfalls für Propeller anwendbar. Der einzige Unterschied zwischen Propellern und Turbinen liegt also in der Berechnung von  $\Phi$ . Wird die Berechnung dieses Winkels also entsprechend aufgesetzt, kann die BEM Methode ohne Weiteres auch auf Propeller angewandt werden.

#### 2.4.4 Korrekturfaktoren

Dieses System der Berechnung hat noch einige Schwachstellen, beziehungsweise werden einige wichtige Einflüsse bisher noch nicht berücksichtigt. Dafür werden verschiedene Korrekturfaktoren in die bisherigen Gleichungen mit aufgenommen. Zum Beispiel entstehen an der Spitze des Rotorblattes Wirbel, für deren Beschleunigung Energie benötigt wird und die damit den Wirkungsgrad des Rotors verringern. Diese Wirbel entstehen, weil Fluid von der Unterseite des Profils, die als Druckseite bezeichnet wird, weil ein höherer Druck als der Umgebungsdruck herrscht, an der Blattspitze zur Oberseite, der Saugseite, mit niedrigerem Druck strömt. Diesen Wirbeln wird mit dem Prandtl Tip Loss Faktor  $F$  Rechnung getragen. Ein weiterer Korrekturfaktor ist der Glauertfaktor.

Der Glauertfaktor wurde bei der experimentellen Untersuchung von Rotorblättern entwickelt und berücksichtigt, dass bei hohen Schnelllaufzahlen die schnell strömende Luft im Nachlauf Luft aus der Umgebung ansaugt. Dies widerspricht aber der grundsätzlichen Annahme, dass sich die einzelnen Elemente gegenseitig nicht beeinflussen und muss daher mit einem Korrekturfaktor abgebildet werden. Dieses Phänomen tritt ab einem  $a$  von 0,5 auf. Wird dieser Fehler in der Berechnung nicht korrigiert nimmt ab einem  $a$  von 0,4 der Schub mit weiter steigendem  $a$  ab. Ein steigendes  $a$  bedeutet, dass die axiale Geschwindigkeit der Abströmung bei gleichbleibender Anströmung steigt. Mit steigender Geschwindigkeitsdifferenz kann der Schub aber nicht abfallen. Glauert entwickelte daher mit Messpunkten eine Näherungsformel für  $C_T$ , die bei einem  $a$  von 0,4 die ursprüngliche  $C_T$ -Kurve tangiert.  $C_T$  ist dabei der Schubbeiwert, der die Schubkraft bezogen auf die Anströmgeschwindigkeit darstellt.

Da die von Glauert entwickelte Gleichung aber keine Rücksicht auf den Prandtl Tip Loss Faktor  $F$  nimmt, entsteht bei dessen Anwendung eine Sprungstelle. Buhl entwickelte daher eine Gleichung, die  $F$  berücksichtigt. Beide Formeln sind so ausgelegt, dass es bei einem  $a$  von 0,4 zu einem stetigen Übergang kommt.

*Abb. 2.9* zeigt die Verläufe dieser vier Kurven. Die klassische BEM Methode berechnet  $C_T$  mit :[12, 8]

$$C_T = 4a(1 - a) \quad (2.61)$$

Mit  $F$  wird die Gleichung erweitert zu:

$$C_T = 4Fa(1 - a) \quad (2.62)$$

Glauert entwickelte die Formel:

$$C_T = 0,889 - \frac{0,0203 - (a - 0,143)^2}{0,6427} \quad (2.63)$$

Formt man die von Buhl [8] entwickelte Gleichung:

$$C_T = \frac{8}{9} + (4F - \frac{40}{9})a + (\frac{50}{9} - 4F)a^2 \quad (2.64)$$

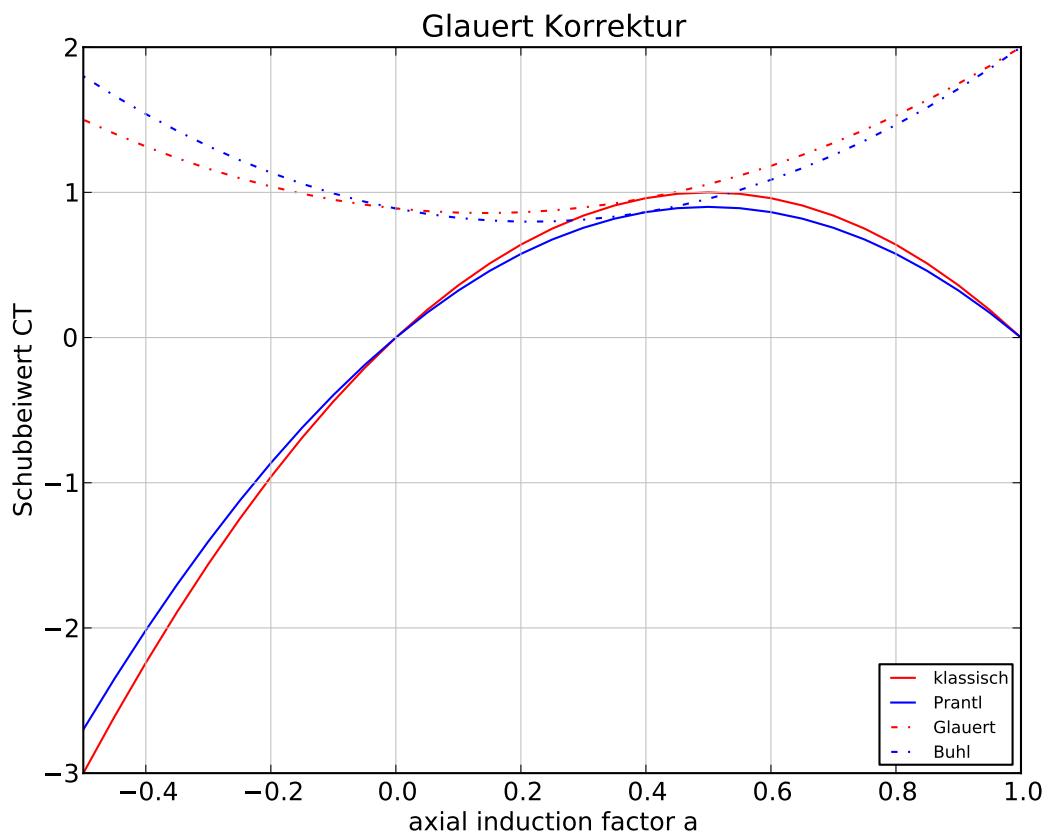
auf  $a$  um, so kommt man auf:

$$a = \frac{18F - 20 - 3\sqrt{C_T(50 - 36F) + 12F(3F - 4)}}{36F - 50} \quad (2.65)$$

Diese Gleichung ist in QBlade wiederzufinden. Sie kommt hier zur Anwendung, wenn  $C_T$ , das mit dem  $a$  der letzten Iteration gebildet wird, den Wert  $0,96F$  übersteigt.  $0,96$  ist jener Wert, den die Funktion  $C_T$  mit  $F = 1$  bei 0,4 annimmt. Also jener Wert, ab dem die von Glauert entwickelte Formel gültig sein sollte.

*Abb. 2.9* zeigt, dass die  $C_T$ -Funktion, auch ohne Korrektur, für negative  $a$  Werte streng monoton wachsend ist. Das bedeutet, dass bei einer betragsmäßig wachsenden Geschwindigkeitsdifferenz auch der Betrag der Schubkraft steigt.

Daher ist die Glauert Korrektur für Propeller irrelevant und es kann immer mit *Gl. (2.48)* gerechnet werden. Damit bleibt auch die Berechnung von Turbinen im Programm möglich.



**Abbildung 2.9:** Glauertkorrektur als Verlauf von  $C_T$  über  $a$



## Kapitel 3

# Praktische Anwendung

Das Programm PropCalc [13] ermöglicht einen raschen Überblick und ist leicht anzuwenden. Aber für eine genaue Abbildung des aktuellen Standes gibt es zu wenig Einstellmöglichkeiten und auch die Arbeitsweise des Programms kann nicht eingesehen werden. Auch die postprocessing Tools sind nicht ausreichend, um das Verbesserungspotential genauer zu analysieren. Da es aber rein auf Propeller ausgelegt ist, kann es als guter Anhaltspunkt für Vergleiche herangezogen werden.

Das Programm QBlade [10] ist eigentlich für Windturbinenentwicklung konzipiert, bietet aber umfangreiche Einstell- und Postprozessingmöglichkeiten. Es soll daher genauer untersucht und bei Bedarf abgeändert werden, um es für die Propellerberechnung zu verwenden.

Durch die Umwandlung in verschiedene Datenformate kommt es wiederholt zu Diskretisierungen der Geometrie, da jedes Datenformat die ursprüngliche Geometrie anders abbildet und abspeichert. Je nach Datenformat werden unterschiedliche Methoden zur Beschreibung der Geometrie mit unterschiedlichen Genauigkeiten verwendet. Mit jeder Umrechnung in ein anderes Datenformat kommt es damit zu steigender Ungenauigkeiten des Modells. Um diesen Fehlerquellen auszuweichen, sollte die Geometrie in möglichst direktem Arbeitsablauf, mit möglichst wenig unterschiedlichen Datenformaten, verarbeitet werden. Daher wurden für die Verarbeitung die Programme python und catia gewählt.

### 3.1 Vorgehensweise

Um mit der Auslegung des Propellers nicht ganz von Vorne anzufangen, soll ein bestehender Propeller simuliert und analysiert werden. Dazu wurden zwei bestehende Propeller mit einem Faro-Lasermessarm eingescannt. Aus den gescannten Daten wurde mit geomagic eine stp-Datei erzeugt. Diese Datei wurde mit Catia verarbeitet, um die Flächengeometrie in einzelne Schnitte zu unterteilen. Die Schnitte werden als Textdatei zwischengespeichert und mit python für die Verwendung in QBlade aufbereitet. Mit diesen Daten kann dann die Simulation in QBlade durchgeführt werden. Die Ergebnisse der Simulation werden als Textdateien abgespeichert und können mit python weiterverarbeitet werden. Der gesamte Arbeitsablauf ist in *Abb. 3.1* dargestellt.

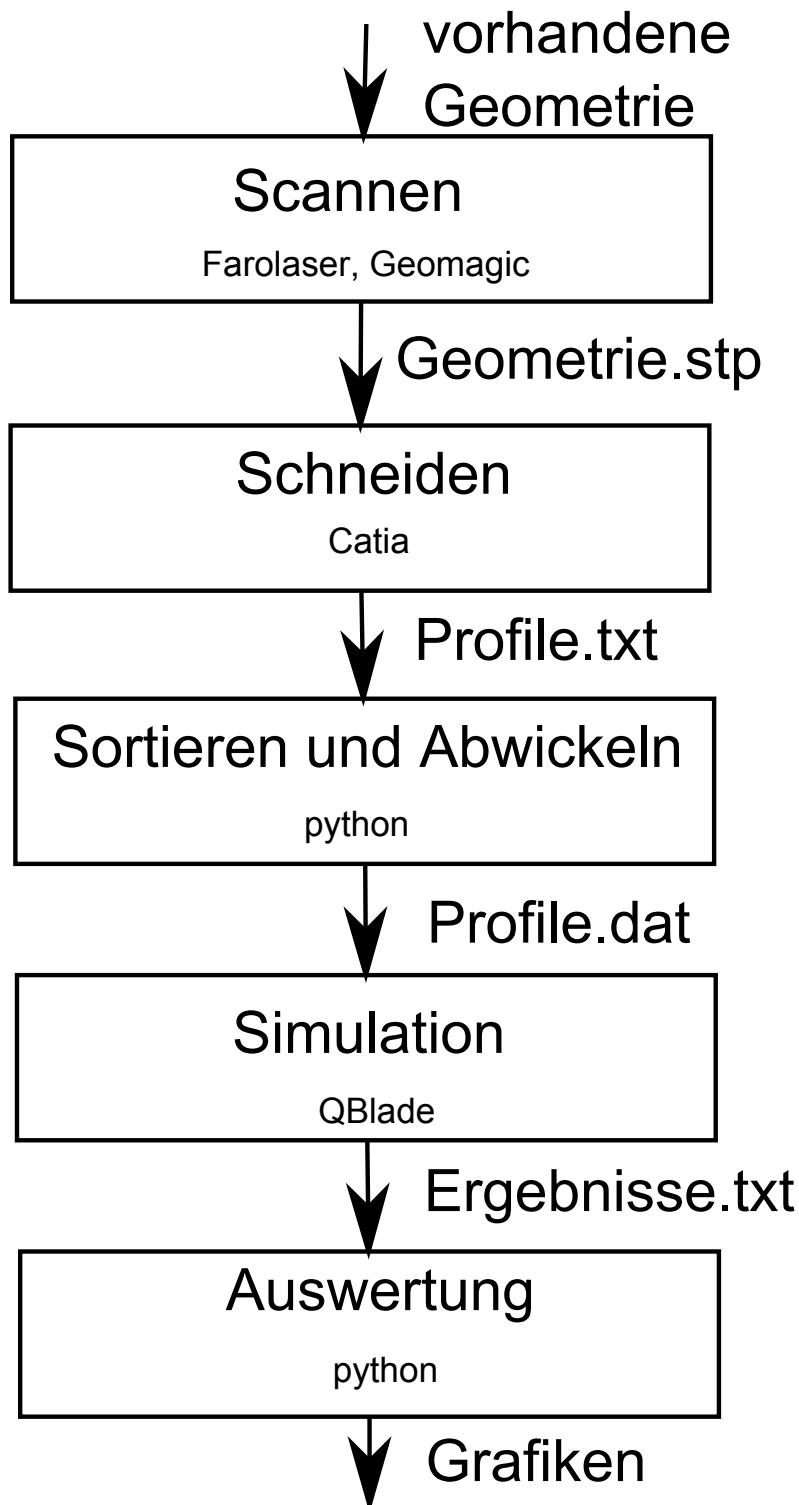


Abbildung 3.1: Grafische Darstellung des Arbeitsablaufes

## 3.2 Scannen der Geometrie

Als Ausgangsbasis sollen zwei als Hardware vorhandene Propeller, die sich im echten Flugbetrieb als geeignet herausgestellt haben, dienen, wie sie in Abb. 3.2 zu sehen sind.



**Abbildung 3.2:** Die beiden von Karl Waser zur Verfügung gestellten Propeller

Diese Propeller wurden an der HTL-Steyr mit der Hilfe von Herrn Georg Theuerkauf mittels eines 3D-Scanners digitalisiert und die daraus entstandenen Punktwolken trianguliert.

Bei dem 3D-Scanner handelt es sich um einen Faro-Laser mit Messarm. Da der 3D-Scanner die Oberfläche mit einem Punktscanner abtastet und dadurch das Messergebnis maßgeblich von der Reflexion der Oberfläche abhängt, wurde das Rotorblatt mit Kreidesstaub eingesprüht. Dadurch werden zwar einige Details wie Riefen oder Kratzer der Oberfläche verdeckt, allerdings sind diese Details Abweichungen von der Idealgeometrie, die eigentlich untersucht werden soll. Mit diesem Verfahren können die Flächen relativ gut abgebildet werden. Auch die Vorderkante kann durch ihren relativ großen Radius gut abgebildet werden. Die Hinterkante hingegen ist so scharf, dass das Licht zu sehr abgelenkt und zerstreut wird und die Kante so zum Teil falsch dargestellt wird. Abb. 3.3 zeigt den mit Kreidesstaub eingesprühten Propeller beim Einscannen.

Da eine optische Messung die Geometrie nie ganz erfassen kann, war es notwendig, den Propeller aus verschiedenen Perspektiven einzuscannen. Mit dem Programm geomagic wurden die einzelnen Scans zusammengesetzt und ein 3D-Netz der Oberfläche erstellt. Um aus den Scan-daten ein gutes Netz zu erstellen, sind einige Schritte notwendig. Unter anderem waren einige Artefakte, die offensichtlich aus Fehlern der Vermessung stammen, vorhanden. Diese konnten aber durch die gut abgebildeten Flächen der Umgebung gut ausgebessert werden.

In einer ersten Simulation wurde nur der für die Strömung wichtige äußere Teil der Rotorblätter eingescannt. Da aber ohne Referenzen eine Bestimmung der Lage nicht möglich ist und damit der Anströmwinkel der Profile nicht bekannt ist, wurde in einem zweiten Scannvorgang eine Welle in die Nabe eingesetzt und mit eingescannt. Diese Welle dient als Referenz und stellt die Rotationsachse dar.

Die Rotorblätter sind, wie in 2 beschrieben, beweglich an der Nabe angebracht und werden im Betrieb durch die Zentrifugalkraft nach außen gezogen und dort gehalten. Da es aber keinen Anschlag gibt, ist die exakte Lage der Rotorblätter nicht festgelegt. Um eine sinnvolle Ausgangslage zu haben, wurde angenommen, dass die Rotorblätter normal auf die Rotationsachse stehen.



**Abbildung 3.3:** mit Kreidestaub bedeckter Propeller

### 3.3 Verarbeitung der Punkte in Catia

Die von Geomagic erzeugte Geometrie kann im iges oder stp Format abgespeichert und in Catia V5 eingelesen werden.

Dort musste das Modell in das Koordinatensystem gedreht werden. An der Welle, die beim Einscannen als Orientierung mitgescannt wurde, sind zwei Kreise eingefügt worden, die durch 3 Punkte des Oberflächennetzes bestimmt sind. Die Rotationsachse wurde dann durch eine Linie, die genau durch die Mittelpunkte der beiden Kreise läuft, erzeugt. Mit einer Bedingung kann im Partdesign die Rotationsachse mit der z-Achse kongruent gesetzt werden. Danach diente ein ausgewählter Punkt an der Spitze des Rotorblattes als Orientierung und wurde mit der x-Achse kongruent gesetzt. Damit sind sowohl der Freiheitsgrad der Rotation als auch der Freiheitsgrad der Verschiebung entlang der Rotationsachse festgesetzt.

Da QBlade zum Erstellen der dreidimensionalen Geometrie einfach die ebenen Profile übereinanderstapelt würde sich die Methode mit ebenen Schnitten anbieten. Für die Simulation wurde jedoch eine Methode mit Zylinderschnitten, wie in *Kap. 2.3.2* beschrieben, gewählt. Dies führt dann zwar dazu, dass die Geometrie in QBlade optisch verzerrt wird, da die Profile vor den Stapeln nicht wieder aufgewickelt werden, aber in der Modellierung für die Berechnung von Schub und Antriebsmoment exakter ist. Beim Bauen eines Rotors kann man dann entweder die, mit zunehmendem Radius kleiner werdende, Ungenauigkeit akzeptieren oder die Profile in einem CAD-Programm auf Zylinder aufwickeln und dann erst stapeln.

In Catia wurde daher eine Zylindermantelfläche mit variablem Radius erstellt und dann mit dem 3D-Modell des Rotorblattes verschnitten. Dadurch entsteht dann eine 3D-Schnittkurve wie in Abb. *Kap. 2.6* in weiß zu sehen ist.

Auf dieser Kurve wurden dann, mit dem Befehl „Wiederholung der Punkterzeugung“, Punkte mit gleichem Abstand auf der Kurve erzeugt. Bei diesem Schritt ist zu bedenken, dass das in QBlade verwendete Xfoil maximal 300 Punkte pro Profil erlaubt. Da aber der erste Punkt des Profils auch der letzte sein muss, um ein geschlossenes Profil zu erzeugen, bleiben effektiv nur mehr 299 Punkte übrig. Um die Punkte weiterverarbeiten zu können, wurden die Punkte mit einem aus dem Internet stammenden Makro [3] in eine Textdatei geschrieben. Bei dieser etwas aufwändigen Prozedur müssen die zu exportierenden Punkte markiert werden bevor das Makro aktiviert wird. Die vom Makro erzeugte Datei wurde umbenannt, um beim nächsten Durchlauf des Makros nicht wieder überschrieben zu werden. Die Modellierung des Zylinders beinhaltete einen Parameter für den Radius, damit dieser einfach im Strukturaum geändert werden kann. Um eine ausreichend genaue Analyse durchführen zu können, wurde ab einem Startradius von 40 mm alle 5 mm ein Schnitt erzeugt. Der Name des Ordners, in dem die Textdateien gespeichert werden, wird im nachfolgenden Pythonskript als Name des Modells erkannt und verwendet.

### 3.4 Verarbeitung mit python

Die in Catia erstellte Textdatei enthält alle drei Koordinaten der zu einem Profil gehörenden Punkte. Die Punkte sind zwar in den meisten Fällen in einer logischen Reihenfolge, aber der Startpunkt ist ein zufälliger Punkt auf der Kurve.

Um ein Profil in Xfoil einlesen zu können, müssen die Punkte aber richtig formatiert sein. Um diese Differenzen zu überwinden, wurde in der Programmiersprache Python [4] ein Skript erstellt, das die Daten der Punkte verarbeiten und richtig ausgeben soll.

Laut Xfoil Dokumentation [9] muss der erste Punkt in der Datei die Hinterkante des Profils sein. Danach sollten die Punkte der Oberseite der Reihe nach von der Hinterkante bis zur Vorderkante aufgelistet werden. Anschließend die Punkte der Unterseite von vorne wieder zurück zur Hinterkante. Als letzter Punkt muss wieder die Hinterkante eingetragen sein. So entsteht eine, wie in Abb. 3.4 gezeigte, geschlossene Kurve, die das Profil einschließt. Prinzipiell ist es auch möglich zuerst die Unterseite zu schreiben und damit das Profil in der anderen Richtung zu beschreiben. Allerdings können dann in Xfoil die Skeletlinie und andere Geometriekennwerte nicht berechnet werden. Über den Befehl „refine globaly“ in QBlade, kann man dann wieder ein ordnungsgemäßes Profil erzeugen. Dieses ist dann aber nur eine Annäherung an das ursprüngliche Profil und daher nicht zu empfehlen. Im Skript wird die Aufgabe des Sortierens derart gelöst, dass

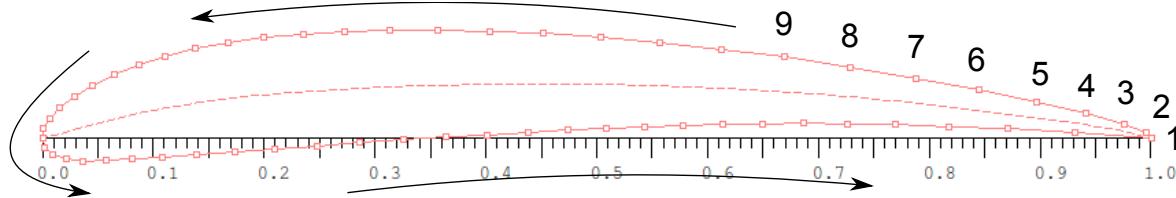


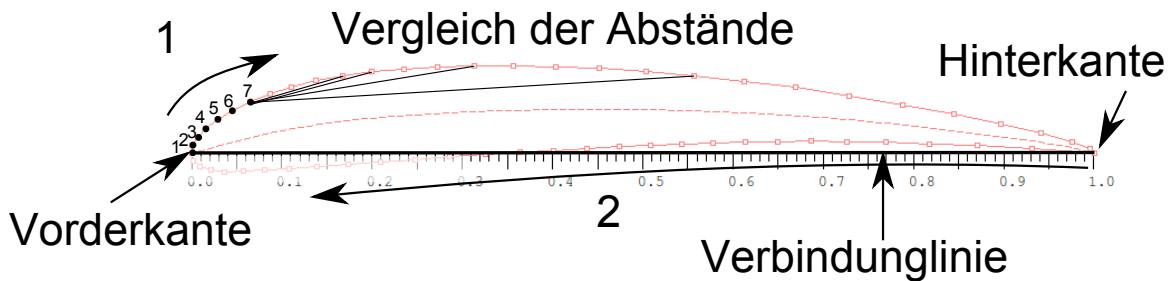
Abbildung 3.4: Reihenfolge der Punkte eines Xfoil Profils

zuerst die Vorder- und Hinterkante gesucht werden. Bei den Profilen kann man davon ausgehen, dass diese beiden Punkte den größten Abstand zueinander haben. Im Skript catiasort.py wird diese Aufgabe dadurch erfüllt, dass eine Schleife jeden Punkt des Profils einmal als Punkt a bezeichnet. Eine verschachtelte Schleife markiert zusätzlich jeden Punkt einmal als Punkt b und berechnet den Abstand zwischen den beiden Punkten mit dem Satz des Pythagoras und den Differenzen der Koordinaten:

$$\text{Abstand} = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2} \quad (3.1)$$

Dieser Abstand wird mit der Variablen „dis“, in der der bisher größte Abstand gespeichert ist, verglichen. Ist der Abstand größer, werden die beiden Punkte gespeichert und der Abstand in die Variable „dis“ übernommen, um für die darauffolgenden Vergleiche als Referenz zu dienen. Durch die beiden Schleifen wird jeder Punkt einmal als a Punkt und pro anderen Punkt als b Punkt zur Berechnung des Abstands eingesetzt. Am Ende der beiden Schleifen sind die beiden Punkte markiert die den größten Abstand zueinander haben. Unter der Annahme, dass das Profil ungefähr so orientiert ist dass die Anströmung in Richtung der positiven x-Achse liegt wird der Punkt mit der kleineren x-Koordinate als Vorderkante interpretiert. Der verbleibende Punkt ist demnach die Hinterkante. Durch diese beiden Punkte wird dann eine Gerade gelegt, die die Punktwolke in zwei Teile teilt. Durch die Krümmung der Profile liegen im oberen Teil etwas mehr

Punkte. Dies ist in Abb. 3.5 dargestellt. Außerdem kommt es bei besonders stark gekrümmten Profilen vor, dass Punkte der Unterseite über die Trennlinie in den oberen Teil ragen. In der Nähe der Vorderkante ist es eher unwahrscheinlich, dass sowohl Punkte der Ober- als auch der Unterseite im Teil über der Geraden liegen. Daher wird von der Vorderkante ausgehend der obere Teil der Punkte sortiert bis die Hinterkante erreicht wird. Zum Sortieren wird, ausgehend vom Startpunkt, für jeden Punkt überprüft ob der Abstand zwischen Punkt und Startpunkt errechnet und dann verglichen ob es bisher schon einen anderen Punkt gab dessen Abstand geringer war. Ist dies nicht der Fall wird der Punkt zwischengespeichert und sein Abstand dient für die folgenden Punkte als Vergleich. Bei ausreichender Auflösung des hinteren Teils des Profils kann dieser auch sortiert werden, ohne dass die Kontur zwischen Ober- und Unterseite hin- und herspringt, da die Abstände zwischen den Punkten einer Seite geringer sind als der Abstand zwischen den Seiten. Bei Erreichen der Hinterkante können alle verbleibenden Punkte der Unterseite zugeordnet und dementsprechend sortiert werden. Um die von Xfoil geforderte Anordnung zu erreichen, werden Unter- und Oberseite zusammengesetzt und in umgekehrter Reihenfolge ausgegeben. Neben dem Sortieren ist auch das Abwickeln eine Hauptaufgabe des



**Abbildung 3.5:** Sortierung der Punkte eines Xfoil Profils (suchen des 8. Punktes)

Skripts. Wie im vorherigen Kapitel erwähnt, wird der Propeller mit Zylindern geschnitten, um die Profile zu erzeugen. Dadurch entstehen dreidimensionale Kurven. Xfoil berechnet aber ein ebenes Problem. Das Python Skript berechnet jetzt für jeden Punkt den Winkel zwischen dem Ortsvektor jedes Punktes und der Normalen auf die Bildebene wie in Abb. 3.6 dargestellt ist. Mit diesem Winkel  $\gamma$  und dem Radius des Schnittzylinders kann dann die Bogenlänge errechnet werden, die dann dem horizontalen Abstand des abgewickelten Punktes entspricht.

$$\gamma = \text{atan} \left( \frac{y - \text{Koordinate}}{x - \text{Koordinate}} \right) \quad (3.2)$$

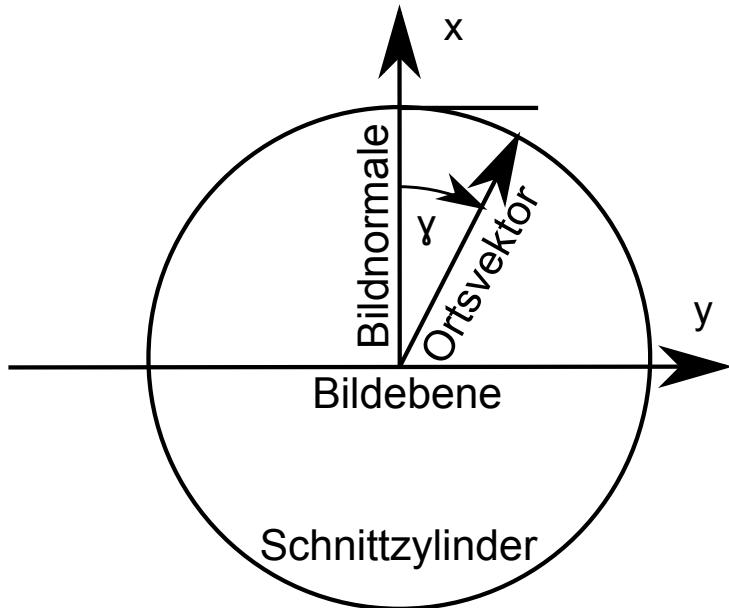
$$\text{Radius} = \sqrt{(y - \text{Koordinate})^2 + (x - \text{Koordinate})^2} \quad (3.3)$$

$$\text{Bogenlänge} = \gamma \text{Radius} \quad (3.4)$$

Setzt man Gl. (3.2) und Gl. (3.3) in Gl. (3.4) ein erhält man:

$$\text{Bogenlänge} = \text{atan} \left( \frac{y - \text{Koordinate}}{x - \text{Koordinate}} \right) \sqrt{(y - \text{Koordinate})^2 + (x - \text{Koordinate})^2} \quad (3.5)$$

Um die Rechnungen von Xfoil einfacher und vergleichbar zu machen, werden alle Profile so ausgerichtet, dass die Profil sehne auf der horizontalen Achse liegt und auf die Länge 1 normiert. Damit die Profile wieder zu einem Propeller zusammengesetzt werden können müssen die



**Abbildung 3.6:** Systematische Darstellung der Abwicklung

Länge und der Anstellwinkel des Profils verfügbar bleiben. Zu diesem Zweck wird neben den Geometriedateien noch eine .log Datei erzeugt, die für jedes Profil den Radius, die Länge und den Neigungswinkel  $\Theta$  angibt. Weiters werden aus den Eingaben strömungsrelevante Daten wie Reynoldszahl und Machzahl berechnet und angegeben. Die Reynoldszahl wird dabei gemäß der Formel  $Re = \frac{c * L}{\nu}$  mit der relativen Anströmgeschwindigkeit  $c$ , der charakteristischen Länge  $L$  und der kinematischen Viskosität  $\nu$  berechnet. Für die Machzahl wird die relative Anströmgeschwindigkeit durch die Schallgeschwindigkeit, die mit 343m/s angenommen wird, dividiert.

Die einzelnen Schritte und deren Abfolge sind als Kommentar im Quelltext beschrieben. Der Quelltext ist unter dem Namen catiasort.py im Anhang B zu finden. Jeder Schnitt wird im Ausgabeordner in einer eigenen Datei gespeichert. Der Name der Datei gibt dabei den Radius des schneidenden Zylinders in mm an. Die Zylinderachse ist gleichzeitig die Rotationsachse des Propellers.

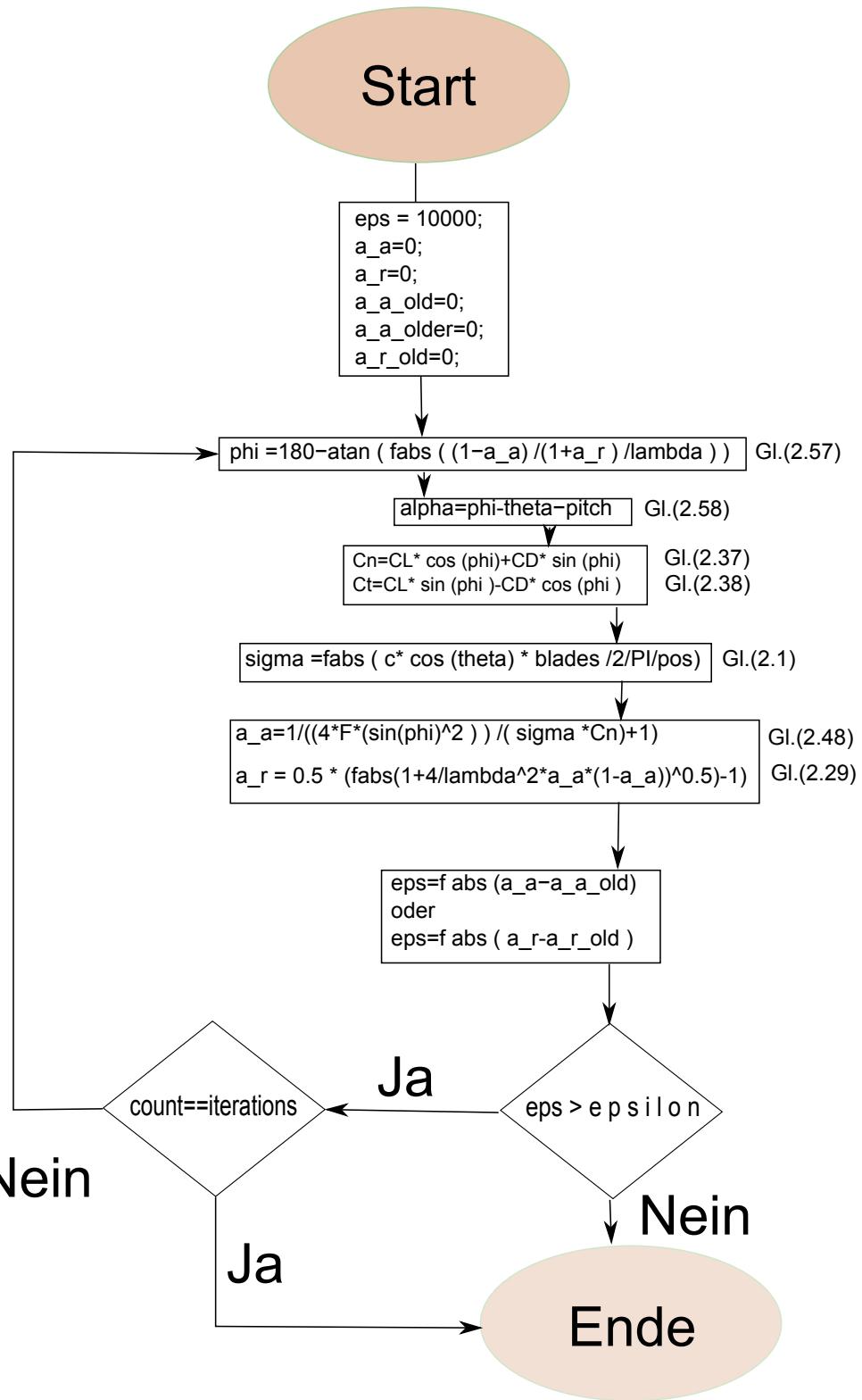
## 3.5 QBlade

### 3.5.1 Allgemeines

QBlade ist ein an der TU Berlin entwickeltes Programm zur Simulation und Auslegung von Windkraftanlagen. Zur Zeit unterstützt es die Berechnung von horizontal axis wind turbines (HWAT) und vertical axis wind turbines (VWAT). Für die Berechnung wird die im *Kap. 2.4* beschriebene BEM Methode angewandt, deren Formelapparat für die Berechnung von Turbinen ausgelegt ist. Für die Berechnung von  $C_L$  und  $C_D$  wurde das Programm XFOIL implementiert. QBlade bietet außerdem eine gut zu handhabende Benutzeroberfläche und vielfältige postprocessing Möglichkeiten, wodurch es vor allem für die Auslegung und Entwicklung interessant ist. Beide Programme sind open source und können daher genau untersucht und geändert werden. Dies ist auch notwendig, da die QBladeversion v0.6-r68 nur die Berechnung von Turbinen, nicht aber von Propellern ermöglicht. Prinzipiell beruhen BEM Rechnungen für Propeller und Turbinen auf den gleichen physikalischen Annahmen und Vereinfachungen, unabhängig davon, in welche Richtung die äußeren Kräfte und Momente wirken. Damit erfüllt QBlade die Grundvoraussetzungen, um auch Propeller rechnen zu können. Da sich die Gleichungen und die geometrischen Beziehungen für die Berechnung von Turbinen aber doch von denen der Propeller unterscheiden ist es notwendig, diese geeignet anzupassen.

### 3.5.2 Quellcode der BEM Simulation

Die Umstellung einer Turbine zu einem Propeller kann durch Umdrehen der Drehrichtung des Rotors erfolgen. Damit ergibt sich ein unterschiedliches Geschwindigkeitsdreieck bei der Anströmung, wie in *Abb. 2.8* zu sehen ist. Daher muss es in der Benutzeroberfläche des Programmes möglich sein, negative Drehzahlen einzugeben. Bisher war diese Möglichkeit unterbunden, um fehlerhafte Ergebnisse zu vermeiden. Durch Deaktivieren der Zeilen 122, 127, 151, 154, 157, 190, 193, 196, 200 und 203 der Datei „SimuWidget.cpp“ im Ordner „XBEM“ ist es nun möglich, negative Drehzahlen, Windgeschwindigkeiten und Schnelllaufzahlen einzugeben. Von der Möglichkeit, negative Windgeschwindigkeiten einzugeben, wurde in einer ersten Simulation Gebrauch gemacht, sollte aber nicht weiter verwendet werden da die nachfolgenden Änderungen für diesen Fall nicht überprüft wurden. Die wichtigsten Berechnungsschritte der BEM Theorie erfolgen in der Datei BData.cpp, die sich ebenfalls im Ordner „XBEM“ befindet. Hier ist besonders die Funktion „OnBEM“ interessant und soll im Folgenden genauer diskutiert werden. Die grundsätzlichen Abläufe sind in *Abb. 3.7* dargestellt. Zuerst werden die Iterationsparameter initialisiert. epsilon ist die Abbruchbedingung, die durch einen in der Benutzeroberfläche eingestellten Wert festgelegt wird. Die Differenz der Induktionsfaktoren zwischen zwei Iterationen wird als eps bezeichnet und muss den durch epsilon festgelegten Wert unterschreiten damit die Rechnung als ausreichend genau angesehen und abgebrochen werden wird. Für die erste Iteration wird eps auf einen sehr großen Wert gesetzt, der sicher nicht die Abbruchbedingung erfüllt. Für die Initialisierung werden die axialen und tangentialen Induktionsfaktoren für die aktuelle und die beiden vorangegangenen Iterationen mit 0 angenommen. Dies würde bedeuten, dass keine Beeinflussung des Fluidstroms durch den Rotor erfolgt. Nach der Initialisierung beginnt der erste Schritt der Iteration. Der Anströmwinkel phi wird berechnet. In *Kap. 2.4* entspricht der Winkel  $\Phi$  der Programmvariablen phi. Die Berechnung entspricht *Gl. (2.57)*. Daraus kann

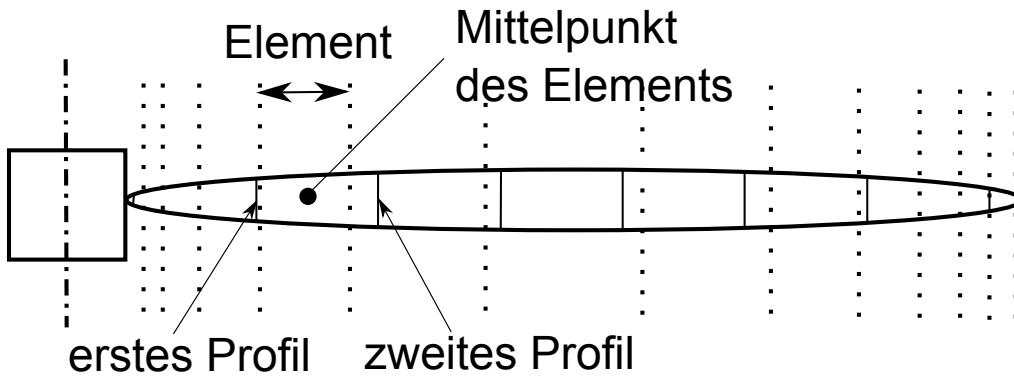


**Abbildung 3.7:** Prinzipieller Ablauf der OnBEM Funktion für Propeller

der Winkel alpha berechnet werden. Die Berechnung von alpha entspricht Gl. (2.58), wobei in

QBlade  $\Theta$  in einen Neigungswinkel des Profils innerhalb des Rotorblattes ( $\theta$ ) und einen Neigungswinkel des gesamten Rotorblattes (pitch) unterteilt. Mit dem Wert für alpha können die entsprechenden Werte für die Auftriebs und Widerstandskoeffizienten nachgeschlagen werden. Die Berechnung der für den Schub beziehungsweise für das Moment zuständigen Komponenten der Auftriebs und Widerstandskoeffizienten erfolgt wie in Gl. (2.37) und Gl. (2.38). Die Berechnung von sigma entspricht Gl. (2.1). Da jetzt alle Werte für Gl. (2.48) und Gl. (2.29) bekannt sind können der axiale und tangentiale induction factor berechnet werden. eps wird aus der Differenz der Faktoren berechnet. Wobei jener Faktor gewählt wird bei dem die größere Differenz zwischen der aktuellen und der letzten Iteration besteht. Damit wird zuerst geprüft ob die Abweichung eps kleiner ist als die maximal zulässige Differenz epsilon und damit die erste Abbruchbedingung erfüllt ist. Danach wird geprüft ob die Anzahl der Iterationen die maximal zulässige Anzahl übersteigt. Ist dies der Fall ist die zweite Abbruchbedingung erfüllt und die Iteration wird abgebrochen. Ist dies nicht der Fall wird mit axial und tangential induction factor ein neues phi errechnet und die Iteration wird erneut durchgeführt. Der Großteil des Codes kann einfach übernommen werden. Die veränderten Stellen wurden im Quellcode mit dem Kürzel CW versehen.

In Listing (3.1) werden zuerst die verwendeten Variablen deklariert. In  $a\_a$ ,  $a\_a\_old$  und  $a\_a\_older$  werden die axial induction factors der aktuelle und der beiden vorangehenden Iterationen gespeichert. In  $a\_r$ ,  $a\_r\_old$  werden die tangential induction factors der aktuellen und vorangehenden Iterationen gespeichert. phi entspricht dem Anströmwinkel  $\Phi$  und alpha den Angriffswinkel  $\alpha$ . Da die Diskretisierung in die verschiedenen Elemente nur in Ausnahmefällen dazu führt, dass ein Element von zwei bekannten Profilen begrenzt wird, liegt das Element irgendwo zwischen den bekannten Profilen. Wurde die Option „foil interpolation“ nicht aktiviert, wird angenommen, dass das gesamte Element den Querschnitt jenes Profils besitzt, das am nächsten zum Mittelpunkt des Elements liegt. In dem in Abb. 3.8 dargestellten Beispiel würde für das gesamte Element das erste Profil angenommen. Wurde die Option „foil interpolation“ jedoch aktiviert, so wird zwischen den beiden Profilen, zwischen denen der Mittelpunkt des Elements liegt, linear interpoliert. In dem in Abb. 3.8 dargestellten Beispiel würde zwischen dem ersten und zweiten Profil linear interpoliert. Daher werden für die Auftriebs und Widerstandsbeiwerte



**Abbildung 3.8:** Diskretisierung des Rotors

der beiden angrenzenden Profilen die Variablen CL, CD, CL2 und CD2 angelegt. Zum Interpolieren zwischen den beiden Profilen wird der Unterschied der Angriffswinkel benötigt. Für diesen wird die Variable DeltaAlpha deklariert. Wenn nicht Interpoliert wird, wird in die Varia-

ble Reynolds die Reynoldszahl des Profils geschrieben. Wird Interpoliert wird die interpolierte Reynoldszahl in die Variable localReynolds geschrieben. Für das gesamte Element werden die Variablen für den Koeffizienten der Normalkraft, den Koeffizienten der Tangentialkraft und den Schubbeiwert die Variablen Cn, Ct und CT deklariert. Für Rotorsteifigkeit aus Gl. (2.1) wird die Variablen sigma deklariert. Für den Prandtl Tip Loss Faktor wird die Variable F und für den New Prandtl Tip Loss die Variable Fl deklariert. Die Variable eps dient der Ermittlung der Abbruchbedingung und i ist eine Laufvariable für Schleifen.

```

167 void BData::OnBEM( double pitch )
168 {
169
170
171 double a_a, a_a_old, a_a_older ;
172 double a_r, a_r_old ;
173 double phi, alpha, F1;
174 double eps, CL, CD, Cn, Ct, CL2, CD2, CT;
175 double DeltaAlpha, sigma, localReynolds;
176 int i;
177 double F, Reynolds;
178
179 windspeedStr . sprintf ( "% . 2 f " , windspeed );
180
181 m_PolarPointers . clear ();
182
183
184 //every element gets a pointer assigned to the polar at the section before (
185 // m_PolarPointers )
186 //the elements center and after the elements center ( m_PolarPointersTO )
187 for ( int i = 0 ; i < m_polar . size () ; i ++ )
188 {
189     m_PolarPointers . append ( Get360Polar ( m_foil . at ( i ) , m_polar . at ( i ) ) );
190     m_PolarPointersTO . append ( Get360Polar ( m_foilTO . at ( i ) , m_polarTO . at ( i ) ) );
191 }
```

**Listing 3.1:** Deklarieren der Variablen

Wie bereits erwähnt, wird der Rotor in einzelne Teilesegmente aufgeteilt, die sich gegenseitig nicht beeinflussen können. In der Funktion „init“ dieser Datei wird die Geometrie mit einer Sinusfunktion diskretisiert, sodass die Abstände gemäß einer Sinusfunktion im Bereich zwischen 0 und  $\pi$  am Anfang klein sind bis zu einem Maximum anwachsen und dann wieder abnehmen. Dadurch ist im Bereich der Nabe und der Spitze die Diskretisierung enger als in der Mitte des Rotorblattes. Die entsprechende Formulierung des Quellcodes kann in Anhang A ab Zeile 100 nachgelesen werden. Für jedes Element werden wichtige Größen berechnet und in ein array geschrieben. Zu diesen Größen gehört auch der Neigungswinkel Theta des jeweiligen Profils. Da für jedes Element ein eigenes Theta berechnet wurde, entspricht die Länge des arrays „m\_theta“ genau der Anzahl der Elemente. Die for-Schleife in Zeile 193 startet also für jedes Element eine neue Berechnung der Strömungsdaten. Die Variablen a\_a und a\_r stehen für die Induktionsfaktoren a und a' und können nur iterativ berechnet werden, da sie von den durch die Strömung induzierten Kräften abhängig sind. Diese wiederum von der Anströmgeschwindigkeit der Profile und die Anströmgeschwindigkeit ist von den Induktionsfaktoren a und a' abhängig.

Daher werden die Induktionsfaktoren  $a$  und  $a'$  erstmal zu Null gesetzt, sodass sie keinen Einfluss auf die erste Berechnung haben.

```

192 //now the first loop starts over all elements
193 for ( i=0;i<m_theta.size(); i++)
194 {
195     //the induction factors and epsilon are initialized
196     eps = 10000;
197     a_a=0;
198     a_r=0;
199     a_a_old=0;
200     a_a_older=0;
201     a_r_old=0;
```

**Listing 3.2:** Initialisieren der Variablen

In *Listing (3.3)* werden die beiden Abbruchbedingungen für die Iteration abgefragt. Die maximale Anzahl der Iterationen kann in der Benutzeroberfläche beim Starten einer BEM Rechnung eingegeben werden und soll bei problematischen Rechnungen die Rechenzeit limitieren. Die zweite Abbruchbedingung wird durch epsilon erstellt. Am Ende dieser Funktion werden wir sehen, dass  $\text{eps}$  die Differenz zwischen dem aktuellen und den vorangegangenen Induktionsfaktoren ist und damit ein Maß dafür ist, wie nahe die Iteration schon am endgültigen Ergebnis ist. Die while-Schleife iteriert also so lange, bis eine der beiden Bedingungen zum Abbruch führt.

```

204 //the counter for the number of iterations is set to zero
205 int count =0;
206
207 //this is the criterion for an iteration to converge
208 while (eps > epsilon)
209 {
210
211     //when the maximum number of iterations is reached the iterations are stopped
212     count++;
213     if (count == iterations)
214     {
215         break ;
216     }
```

**Listing 3.3:** Anfragen der Abbruchbedingungen und Starten der Iteration

Der in *Listing (3.4)* berechnete Anströmwinkel  $\phi$  ist in erster Linie von der Umfangsgeschwindigkeit des jeweiligen Bereiches und von der Anströmgeschwindigkeit in axialer Richtung, die näherungsweise der Fluggeschwindigkeit gleichgesetzt werden kann, abhängig. Die beiden Geschwindigkeiten werden in der Schnelllaufzahl aus *Gl. (2.26)* zusammengefasst. Die ursprüngliche Formel, die in Zeile 222 steht, wurde mit Geschwindigkeiten und Winkeln, wie sie für Turbinen gelten, erstellt. Sie soll auch weiterhin für Turbinen, die sich an einem positiven Lambda erkennen lassen, angewandt werden. Aber wie bereits anhand *Abb. 2.8* beschrieben, weist die Drehrichtung von Propellern in die andere Richtung, weshalb auch das Lambda negativ wird. Die Formel muss also wie in Zeile 226 angepasst werden.

```

218     //the angle phi is computed
219 // new code CW
220     if ( m_lambda_local.at ( i )>=0)
```

```

221     {
222         phi = atan( (1-a_a)/(1+a_r) / m_lambda_local.at(i) )/2/PI*360;
223     }
224     else
225     {
226         phi =180-atan( fabs( (1-a_a)/(1+a_r) / m_lambda_local.at(i) ))/2/PI
227             *360;
228     }
229 //end new code CW

```

**Listing 3.4:** Berechnung des Anströmwinkels

Bei einer Auftrieb erzeugenden Strömung ist sowohl für den Propeller als auch für die Turbine  $\Phi$  größer als  $\Theta$  und deren Differenz ergibt den Winkel  $\alpha$ . Die Formel in *Listing (3.5)* kann also unverändert bleiben und entspricht *Gl. (2.31)*. Sollten noch besonders unübliche Winkel auftreten wird außerdem  $\alpha$  wieder in den Bereich von  $-180^\circ$  bis  $180^\circ$  gedreht, um  $C_L$  und  $C_D$  Werte auslesen zu können.

```

229     //alpha is computed
230     alpha=phi-m_theta.at(i)-pitch;
231
232     while (alpha < -180) alpha+=360;
233     while (alpha > 180) alpha-=360;

```

**Listing 3.5:** Berechnung des Angriffswinkels

Das Einlesen der Per Polare der angrenzenden Profile ist rein von der Geometrie abhängig und damit für Turbine und Propeller gleich. Die Berechnung von  $\alpha$  ist bereits an die beiden Fälle angepasst und jeweils auf die chordline bezogen. Außerdem wird nur dann eine zweite Polare eingelesen und interpoliert, wenn die Option „foil interpolation“ in der Benutzeroberfläche aktiviert wurde.

```

237     bool found=false;
238     bool found2=false;
239
240     //here the polar data from the polar at the last section is extracted
241     for (int j=0; j<m_PolarPointers.at(i)->m_Alpha.size(); j++)
242     {
243
244         if (m_PolarPointers.at(i)->m_Alpha.at(j) >= alpha && !found)
245         {
246
247
248             DeltaAlpha = m_PolarPointers.at(i)->m_Alpha.at(j)-m_PolarPointers.
249                         at(i)->m_Alpha.at(j-1);
250
251
252             CL = m_PolarPointers.at(i)->m_Cl.at(j-1)+(m_PolarPointers.at(i)->
253                 m_Cl.at(j)-m_PolarPointers.at(i)->m_Cl.at(j-1))/DeltaAlpha*(
254                 alpha-m_PolarPointers.at(i)->m_Alpha.at(j-1));
255             CD = m_PolarPointers.at(i)->m_Cd.at(j-1)+(m_PolarPointers.at(i)->
256                 m_Cd.at(j)-m_PolarPointers.at(i)->m_Cd.at(j-1))/DeltaAlpha*(
257                 alpha-m_PolarPointers.at(i)->m_Alpha.at(j-1));
258
259             if (m_bCdReynolds)

```

```

255         {
256             localReynolds = pow(pow(windspeed,2)+pow(m_lambda_local.at(i)*
257                                 windspeed,2),0.5)*m_c_local.at(i)/visc*rho;
258             CD = CD * pow(m_PolarPointers.at(i)->m_Reynolds/localReynolds
259                                 ,0.2);
260         }
261
262         Reynolds = m_PolarPointers.at(i)->m_Reynolds;
263
264         //implementation the 3D correction after SNEL
265         if (m_b3DCorrection && m_pos.at(i) <= outer_radius
266             *0.8 && alpha >= m_PolarPointers.at(i)->
267             m_ACrit)
268         {
269             qDebug() << m_PolarPointers.at(i)->m_ACrit
270             << m_PolarPointers.at(i)->m_FoilName << m_PolarPointers.at(i)->m_ASpec;
271
272             double blend = 0;
273             if (alpha < 30 && alpha >= m_PolarPointers
274                 .at(i)->m_ACrit) blend = 1;
275             else if (alpha <= 50 && alpha >
276                     m_PolarPointers.at(i)->m_ACrit) blend
277                     = 1-(alpha-30)/20;
278
279             if ((2*PI*(alpha-m_PolarPointers.at(i)->
280                     m_ACrit)/360*2*PI-CL)>0)
281                 CL = CL + (double(3.1)*pow(m_lambda_local.
282                     at(i),2))/(double(1)+pow(
283                         m_lambda_local.at(i),2))*blend*pow((
284                             m_c_local.at(i)/m_pos.at(i)),2)*(
285                             m_PolarPointers.at(i)->m_ASpec*(alpha-
286                             m_PolarPointers.at(i)->m_ACrit)-CL);
287         }
288         found = true;
289     }
290
291     }
292
293     //here the polar data from the polar at the next section is extracted
294     if (m_bInterpolation && m_PolarPointers.at(i)->m_PlName !=
295         m_PolarPointersTO.at(i)->m_PlName)
296     {
297         for (int k=0; k< m_PolarPointersTO.at(i)->m_Alpha.size();k++)
298         {
299
300             if (m_PolarPointersTO.at(i)->m_Alpha.at(k) >= alpha && !found2)
301             {
302
303                 DeltaAlpha = m_PolarPointersTO.at(i)->m_Alpha.at(k)-
304                     m_PolarPointersTO.at(i)->m_Alpha.at(k-1);
305
306                 CL2 = m_PolarPointersTO.at(i)->m_Cl.at(k-1)+(m_PolarPointersTO
307                     .at(i)->m_Cl.at(k)-m_PolarPointersTO.at(i)->m_Cl.at(k-1))/
```

```

293                               DeltaAlpha*(alpha-m_PolarPointersTO . at ( i )->m_Alpha . at ( k-1 )
294                               );
295 CD2 = m_PolarPointersTO . at ( i )->m_Cd . at ( k-1 )+(m_PolarPointersTO
296 . at ( i )->m_Cd . at ( k )-m_PolarPointersTO . at ( i )->m_Cd . at ( k-1 ))/
297 DeltaAlpha*(alpha-m_PolarPointersTO . at ( i )->m_Alpha . at ( k-1 )
298 );
299
300
301
302 //implementation the 3D correction after SNEL
303 if (m_b3DCorrection && m_pos . at ( i ) <=
304         outer_radius*0.8 && alpha >=
305         m_PolarPointers . at ( i )->m_ACrit)
306 {
307     qDebug () << m_PolarPointers . at ( i )
308     ->m_ACrit << m_PolarPointers . at ( i )->m_FoilName << m_PolarPointers . at ( i
309     )->m_ASpec;
310
311     double blend = 0;
312     if (alpha < 30 && alpha >=
313         m_PolarPointers . at ( i )->m_ACrit
314         ) blend = 1;
315     else if (alpha <= 50 && alpha >
316         m_PolarPointers . at ( i )->m_ACrit
317         ) blend = 1-(alpha-30)/20;
318
319     if (((2*PI*(alpha-m_PolarPointers .
320             at ( i )->m_ACrit)/360*2*PI-CL)
321             >0)
322             CL = CL + (double(3.1)*pow(
323                 m_lambda_local . at ( i ),2))/(
324                 double(1)+pow(m_lambda_local .
325                     at ( i ),2))*blend*pow((m_c_local
326                     . at ( i )/m_pos . at ( i )),2)*(
327                         m_PolarPointers . at ( i )->m_ASpec
328                         *(alpha-m_PolarPointers . at ( i )
329                         ->m_ACrit)-CL);
330
331     }
332 //here the data is interpolated between the two polars if foil
333     interpolation is on
334 Reynolds = m_PolarPointers . at ( i )->m_Reynolds*m_between . at ( i )
335     +(1-m_between . at ( i ))*m_PolarPointersTO . at ( i )->m_Reynolds;
336 double newCL = CL*m_between . at ( i )+(1.0-m_between . at ( i ))*CL2;
337 double newCD = CD*m_between . at ( i )+(1.0-m_between . at ( i ))*CD2;
338
339 CL = newCL;
340 CD = newCD;
341 found2 = true;
342 }
```

```

323 }
324
325
326
327
328 }
```

**Listing 3.6:** Einlesen der Polare der angrenzenden Profile

Die Faktoren CL und CD werden von xfoil bereitgestellt und geben den Auftrieb und den Widerstand bezogen auf die Strömungsrichtung an. Für die Antriebsleistung und den Schub sind jedoch die Kräfte in tangentialer und normaler Richtung zu verwenden. Die Umrechnung ist rein von der Geometrie abhängig und erfolgt entsprechend *Gl. (2.37)* und *Gl. (2.38)*.

```

330 // computation of normal and thrust coefficient
331 Cn=CL*cos(phi/360*2*PI)+CD*sin(phi/360*2*PI);
332 Ct=CL*sin(phi/360*2*PI)-CD*cos(phi/360*2*PI);
```

**Listing 3.7:** Berechnen der Koeffizienten der Normal- und Tangentialkraft

Die Rotorsteifigkeit (solidiy) wird, entgegen der Definition von *Gl. (2.1)*, mit der in der Rotationsebene gemessenen Profillänge berechnet. Bei Propellern ist  $\Theta$  größer als  $90^\circ$  und würde unzulässig negative Sigmas ergeben. Daher wurde in *Listing (3.8)* ein Betrag eingeführt.

```

336 // computation of solidity
337
338 sigma =fabs( m_c_local.at(i)*cos(m_theta.at(i)/360*2*PI)*blades/2/PI/m_pos
339 .at(i)); //CW
340
341 //the old induction factors are stored to compute the convergence
342 //criterion
342 a_a_older=a_a_old;
343 a_a_old=a_a;
344 a_r_old=a_r;
```

**Listing 3.8:** Berechnen der Rotorsteifigkeit

Wie bereits im *Kap. 2.4* beschrieben sind einige Faktoren welche die Strömung beeinflussen in den Grundgleichungen der BEM Methode nicht berücksichtigt und die entsprechenden Faktoren werden hier eingefügt. Bis auf den New Tip und Root Loss weisen die Faktoren aber keine Abhängigkeit von der Drehrichtung ab und können daher angewendet werden.

```

346 // computation of the PRANDTL tip loss factor
347 F=1;
348
349 if (m_bTipLoss || m_bNewTipLoss)
350 {
351 double f=sin(phi/360*2*PI);
352 double g=(outer_radius-m_pos.at(i))/m_pos.at(i);
353 double Ft=2/PI*acos(exp(-blades/2*fabs(g/f)));
354 F = F*Ft;
355 }
356
357 if (m_bRootLoss || m_bNewRootLoss)
```

```

358     {
359         double f=sin(phi/360*2*PI);
360         double g=(m_pos.at(i)-inner_radius)/m_pos.at(i);
361         double Fr=2/PI*acos(exp(-blades/2*fabs(g/f)));
362         F = F*Fr;
363     }
364
365
366
367     //here the new tip loss model is implemented
368     F1 = 1;
369
370     if (m_bNewTipLoss || m_bNewRootLoss)
371     {
372         if (m_bNewTipLoss)
373         {
374             double f=sin(phi/360*2*PI);
375             double g=(outer_radius-m_pos.at(i))/m_pos.at(i);
376             double Flt = 2/PI*acos(exp(-blades/2*fabs(g/f))*(exp(-0.15*(blades*
377                         m_lambda_local[i]-21))+0.1)));
378             F1 = F1 * Flt;
379         }
380
381         if (m_bNewRootLoss)
382         {
383             double f=sin(phi/360*2*PI);
384             double g=(m_pos.at(i)-inner_radius)/m_pos.at(i);
385             double Flt = 2/PI*acos(exp(-blades/2*fabs(g/f))*(exp(-0.15*(blades*
386                         m_lambda_local[i]-21))+0.1)));           //CW
387             F1 = F1 * Flt;
388         }
389
390         if (m_bNewTipLoss || m_bNewRootLoss)
391         {
392             Cn = F1 * Cn;
393             Ct = F1 * Ct;
394         }
395     }

```

**Listing 3.9:** Berechnung der Korrekturfaktoren

Der in *Listing (3.10)* berechnete local thrust coefficient ist die dimensionslose Form des Schubs  $C_T$ . Der Schub, sprich die Kraft auf das Fluid, der die Strömung am Rotorblatt beeinflusst ist ein zentrales Element der BEM Methode. Die in der Formel vorkommenden Induktionsfaktoren  $a_a$  und  $a_r$  sind aus der vorherigen Iteration und bei der ersten Iteration mit 0 initialisiert worden. Sigma ist eine positive, von der Geometrie abhängige Konstante aus *Gl. (2.1)* und das negative Vorzeichen des sin bei Werten größer  $90^\circ$  wird durch das Quadrat ausgeglichen. Die Formel ist somit sowohl für Propeller als auch Turbinen zulässig. Der Änderung der Kraftrichtung wird durch die Änderung des Vorzeichens von  $C_n$  Rechnung getragen.

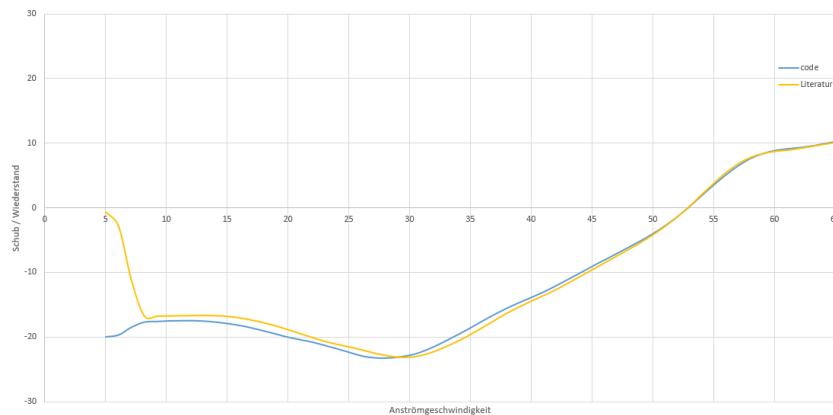
```

396     //computation of the local thrust coefficient
397     CT = sigma*pow(1-a_a,2)*Cn/pow(sin(phi/360*2*PI),2);

```

**Listing 3.10:** Berechnung des Schubbeiwerts des Rotors

Die Berechnung der Induktionsfaktoren ist der zentrale Punkt in der BEM Methode. Für die Herleitung dieser Formeln sind einige Gleichungen, Annahmen und Rechenschritte notwendig, die im *Kap. 2.4* erklärt wurden. Es wurde ebenfalls genannt, dass es zwei Gleichungen für die Berechnung von  $a'$  gibt. In der *Abb. 3.9* ist der Verlauf des Schubes über die Anströmgeschwindigkeit aufgetragen. Es wird jeweils der gleiche Propeller mit *Gl. (2.52)* aus der Literatur und *Gl. (2.29)* aus dem Quesllcode in QBlade gerechnet und einmal mit dem Programm Propcalc. Alle drei Kurven liegen in den relevanten Bereichen gut übereinander, wie in *Abb. 3.9* zu sehen ist. Die Abweichungen zwischen den von QBlade gerechneten Ergebnissen sind durchaus zulässig, da es bei numerischen Methoden immer nur zu einer Annäherung an die Realität kommen kann und je nach Formel die Ergebnisse leicht voneinander abweichen dürfen. Die etwas größeren Unterschiede zu der Rechnung von propcalc entstehen dadurch, dass zwar in beiden Programmen das gleiche Profil (E392) verwendet wird, aber die Auftriebs- und Widerstandswerte, die von xfoil berechnet werden, von den in propcalc hinterlegten Messergebnissen abweichen. xfoil hat sich aber schon ausreichend bewährt, um ohne weitere Validierung für die Berechnung verwendet zu werden. In weiterer Folge wird die *Gl. (2.29)* verwendet, da auch der ursprüngliche QBladecode damit rechnet.



**Abbildung 3.9:** Vergleich der beiden Formulierungen von  $a'$

```

399 //computation of the axial induction factor
400 if (CT <= 0.96*F)
401 {
402     a_a=1/((4*F*pow( sin( phi/360*2*PI) ,2 ))/(sigma*Cn)+1);
403 }
404 else
405 {
406     a_a = (18*F-20-3*pow( fabs(CT*(50-36*F)+12*F*(3*F-4)) ,0.5 ))/(36*F
407         -50);
408 }
409 //computation of the tangential induction factor
410 //a_r=1/((4*cos(phi/360*2*PI)*sin(phi/360*2*PI))/(sigma*Ct)-1);
411 a_r = 0.5 * (pow( fabs(1+4/pow( m_lambda_local.at( i ) ,2 )*a_a*(1-a_a)) ,0.5 )-1)
412 ;      //CW

```

**Listing 3.11:** Berechnung der axialen und tangentialen Induktionsfaktoren

Die eigentliche BEM Berechnung ist jetzt abgeschlossen. Für die nächste Iteration werden die Induktionsfaktoren dieser Iteration benötigt. Davon abhängig, wie oft bereits iteriert wurde, werden die Werte aber nur indirekt weitergegeben. Um zu verhindern, dass die Iterationen unendlich lange um das eigentliche Ergebnis springen, ohne es zu erreichen, wird bei den ersten Iterationen ein Mittelwert der letzten Iterationen gebildet und ab der 11. Iteration ein Relaxfaktor verwendet. Dieser Faktor kann beim Aufsetzen der Rechnung im Interface eingestellt werden. Die letzte Klammer in diesem Abschnitt gibt das Ende der while Schleife für die Iteration an.

```

414     //implementation of the relaxation factor
415     if (count <10)
416     {
417         a_a=a_a;
418     }
419     if (count <11)
420     {
421         a_a=0.25*a_a+0.5*a_a_old+0.25*a_a_older ;
422     }
423     else
424     {
425         a_a=relax*a_a+(1-relax)*a_a_old ;
426     }
427
428     //computation of epsilon
429     if (fabs(a_a-a_a_old)>fabs(a_r-a_r_old) )
430     {
431         eps=fabs(a_a-a_a_old) ;
432     }
433     else
434     {
435         eps=fabs(a_r-a_r_old) ;
436     }
437
438
439 }
```

**Listing 3.12:** Dämpfen und Weitergeben der axialen und tangentialen Induktionsfaktoren an die nächste Iteration

Die bisher errechneten Werte werden noch in den Speicher geschrieben.

```

442 //now results are appended in the arrays , if the results are computed later ,
443 //during a
444 //turbine simulation a zero as placeholder is appended
445 m_a_axial.append(a_a);
446 m_a_radial.append(a_r);
447 m_Fa_axial.append(F*a_a);
448 m_Fa_radial.append(F*a_r);
449 double Vrel2 = (pow((1-m_a_axial.at(i)),2)+pow(m_lambda_local.at(i)*(1+
    m_a_radial.at(i)),2));
450 m_p_normal.append(Cn*0.5*Vrel2*m_c_local.at(i));
451 m_p_tangential.append(Ct*0.5*Vrel2*m_c_local.at(i));
452 m_phi.append(phi);
453 m_alpha.append(phi-m_theta.at(i)-pitch);
454 m_CL.append(CL);
455 m_CD.append(CD);
```

```

455     m_LD.append(CL/CD);
456     m_Cn.append(Cn);
457     m_Ct.append(Ct);
458     m_F.append(F);
459     m_Reynolds.append(pow(Vrel2,0.5)*m_c_local.at(i));
460     m_DeltaReynolds.append(Reynolds);
461     m_Roughness.append(0);
462     m_Windspeed.append(0);
463     m_Iterations.append(count);
464     m_Mach.append(0);
465     m_circ.append(0);
466
467 }
```

**Listing 3.13:** Speichern der errechneten Werte

Um die Leistung und den Widerstand / Schub zu bestimmen, werden diese Werte für jedes Element separat berechnet und aufsummiert. Die Berechnung erfolgt laut Definition durch Multiplikation des jeweiligen Beiwertes mit der relativen Anströmgeschwindigkeit am Profil. Der letzte Schritt wird aber in einer anderen Funktion durchgeführt. „deltas.at(i)“ gibt dabei die Dicke dr des jeweiligen Elements an.

```

468 // calculation of power coefficient Cp//
469 double power=0, windenergy, thrust;
470 for (int i=0;i<m_pos.size(); i++)
471 {
472     power = power + m_pos.at(i)*m_p_tangential.at(i)*deltas.at(i);
473 }
474
475 power=power*blades*lambda_global/outer_radius;
476 windenergy= PI/2*pow(outer_radius,2);
477 cp = power/windenergy;
478
479 // if (cp < 0) cp = 0;                                //CW
480
481
482 // calculation of thrust coefficient Ct//
483 power=0;
484 for (int i=0;i<m_pos.size(); i++)
485 {
486     power = power + m_p_normal.at(i)*deltas.at(i)*blades;
487 }
488
489 thrust = PI/2*pow(outer_radius,2);
490 ct = power/thrust;
491
492 // if (ct < 0) ct = 0;                                //CW
493
494
495
496 }
```

**Listing 3.14:** Berechnung von Schub und Leistung

### 3.5.3 Anwendung

Die Schnitte können in QBlade durch einfaches Öffnen der .dat Datei des jeweiligen Profils in die Arbeitsumgebung geladen werden. Diese werden im „airfoil design“ Menü angezeigt. Wenn das Profil richtig erkannt wurde, wird der Name des Profils aus der ersten Zeile der eingelesenen Datei übernommen und es können sowohl die einzelnen Punkte als auch die Skeletlinie angezeigt werden. Außerdem werden im unteren Bereich Informationen wie maximale Dicke und Krümmung des Profils angezeigt.

Bei den eingescannten Profilen ist es meist der Fall, dass die Hinterkante zu rund dargestellt wird und nachbearbeitet werden muss. Dafür gibt es zwei Möglichkeiten.

1. Einerseits kann man mit einem Rechtsklick auf den Profilnamen die Option „edit foil coordinates“ auswählen und bekommt dann eine Liste mit allen Koordinaten der einzelnen Punkte angezeigt. Diese können dann einzeln editiert werden. Die veränderten Punkte werden während der Bearbeitung in der Grafik grau dargestellt, wie in Abb. 3.10 zu sehen ist. Mit dieser Möglichkeit lassen sich relativ einfach kleinere Veränderungen vornehmen.

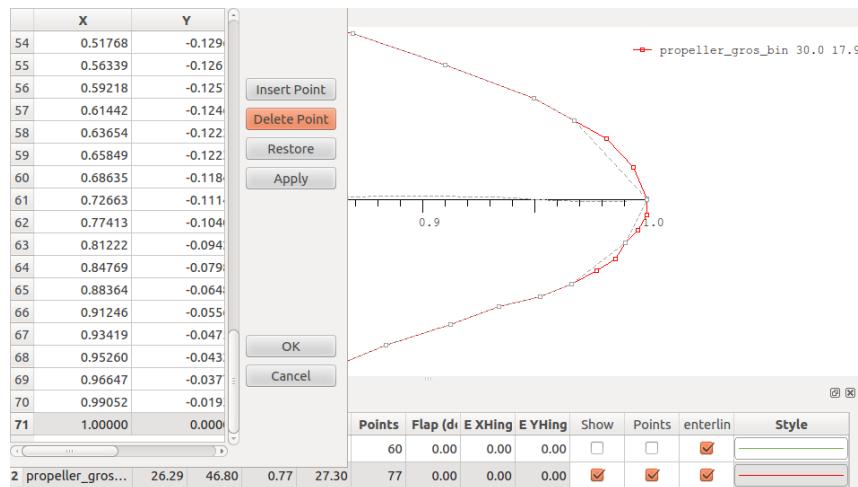


Abbildung 3.10: Schärfen der Kante zur Verhinderung von Turbulenzen

Bei großen Veränderungen ist es jedoch schwierig, gleichmäßige Krümmungen beizubehalten.

2. Andererseits wird bei den Profilen auch immer ein Spline mit angezeigt. Dieser kann durch Ziehen der Kontrollpunkte an die Geometrie angepasst werden. Mit der richtigen Anzahl an Kontrollpunkten kann die Form sehr gut angenähert und gleichzeitig eine spitze Hinterkante modelliert werden. Dabei entsteht eine sehr ruhige und glatte Form. Mit dem Befehl „store spline as foil“ wird der Spline gespeichert und kann weiterverarbeitet werden.

In der Arbeitsumgebung „Xfoil direct analysis“ werden die gespeicherten Profile analysiert und die Polare erstellt. Durch Drücken der F6 Taste kommt man zu den Einstellungen der Analyse, wie sie in Abb. 3.11 zu sehen sind. Hier werden die Werte für die Reynoldszahl und die Machzahl eingetragen. Diese können aus der, durch das python Skript catiasort.py für jeden verarbeiteten Propeller erstellten, .log Datei entnommen werden.

Weiters gibt es eine Option für NCrit.  $e^{NCrit}$  ist ein Maß dafür, wie weit Störungen in der Anströmung wachsen dürfen, bevor die Strömung turbulent wird. Bei sehr ruhiger und gleichmäßiger Anströmung wird ein größeres N gewählt als bei unruhiger, verwirbelter Anströmung.[1]

In der Dokumentation von Xfoil [9] wird folgende Tabelle angegeben:

| 1 | situation           | Ncrit                          |
|---|---------------------|--------------------------------|
| 2 |                     |                                |
| 3 | sailplane           | 12–14                          |
| 4 | motorglider         | 11–13                          |
| 5 | clean wind tunnel   | 10–12                          |
| 6 | average wind tunnel | 9 $\leq$ standard "e^9 method" |
| 7 | dirty wind tunnel   | 4–8                            |

Manchmal kommt es zu unplausiblen Ergebnissen, bei denen der Widerstandsbeiwert  $c_D$  zu Null wird. In den meisten Fällen liefern Werte zwischen 9 und 11 ganz gute Ergebnisse. Die Reynoldszahl beeinflusst die Charakteristik des Profils sehr stark. Daher ist es wichtig, dass die hier angegebene Reynoldszahl nahe an der tatsächlich im Betrieb vorkommenden liegt. In der vom Pythonscript catiasort.py erstellten .log Datei gibt es für jedes Profil eine Reynoldszahl, die mit Länge der chord und der relativen Anströmgeschwindigkeit berechnet wurde. Dieser Wert ist nur eine Annäherung, da sowohl die Anströmwinkel als auch die Induktionsfaktoren nicht berücksichtigt wurden. Diese Faktoren haben aber nur einen geringen Einfluss auf die Re-Zahl und daher können sie vernachlässigt werden. Nachdem der Bereich für den Anströmwinkel  $\alpha$  eingestellt wurde, kann die Analyse gestartet werden und die Polare werden mit XFOIL berechnet.

Bei den Profilen in der Nähe der Spitze kommt es durch die Rotation zu sehr großen Geschwindigkeiten. XFOIL ist nur für die Berechnung im Unterschallbereich konzipiert und gibt ab ca Mach 0.7 eine Warnung aus. Wendet man das Programm darüber an, muss man bedenken, dass die Ergebnisse dadurch ungenauer werden.

Nachdem die Berechnung abgeschlossen ist, werden standardmäßig die in Abb. 3.12 gezeigten Diagramme angezeigt. Mit einem Doppelklick können aber für jedes Diagramm die dargestellten Parameter geändert werden. Im linken oberen Diagramm wird der Auftrieb über dem Widerstand aufgetragen. Bei der Optimierung ist es also ein Ziel, in die linke obere Ecke des Diagramms zu kommen und bei möglichst wenig Widerstand den maximal möglichen Auftrieb zu erzeugen.

Die Berechnung der Polare ist allerdings auf einen kleinen Bereich der Angriffswinkel begrenzt, da es nicht möglich ist, eine turbulente Strömung zu simulieren. Aus diesem Grund stellen auch die zu großen Radien an der Hinterkante der Profile ein Problem dar. Wird nun das Profil in der Anströmung zu sehr geneigt, beginnt die Strömung turbulent zu werden. Somit liefert XFOIL keine Ergebnisse mehr. Durch den sehr großen Bereich der Fluggeschwindigkeiten kommt es aber auch zu einem sehr großen Bereich an Angriffswinkeln. In der Simulation traten  $\alpha$  von bis zu  $40^\circ$  auf. QBlade löst diese Art von Problemen mit einer  $360^\circ$  Extrapolation der Polaren. Diese wird in einer eigenen Arbeitsumgebung durchgeführt, siehe Abb. 3.13. Dabei werden Auftriebs- und Widerstandsbeiwerte für alle  $360^\circ$  Angriffswinkel erstellt. Für die Extrapolation stehen dabei zwei Algorithmen zur Auswahl. Der Montgomerie Algorithmus geht davon aus, dass sich die Profile bei großen Angriffswinkeln wie ebene Platten verhalten und versucht so, eine Kurve in die von XFOIL berechnete Kurve einzupassen. Mit den auf der linken Seite befindlichen

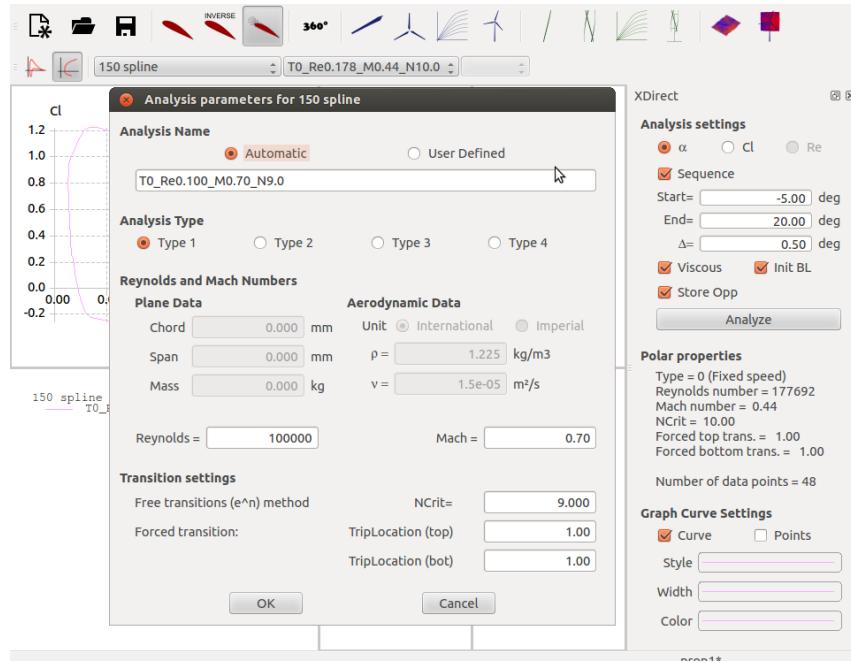


Abbildung 3.11: Einstellungen für die Berechnung der Polare

Schiebereglern kann die Form noch geringfügig beeinflusst werden. Trotzdem bleiben die Werte in diesem Bereich nur eine grobe Annäherung und stellen eine Schwachstelle in der Berechnung dar. Daher sollte versucht werden, die auftretenden  $\alpha$  möglichst schon im XFOIL zu berechnen, da diese Daten von der 360° Extrapolation unangetastet bleiben. Außerdem sei erwähnt, dass die vorhin genannten 40° nur im Bereich der Nabe auftreten und dieser Bereich für die Erzeugung des Schubes und den Leistungsverbrauch aufgrund des geringen Radius vergleichsweise wenig

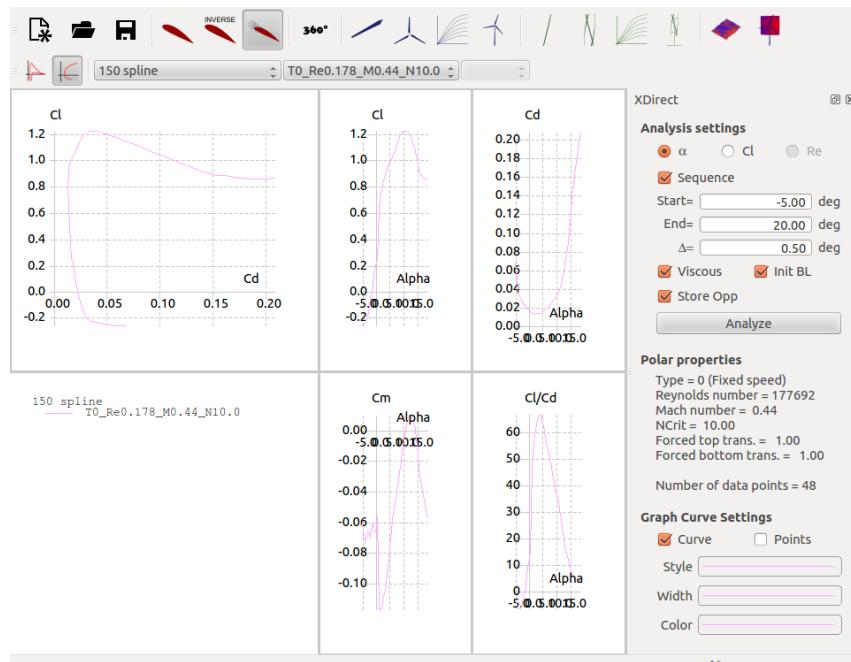
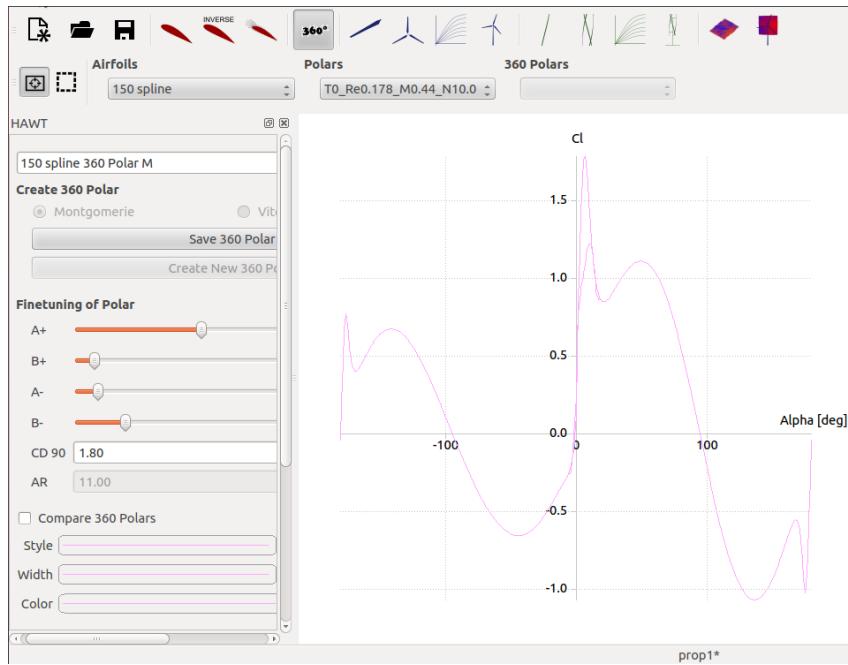


Abbildung 3.12: Darstellung der berechneten Polare

Einfluss hat. Wurden für alle benötigten Profile 360° Polare erzeugt, kann damit begonnen



**Abbildung 3.13:** Extrapolation der Polare für einen Bereich von 360°

werden, den Rotor zusammenzusetzen. Im Menü „HAWT rotor blade design“ kann ein neuer Rotor erstellt werden oder der bestehende verändert werden. Nach den Einstellungen für die 3D-Ansicht folgt der Name des Rotors. Anschließend gibt die Option „hub radius“ den Radius der Nabe an, die bei der Berechnung nicht berücksichtigt wird und als Zylinder dargestellt wird. Wählt man die Option „Blade Coords“ ab, werden alle Radien auf die Rotationsachse bezogen. Bleibt die Option jedoch aktiv, werden alle Abstände vom Beginn des Rotorblattes, also ab der Nabe, gemessen. In die darauffolgende Tabelle müssen nun alle errechneten Polare und Profile eingetragen werden. Die .log Datei soll das Ausfüllen erleichtern, da hier für jedes Profil Radius, Profillänge und Neigungswinkel gelistet sind. Wenn die Nomenklatur beibehalten wurde, sind außerdem die Profile und Polare nach den Radien benannt. Da QBlade ursprünglich für die Berechnung von großen Windkraftanlagen gedacht war, sind die Einheiten standardmäßig auf Meter eingestellt. Mit maximal zwei Kommastellen ist es damit aber nicht möglich, die Profillänge ausreichend genau anzugeben. Daher muss unter „Options -> Units..“ der Länge die Einheit mm zugeordnet werden. Jedes eingetragene Profil wird sofort in der 3D-Ansicht aktualisiert. Nachdem alle Profile eingetragen wurden und der Rotor gespeichert wurde, kann mit der Simulation und Analyse begonnen werden.

In der „Rotor BEM Simulation“ kann ein einzelnes Rotorblatt untersucht werden. Hier lässt sich erkennen, welche Bereiche des Rotors mit welchem Winkel angeströmt werden und wie viel sie zum Gesamtschub, beziehungsweise dem Moment, beitragen.

In der „Multi Parameter BEM Simulation“ werden Leistungsdaten des gesamten Rotors mit allen Rotorblättern angezeigt. Auch hier können die Diagramme mittels Doppelklick eingestellt werden. Besonders interessant ist hier der Verlauf von Schub und Leistung über die Fluggeschwindigkeit. So wie bei allen Diagrammen in QBlade können die angezeigten Daten der Diagramme mittels Rechtsklick auf das Diagramm und der Option „current Graph -> export Graph“ als .csv

exportiert werden. Von dieser Funktion wurde sowohl bei der Erstellung diverser Diagramme als auch für das Auswerteskript in python Gebrauch gemacht.

### 3.6 Bewertung mit python

Um die verschiedenen Simulationen bewerten und vergleichen zu können, sind festgelegte Kennwerte notwendig. Ein weiteres Python Skript soll nun die aus der Berechnung stammenden Diagramme auswerten und einen Wert errechnen, der es ermöglicht die verschiedenen Profile zu vergleichen. Da das Ziel ein möglichst großer Schub bei möglichst wenig verbrauchter Leistung ist, erscheinen diese beiden Werte sinnvoll. Die Aufgabe besteht aber nicht darin, den Propeller für einen bestimmten Betriebspunkt auszulegen. Vielmehr soll der Propeller über den gesamten Beschleunigungsvorgang einen guten Wirkungsgrad aufweisen. Da auch beim Wettbewerb die verbrauchte Energie gemessen wird, soll diese durch Multiplikation der verbrauchten Leistung mit der Zeit berechnet werden. Würde man nun einen Wirkungsgrad berechnen wollen, muss man die durch den Schub an das Flugzeug abgegebene Energie auf die verwendete Energie beziehen. Die an das Flugzeug abgegebene Energie ist dann  $\int dE = \int Fvdt$ . Damit wird aber Schub bei höheren Geschwindigkeiten stärker bewertet als jener bei langsamen Geschwindigkeiten. Bei der Beschleunigung ist aber ein hoher Schub über den gesamten Geschwindigkeitsbereich gefragt. Daher soll als zweiter Vergleichswert der Impuls, das Integral von  $\int Fdt$ , verwendet werden.

Die Berechnung in QBlade erfolgt in gleichbleibenden Geschwindigkeitsabständen. Die Integration soll aber nach der Zeit erfolgen. Für die Umrechnung wird als Näherung eine konstante Beschleunigung angenommen, die aus der gegebenen Anfangsgeschwindigkeit, der Endgeschwindigkeit und der Dauer des Beschleunigungsvorganges folgendermaßen berechnet wird:

$$\text{Beschleunigung} = \frac{v_{\text{ende}} - v_{\text{anfang}}}{t} \quad (3.6)$$

Die Zeit zwischen zwei Geschwindigkeiten kann dann mit

$$dt = \frac{v_2 - v_1}{\text{Beschleunigung}} \quad (3.7)$$

berechnet werden. Diese Bestimmung der Zeit ist jedoch nur eine grobe Annäherung. Will man die verbrauchte Energie mit Messdaten vergleichen, müsste man den Geschwindigkeitsverlauf ebenfalls aus Messdaten verwenden oder ein genaues Modell erstellen und Gewicht, Luftwiderstand und Schub berücksichtigen, um die Beschleunigung zu ermitteln.

Die Integration wurde durch Summe der diskretisierten Einzelabschnitte angenähert. Für die Leistung wird der Mittelwert der Leistungen von zwei Zeitpunkten ( $P_1, P_2$ ) verwendet und mit der zeitlichen Differenz der beiden Zeitpunkte ( $dt$ ) multipliziert.

$$E = \Sigma(P_1 + P_2)/2 * \Delta t \quad (3.8)$$

Analog dazu wurde die Summe für den Schub gebildet:

$$I = \Sigma(F_1 + F_2)/2 * \Delta t \quad (3.9)$$

Dabei sei erwähnt, dass es sich bei dem Verlauf nicht um einen dynamischen Verlauf handelt, sondern nur eine Aneinanderreihung von statischen Zuständen ist. So werden dynamische Effekte wie Massenträgheit der Luft und des Flugzeugs nicht berücksichtigt. In Python erfolgt die Berechnung mit der Funktion bewerten.

```

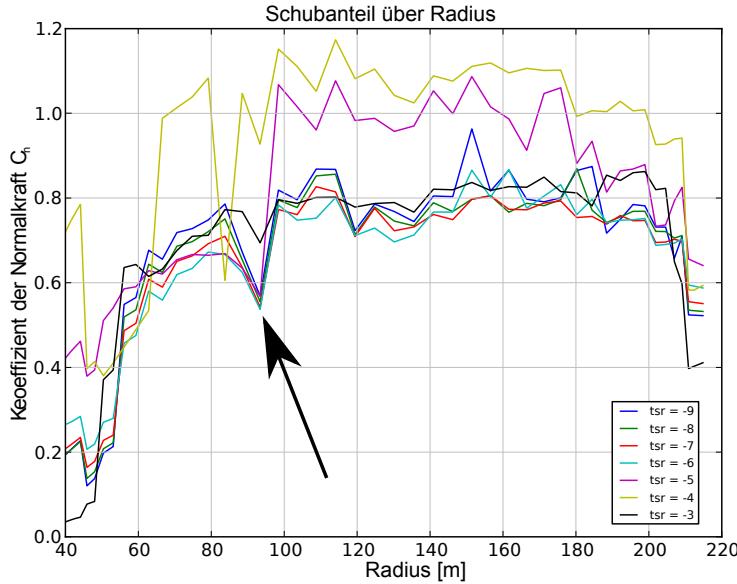
1 def bewerten(v_power, powerdat, thrustqb):
2     if len(v_power)!=len(powerdat) or len(v_power)!=len(thrustqb):
3         print('Fehler Eingabedaten überprüfen! Die Eingangslisten sind
4             unterschiedlich lang.')
5     else:
6         out.write("\nunverwendete Energie      akk Schub      Schub/Energie")
7         vstart=v_power[0]
8         vend=v_power[-1]
9         pverbrauch=0
10       akkumschub=0
11        a=(vend-vstart)/time
12        i=0
13        up=len(powerdat)-1
14        while i<up:
15            ii=i+1
16            dt=(v_power[ii]-v_power[i])/a
17            pverbrauch+=(powerdat[i]+powerdat[ii])*dt/2
18            akkumschub+=(thrustqb[i]+thrustqb[ii])*dt/2
19            i+=1
20        out.write("\n"+str(pverbrauch)+"      "+str(akkumschub)+"      "+str(
21             akkumschub/pverbrauch))

```

Als Eingabe werden Listen für die Geschwindigkeit, die Leistung und den Schub erwartet. Da diese aus der Gleichen Simulation stammen müssen müssen sie auch gleich viele Einträge enthalten. Für die Startgeschwindigkeit wird der erste Wert in der Liste der Geschwindigkeiten verwendet und für die Endgeschwindigkeit der letzte Wert. Für jeden Einzelabschnitt wird die angenommene Differenzzeit errechnet. Danach werden den Summen die Werte des Einzelementes entsprechend Gl. (3.8) und Gl. (3.9) hinzu addiert. Am Ende der Funktion werden die Ergebnisse in eine außerhalb der Funktion geöffnete Textdatei geschrieben.

### 3.7 Auswertung

In Abb. 3.14 ist der Schubverlauf über dem Radius für den Propeller 1 aufgetragen. Im Bereich der Nabe (40mm) und der Rotorblattspitze wird der Schub zum Einen von der Geometrie zum Anderen auch von den sich bildenden Wirbeln beeinträchtigt. Die Wirbel werden durch den in Kap. 2.4.4 erklärten Prandtl Tip/Root Loss Faktor berücksichtigt. Außerdem kann man bei einem Radius von ca. 95mm einen starken Einbruch erkennen. Hier gibt es also Verbesserungspotential.



**Abbildung 3.14:** Schubkoeffizient  $C_n$  über den Radius bei verschiedenen  $\lambda$

Vergleicht man die, in Abb. 3.15 dargestellten, Profile bei 90, 95 und 100 mm, unterscheiden sie sich nur minimal in Form und Krümmung.

Abb. 3.16 und Abb. 3.17 zeigen die Diagramme von  $C_L$  über  $\alpha$  und  $C_D$  über  $\alpha$ . Aufgrund der Größenordnung der Koeffizienten ist der Einfluss von  $C_L$  auf den Schub größer. Für  $C_L$  hat das Profil 95 fast über den ganzen dargestellten Bereich von  $\alpha$  größere Werte.

Das in Abb. 3.18 dargestellte Verhältnis von  $C_L/C_D$  beschreibt wie gut ein Profil, bei begrenztem Widerstand, Auftrieb erzeugen kann. Ein hoher Wert weist also auf ein effektives Profil hin und ist daher anzustreben. Bei  $\alpha$ -Werten zwischen  $5^\circ$  und  $10^\circ$  ist das Profil 95 um ca 10% besser als das Profil 100. In den übrigen Bereichen sind die Profile in etwa gleich gut. Das Profil 90 liefert im gesamten dargestellten Bereich deutlich kleinere Werte. Dies deutet darauf hin, dass das Profil 95 am effektivsten Antriebsleistung in Schub umsetzen kann.

Für den Einsatz in einem Propeller ist der tatsächlich erzeugte Schub relevant. Die Abb. 3.19 und Abb. 3.20 zeigen den Schub und die Leistung des ganzen Propeller bei verschiedenen Fluggeschwindigkeiten. Bei der Linie Profil 100 wurden die Profile der Basis bei den Radien 90, 95 und 100 durch das Profil 100 ersetzt. Bei der anderen Kurve wurde statt Profil 100 das Profil 95 verwendet. Bei diesen Bildern ist der Vorteil, in der Nähe von  $\alpha = 5^\circ$ , des Profils 95 gegenüber dem

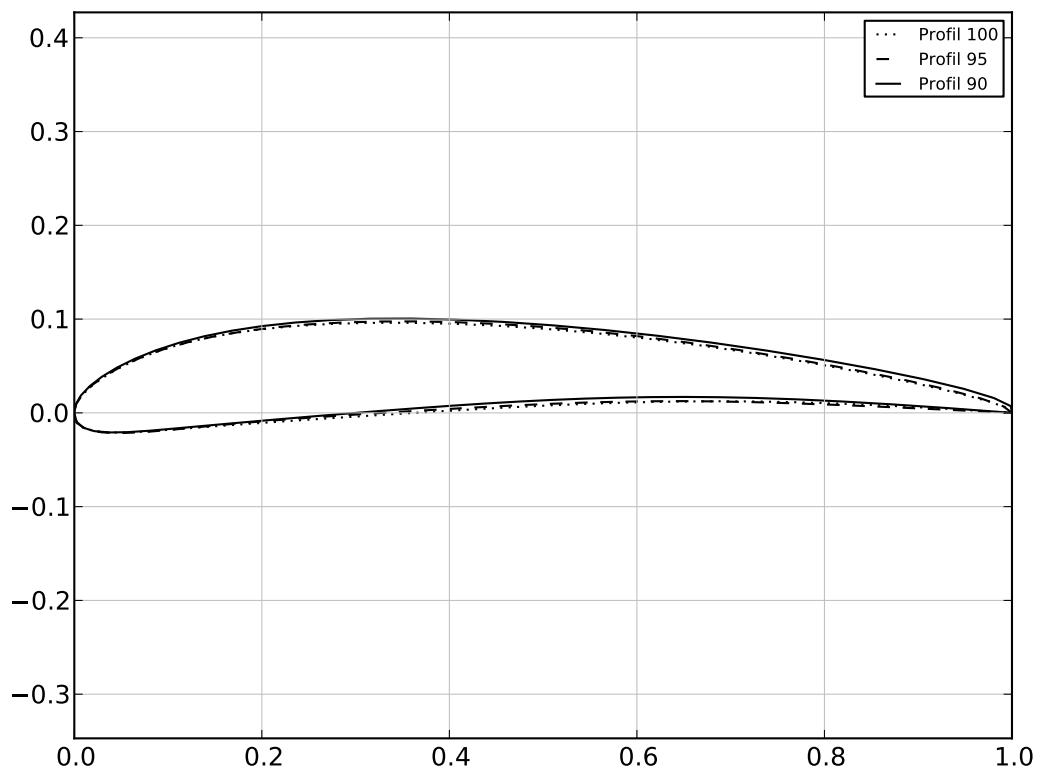


Abbildung 3.15: Vergleich der Profile 90, 95 und 100

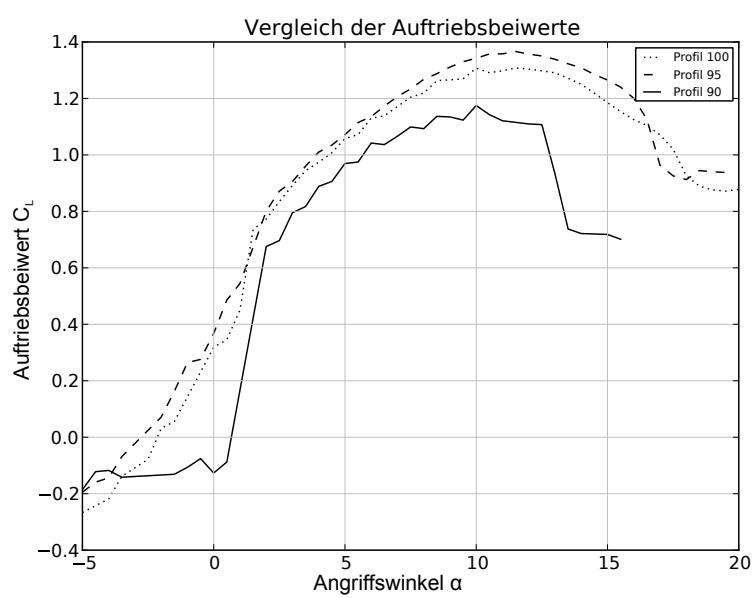
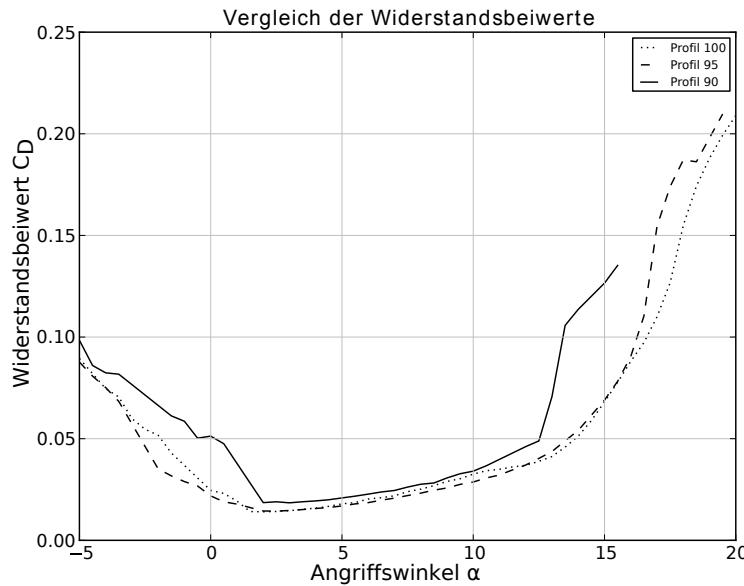
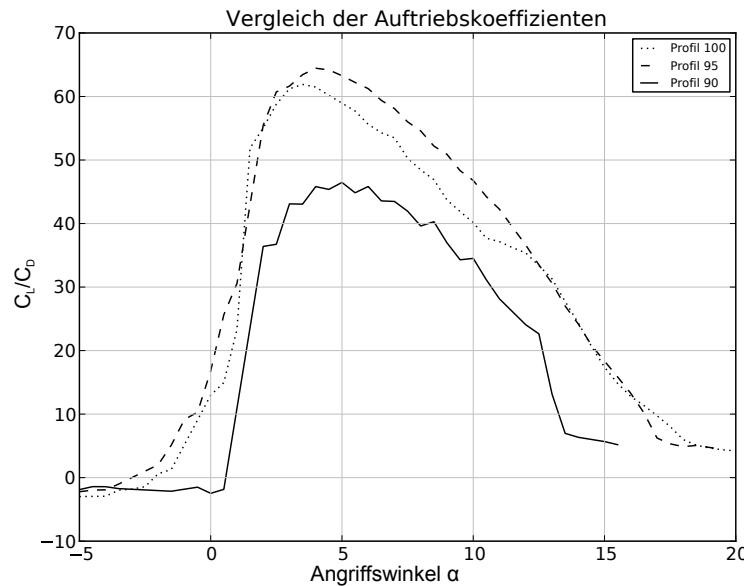
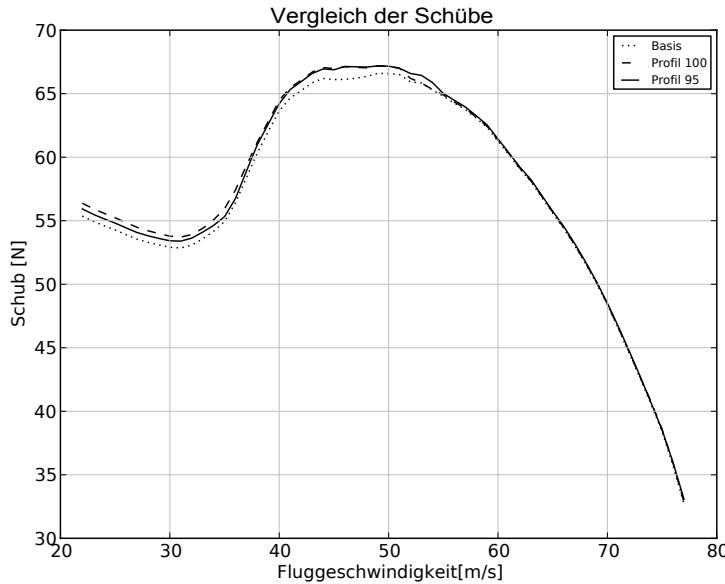


Abbildung 3.16:  $C_L$  über  $\alpha$

Abbildung 3.17:  $C_D$  über  $\alpha$ 

Profil 100 nicht mehr zu erkennen. Das kommt daher, dass der Propeller so ausgelegt ist dass er bei maximaler Fluggeschwindigkeit bei ca  $\alpha = 0^\circ$  liegt. Bei dieser Geschwindigkeit ist  $\Phi$  bei seinem Maximalwert. Dadurch kommt es beim Propeller dazu, dass die Auftriebskraft weniger Einfluss auf den Schub hat als bei Kleinem  $\Phi$ . Gleichzeitig bewirkt aber die Widerstandskraft eine Kraft die dem Schub entgegenwirkt. So sind die beiden Kurven sehr nahe beieinander und der Vorteil des Profil 95 kommt nicht zur Geltung. Wird die Fluggeschwindigkeit jedoch geringer

Abbildung 3.18: Verhältnis von  $C_L/C_D$  mit verschiedenen Profilen

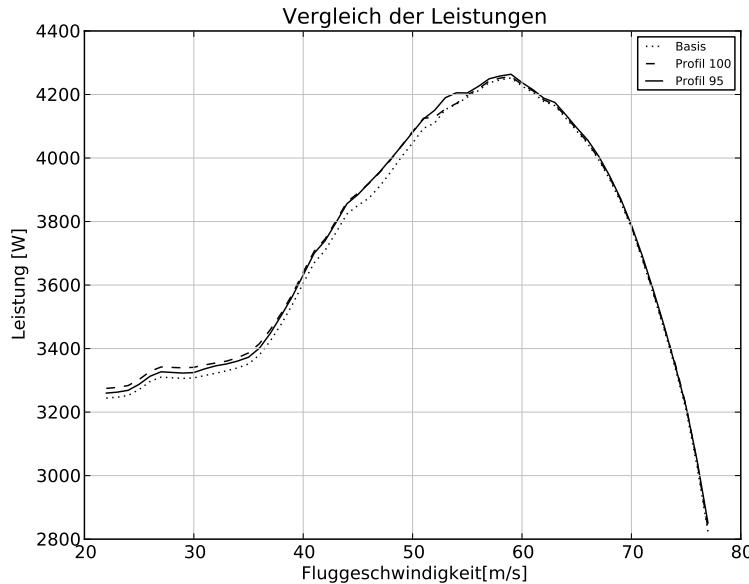


**Abbildung 3.19:** Vergleich des Schubes

nimmt  $\alpha$  zu und steigt in Bereiche die außerhalb von Abb. 3.18 liegen. In diesem Bereich sind die Auftriebs und Widerstandsbeiwerte extrapoliert und daher ungenauer.

Auch wenn die im Kap. 3.6 eingeführte Bewertung angewandt wird ergibt sich für das Profil 100 das Beste Verhältnis von Schub zu Leistung:

|   |   |
|---|---|
| 1 | Basis :   |
| 2 | verwendete Energie      akk Schub      Schub/Energie  |
| 3 | 11158.9681909      171.419303727      0.0153615729335 |



**Abbildung 3.20:** Vergleich der Leistung

|   |                    |               |                 |
|---|--------------------|---------------|-----------------|
| 4 | Profil 95:         |               |                 |
| 5 | verwendete Energie | akk Schub     | Schub/Energie   |
| 6 | 11221.0552364      | 172.686328636 | 0.0153894910059 |
| 7 | Profil 100:        |               |                 |
| 8 | verwendete Energie | akk Schub     | Schub/Energie   |
| 9 | 11223.8456727      | 172.927418182 | 0.0154071450396 |

Das größte Problem bei der Optimierung ist der sehr große Geschwindigkeitsbereich, bei dem ein Propeller betrieben wird. In diesem Fall variiert die Geschwindigkeit zwischen 80 und 280 km/h. Die Drehzahl des Propellers bleibt jedoch relativ konstant bei 9000 rpm. Berechnet man den Anströmwinkel, ohne die Beeinflussung durch den Propeller zu berücksichtigen, mit

$$\varphi = \arctan \left( \frac{v_0}{r\Omega} \right) \quad (3.10)$$

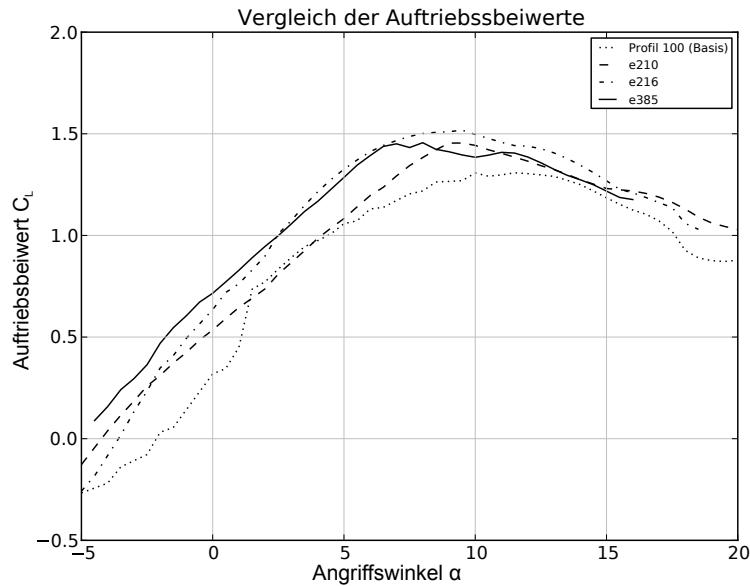
erhält man für 80km/h mit dem Radius von 0,2m ein  $\varphi$  von  $36,53^\circ$  für 280 km/h hingegen  $68,91^\circ$ . Daraus ergibt sich eine Differenz von  $32,38^\circ$ . Das bedeutet, dass selbst wenn das Profil bei der Endgeschwindigkeit von 280 km/h so angeordnet ist, dass es mit  $\alpha = 0^\circ$  angeströmt wird erfährt es am Beginn der Beschleunigung ein  $\alpha = 32,38^\circ$ . In der Luftfahrt werden Profile oft nur bis  $15^\circ$  verwendet. Die Strömung wird immer unvorteilhafter und die aerodynamischen Eigenschaften der Profile ähneln immer mehr der einer ebenen Platte. Bei ca.  $20^\circ$  löst sich bei den meisten Profilen die Strömung ab. Dies äußert sich in Xfoil dadurch, dass die Berechnung keine Ergebnisse mehr liefert. QBlade begegnet diesem Problem mit der  $360^\circ$ -Extrapolation. Die Extrapolation kann sehr stark mit Schiebereglern beeinflusst werden. Damit werden die Leistungswerte im niedrigen Geschwindigkeitsbereich wesentlich ungenauer als bei hohen Geschwindigkeiten.

Auf airfoiltools.com [2] sind viele verschiedene Profile zum Download verfügbar. Bei der Durchsicht der Eppler-Profiles zeigten einige Ähnlichkeiten zu den im Propeller verwendeten Profilen. Bei der Berechnung der Polaren zeigten einige ganz gute Werte. In Abb. 3.21 und Abb. 3.22 werden die Auftriebs- und Widerstandsbeiwerte mit dem ursprünglichen Profil an der Position von 100mm verglichen.

Bei dem Versuch das Profil bei 100mm Radius zu optimieren wurden die vorhin untersuchten Eppler-Profiles bei den Profilen von 90 bis 100 eingesetzt und dann der gesamte Flügel berechnet. In Abb. 3.23 lässt sich ein erhöhter Schub bei den schnellen Fluggeschwindigkeiten erkennen. Bei den eher flachen Anströmwinkeln kommen die besseren Auftriebswerte gut zur Geltung. Durch die  $360^\circ$ -Extrapolation werden die Profile im niedrigen Geschwindigkeitsbereich schlechter bewertet als das Originalprofil. In Abb. 3.24 ist der dazugehörige Leistungsverbrauch dargestellt.

Bei der Bewertung über den Geschwindigkeitsbereich stellt sich heraus, dass Vorteile des Originalprofils bei einem Radius von 100mm überwiegen. Der Gesamtschub pro verbrauchter Leistung liegt leicht über dem der anderen Profile. Auch der Gesamtschub an sich ist größer als bei den eppler Profilen.

|   |                    |               |                 |
|---|--------------------|---------------|-----------------|
| 1 | Basis :            |               |                 |
| 2 | verwendete Energie | akk Schub     | Schub/Energie   |
| 3 | 11224.0075364      | 172.9311      | 0.0154072508807 |
| 4 | e210 :             |               |                 |
| 5 | verwendete Energie | akk Schub     | Schub/Energie   |
| 6 | 11189.5970455      | 171.738616364 | 0.0153480608521 |

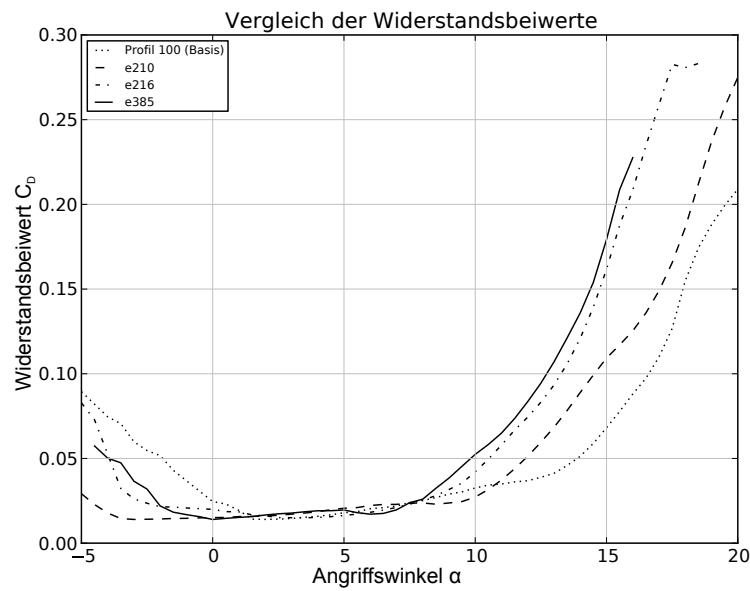


**Abbildung 3.21:** Vergleich der Auftriebsbeiwerte von verschiedenen Profilen

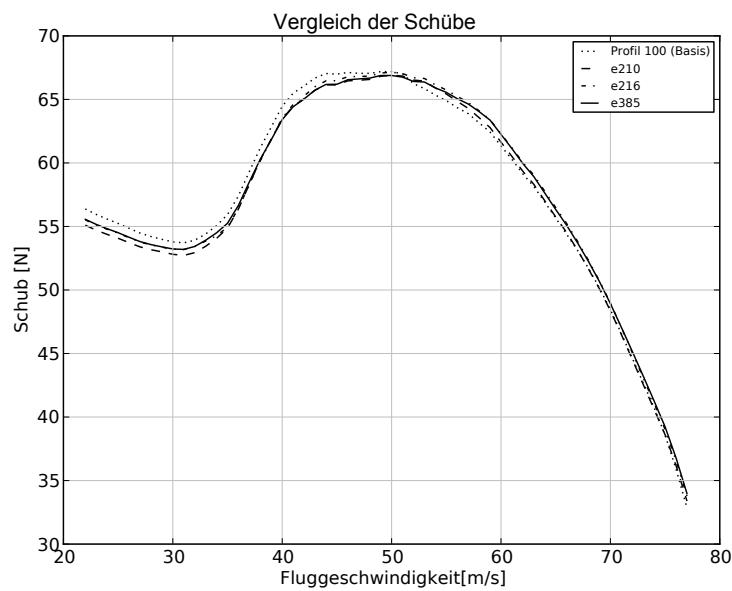
```

7| e216:
8| verwendete Energie      akk Schub      Schub/Energie
9| 11276.82228364        172.913193273    0.0153335026879
10| e385:
11| verwendete Energie      akk Schub      Schub/Energie
12| 11261.5328727        172.760308636    0.015340745402

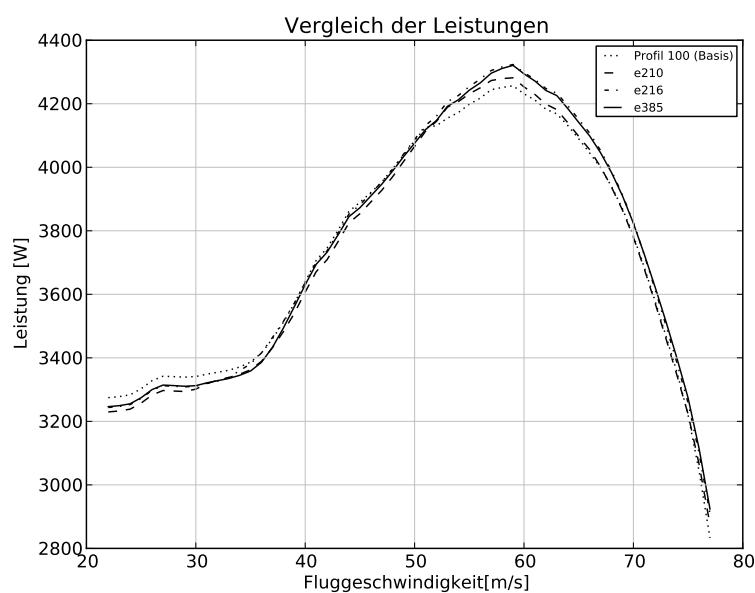
```



**Abbildung 3.22:** Vergleich der Widerstandsbeiwerte von verschiedenen Profilen



**Abbildung 3.23:** Vergleich der Schübe bei verschiedenen Profilen



**Abbildung 3.24:** Vergleich der Leistungen bei verschiedenen Profilen



# Kapitel 4

## Ausblick

Diese Arbeit beschreibt die theoretischen Grundlagen der BEM-Theorie und deren praktische Anwendung im Simulationsprogramm QBlade. Es wird gezeigt in welchen Bereichen die Berechnung von Propellern und Turbinen gleich ist und wo es Unterschiede gibt. Weiters wird eine Möglichkeit zur Analyse der Ergebnisse beschrieben.

Weiters wurden zwei Propeller optisch vermessen und mit QBlade simuliert. Um die durch einen 3D-Scan erzeugten Daten in Qblade einlesen zu können wurden die Geometrien in Catia zerschnitten und ein Python Skript erstellt, das die Schnitte vereinbart und die enthaltenen Punkte sortiert. Für die beiden zur Verfügung gestellten Propeller wurde eine Simulation durchgeführt, deren Ergebnisse für einen Propeller auch diskutiert wurden.

Interessierte können sich nun selbst die Zeit nehmen und diverse Profile unterschiedlich anordnen und die Ergebnisse vergleichen. Um die Rechenergebnisse richtig bewerten zu können, wäre es natürlich interessant, sie mit Messergebnissen aus Versuchen zu vergleichen. Wenn der Rumpf eines Modellflugzeuges an einer Schnur in einem Windkanal platziert wird, kann der vom Propeller erzeugte Schub einfach mittels Kraftmessdose gemessen werden. Wenn dabei auch Strom und Spannung am Motor gemessen werden, sind alle Parameter vorhanden, um die Simulationsergebnisse zu verifizieren.

Außerdem bietet QBlade für Turbinen einige Optimierungstools, die für Propeller auch interessant wären.



# Anhang A

## Anhang

### A.1 BData.cpp

```
1 ****
2
3     BData Class
4         Copyright (C) 2010 David Marten qblade@web.de
5
6     This program is free software; you can redistribute it and/or modify
7     it under the terms of the GNU General Public License as published by
8     the Free Software Foundation; either version 2 of the License, or
9     (at your option) any later version.
10
11    This program is distributed in the hope that it will be useful,
12    but WITHOUT ANY WARRANTY; without even the implied warranty of
13    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
14    GNU General Public License for more details.
15
16    You should have received a copy of the GNU General Public License
17    along with this program; if not, write to the Free Software
18    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
19
20 ****
21
22 #include "BData.h"
23 #include <QDebug>
24 #include <math.h>
25 #include "../Objects/Polar.h"
26 #include "../Globals.h"
27
28 QList<void*> *BData::s_poa360Polar;
29
30 BData::BData()
31 {
32
33     m_bShowPoints = false;
34     m_bVisible = true;
35     m_Style = 0;
```

```

37     m_Width          =    1;
38
39     m_bTipLoss = false;
40     m_bRootLoss= false;
41     m_b3DCorrection = false;
42     m_bInterpolation = false;
43     m_bNewRootLoss = false;
44     m_bNewTipLoss = false;
45     m_bCdReynolds = false;
46
47     elements      =    100;
48     epsilon       =    0.001;
49     iterations    =    1000;
50     relax         =    0.4;
51
52 }
53
54 BData::~BData()
55 {
56 }
57
58 CPolar* BData::Get360Polar( QString m_FoilName,  QString PolarName)
59 {
60
61
62     CPolar *pPolar;
63     for ( int i=0; i < s_poa360Polar->size() ; i++)
64     {
65         pPolar = (CPolar *) s_poa360Polar->at(i);
66         if (pPolar->m_FoilName == m_FoilName && pPolar->m_PlName ==
67             PolarName)
68         {
69             return pPolar;
70         }
71     }
72     return NULL;
73 }
74
75 void BData::Init(CBlade *pWing,  double lambda)
76 {
77     //discretization of a wing
78
79     m_WingName=pWing->m_WingName;
80     lambda_global=lambda;
81     outer_radius = pWing->m_TPos[pWing->m_NPanel];
82     inner_radius = pWing->m_TPos[0];
83     length=outer_radius-inner_radius;
84     delta=length/elements;
85
86     blades = double(pWing->blades);
87
88     lambdaGlobal.sprintf("%.2f",lambda_global);
89
90     double between=0;
91     double chord=0;

```

```
92     double lambdalocal=0;
93     double theta=0;
94     double sections = elements+1;
95     double deltaangle = PI/sections ;
96     double tot=0;
97     double thisdelta=0;
98     double position=0;
99
100    //this is the sinusoidal spacing of the elements
101   for (double i=deltaangle ;i<=PI; i=i+deltaangle)
102   {
103       dist.append(sin(i));
104       tot = tot + sin(i);
105   }
106   thisdelta = length/tot;
107
108
109  for (int i=0;i<dist.size();i++)
110  {
111      //deltas hold the width of each element
112      deltas.append(thisdelta*dist.at(i));
113  }
114
115  //the root radius of the wing as first position
116  position = pWing->m_TPos[0];
117
118  //now the discretization starts
119  for (int i=0;i<elements ;i++)
120  {
121
122      //the positions mark the centers of each element
123      position=position+deltas[i]/2;
124
125      for (int j=0;j<pWing->m_NPanel+1;j++)
126      {
127
128          //here it is computed between which stations the current element center
129          //lies
130          if (position >= pWing->m_TPos[j] && position <= pWing->m_TPos[j+1])
131          {
132              from=j ;
133              to=j+1;
134          }
135      }
136      //between stores how close the element is to the next station (between = 1 ->
137      //on the next station , between = 0 -> on the last station)
138      between = (position-pWing->m_TPos[from]) / (pWing->m_TPos[to]-pWing->m_TPos[
139          from]);
140      //the chord for every element
141      chord = pWing->m_TChord[from]+between*(pWing->m_TChord[to]-pWing->m_TChord[
142          from]);
143      //the local tip speed ratio for every element
144      lambdalocal = lambda_global*position / outer_radius;
145      //the build angle for every element
146      theta = pWing->m_TTwist[from]+between*(pWing->m_TTwist[to]-pWing->m_TTwist[
147          from]);
```

```
143     m_between.append(double(1)-between);
144     m_pos.append(position);
145     m_c_local.append(chord);
146     m_lambda_local.append(lambdalocal);
147     m_theta.append(theta);
148     //this stores a pointer to the polar from the last section
149     m_polar.append(pWing->m_Polar[from]);
150     //this stores a pointer to the polar at the next section
151     m_polarTO.append(pWing->m_Polar[to]);
152     //this stores a pointer to the foil from the last section
153     m_foil.append(pWing->m_RFoil[from]);
154     //this stores a pointer to the foil from the next section
155     m_foilTO.append(pWing->m_RFoil[to]);
156
157
158     position=position+deltas[i]/2;
159
160 }
161 }
162 }
163 }
164
165
166
167 void BData::OnBEM(double pitch)
168 {
169
170
171 double a_a, a_a_old, a_a_older;
172 double a_r, a_r_old;
173 double phi, alpha, F1;
174 double eps, CL, CD, Cn, Ct, CL2, CD2, CT;
175 double DeltaAlpha, sigma, localReynolds;
176 int i;
177 double F, Reynolds;
178
179 windspeedStr.sprintf("%.2f", windspeed);
180
181 m_PolarPointers.clear();
182
183
184 //every element gets a pointer assigned to the polar at the section before (
185     m_PolarPointers)
186 //the elements center and after the elements center (m_PolarPointersTO)
187 for (int i=0;i<m_polar.size();i++)
188 {
189     m_PolarPointers.append(Get360Polar(m_foil.at(i),m_polar.at(i)));
190     m_PolarPointersTO.append(Get360Polar(m_foilTO.at(i),m_polarTO.at(i)));
191 }
192
193 //now the first loop starts over all elements
194 for (i=0;i<m_theta.size();i++)
195 {
196     //the induction factors and epsilon are initialized
197     eps = 10000;
198     a_a=0;
```

```
198     a_r=0;
199     a_a_old=0;
200     a_a_older=0;
201     a_r_old=0;
202
203
204 //the counter for the number of iterations is set to zero
205 int count =0;
206
207 //this is the criterion for an iteration to converge
208 while (eps > epsilon)
209 {
210
211 //when the maximum number of iterations is reached the iterations are stopped
212     count++;
213     if (count == iterations)
214     {
215         break;
216     }
217
218     //the angle phi is computed
219 // new code CW
220     if (m_lambda_local.at(i)>=0)
221     {
222         phi = atan( (1-a_a)/(1+a_r) / m_lambda_local.at(i) )/2/PI*360;
223     }
224     else
225     {
226         phi =180-atan( fabs( (1-a_a)/(1+a_r) / m_lambda_local.at(i) ))/2/PI
227             *360;
228     }
229 //end new code CW
230     //alpha is computed
231     alpha=phi-m_theta.at(i)-pitch;
232
233     while (alpha < -180 ) alpha+=360;
234     while (alpha > 180 ) alpha-=360;
235
236
237     bool found=false;
238     bool found2=false;
239
240 //here the polar data from the polar at the last section is extracted
241 for (int j=0; j< m_PolarPointers.at(i)->m_Alpha.size();j++)
242 {
243
244     if ( m_PolarPointers.at(i)->m_Alpha.at(j) >= alpha && !found)
245     {
246
247
248         DeltaAlpha = m_PolarPointers.at(i)->m_Alpha.at(j)-m_PolarPointers.
249             at(i)->m_Alpha.at(j-1);
250 }
```

```

251     CL = m_PolarPointers .at( i )->m_Cl .at( j -1)+(m_PolarPointers .at( i )->
252         m_Cl .at( j )-m_PolarPointers .at( i )->m_Cl .at( j -1))/DeltaAlpha *( 
253             alpha-m_PolarPointers .at( i )->m_Alpha .at( j -1));
254     CD = m_PolarPointers .at( i )->m_Cd .at( j -1)+(m_PolarPointers .at( i )->
255         m_Cd .at( j )-m_PolarPointers .at( i )->m_Cd .at( j -1))/DeltaAlpha *( 
256             alpha-m_PolarPointers .at( i )->m_Alpha .at( j -1));
257
258     if ( m_bCdReynolds )
259     {
260         localReynolds = pow( pow( windspeed , 2)+pow( m_lambda_local .at( i ) *
261             windspeed , 2) , 0.5)*m_c_local .at( i )/visc*rho ;
262         CD = CD * pow( m_PolarPointers .at( i )->m_Reynolds/localReynolds ,
263             0.2 );
264     }
265
266 // implementation the 3D correction after SNEL
267     if ( m_b3DCorrection && m_pos .at( i ) <= outer_radius
268         *0.8 && alpha >= m_PolarPointers .at( i )->
269         m_ACrit )
270     {
271         qDebug() << m_PolarPointers .at( i )->m_ACrit
272         << m_PolarPointers .at( i )->m_FoilName << m_PolarPointers .at( i )->m_ASpec;
273
274         double blend = 0;
275         if ( alpha < 30 && alpha >= m_PolarPointers
276             .at( i )->m_ACrit) blend = 1;
277         else if ( alpha <= 50 && alpha >
278             m_PolarPointers .at( i )->m_ACrit) blend
279             = 1-(alpha -30)/20;
280
281         if ((2*PI*(alpha-m_PolarPointers .at( i )->
282             m_ACrit)/360*2*PI-CL)>0)
283             CL = CL + (double(3.1)*pow( m_lambda_local .
284                 at( i ) ,2))/(double(1)+pow(
285                     m_lambda_local .at( i ) ,2))*blend*pow((
286                         m_c_local .at( i )/m_pos .at( i )) ,2)*(
287                             m_PolarPointers .at( i )->m_ASpec*(alpha-
288                             m_PolarPointers .at( i )->m_ACrit)-CL);
289     }
290     found = true ;
291 }
292
293 }
294
295 //here the polar data from the polar at the next section is extracted
296 if ( m_bInterpolation && m_PolarPointers .at( i )->m_PlName !=
297     m_PolarPointersTO .at( i )->m_PlName)
298 {
299     for ( int k=0; k< m_PolarPointersTO .at( i )->m_Alpha .size () ;k++)
300     {
301
302         if ( m_PolarPointersTO .at( i )->m_Alpha .at( k ) >= alpha && !found2 )
303         {

```

```

288
289     DeltaAlpha = m_PolarPointersTO . at ( i )->m_Alpha . at ( k )-
290             m_PolarPointersTO . at ( i )->m_Alpha . at ( k-1 );
291
292     CL2 = m_PolarPointersTO . at ( i )->m_Cl . at ( k-1 )+(m_PolarPointersTO
293         . at ( i )->m_Cl . at ( k )-m_PolarPointersTO . at ( i )->m_Cl . at ( k-1 ))/
294         DeltaAlpha*(alpha-m_PolarPointersTO . at ( i )->m_Alpha . at ( k-1 ))
295         );
296     CD2 = m_PolarPointersTO . at ( i )->m_Cd . at ( k-1 )+(m_PolarPointersTO
297         . at ( i )->m_Cd . at ( k )-m_PolarPointersTO . at ( i )->m_Cd . at ( k-1 ))/
298         DeltaAlpha*(alpha-m_PolarPointersTO . at ( i )->m_Alpha . at ( k-1 ))
299         );
300
301
302     //implementation the 3D correction after SNEL
303     if (m_b3DCorrection && m_pos . at ( i ) <=
304         outer_radius*0.8 && alpha >=
305         m_PolarPointers . at ( i )->m_ACrit)
306     {
307         //qDebug () << m_PolarPointers . at ( i )
308         ->m_ACrit << m_PolarPointers . at ( i )->m_FoilName << m_PolarPointers . at ( i
309         )->m_ASpec;
310
311         double blend = 0;
312         if (alpha < 30 && alpha >=
313             m_PolarPointers . at ( i )->m_ACrit
314             ) blend = 1;
315         else if (alpha <= 50 && alpha >
316             m_PolarPointers . at ( i )->m_ACrit
317             ) blend = 1-(alpha-30)/20;
318
319         if (((2*PI*(alpha-m_PolarPointers .
320             at ( i )->m_ACrit)/360*2*PI-CL)
321             >0)
322             CL = CL + (double(3.1)*pow(
323                 m_lambda_local . at ( i ),2))/(
324                 double(1)+pow(m_lambda_local .
325                     at ( i ),2))*blend*pow((m_c_local
326                     . at ( i )/m_pos . at ( i )),2)*(
327                         m_PolarPointers . at ( i )->m_ASpec
328                         *(alpha-m_PolarPointers . at ( i )
329                             ->m_ACrit)-CL);
330
331         }
332         //here the data is interpolated between the two polars if foil
333         interpolation is on
334         Reynolds = m_PolarPointers . at ( i )->m_Reynolds*m_between . at ( i )
335             +(1-m_between . at ( i ))*m_PolarPointersTO . at ( i )->m_Reynolds;

```

```

316         double newCL = CL*m_between . at ( i )+(1.0-m_between . at ( i ))*CL2;
317         double newCD = CD*m_between . at ( i )+(1.0-m_between . at ( i ))*CD2;
318
319         CL = newCL;
320         CD = newCD;
321         found2 = true ;
322     }
323 }
324
325
326
327
328 }
329
330 //computation of normal and thrust coefficient
331 Cn=CL*cos (phi/360*2*PI)+CD*sin (phi/360*2*PI) ;
332 Ct=CL*sin (phi/360*2*PI)-CD*cos (phi/360*2*PI) ;
333
334
335
336 //computation of solidity
337
338 sigma = fabs( m_c_local . at ( i )*cos (m_theta . at ( i )/360*2*PI)*blades/2/PI/
339             m_pos . at ( i )) ; //CW
340
341
342 //the old induction factors are stored to compute the convergence
343 //criterion
344 a_a_older=a_a_old;
345 a_a_old=a_a;
346 a_r_old=a_r;
347
348
349 //computation of the PRANDTL tip loss factor
350 F=1;
351
352 if ( m_bTipLoss || m_bNewTipLoss )
353 {
354     double f=sin (phi/360*2*PI);
355     double g=(outer_radius-m_pos . at ( i ))/m_pos . at ( i );
356     double Ft=2/PI*acos (exp(-blades/2*fabs (g/f))) ;
357     F = F*Ft;
358 }
359
360 if ( m_bRootLoss || m_bNewRootLoss )
361 {
362     double f=sin (phi/360*2*PI);
363     double g=(m_pos . at ( i )-inner_radius)/m_pos . at ( i );
364     double Fr=2/PI*acos (exp(-blades/2*fabs (g/f))) ;
365     F = F*Fr;
366 }
367
368 //here the new tip loss model is implemented
369 F1 = 1;

```

```

370     if (m_bNewTipLoss || m_bNewRootLoss)
371     {
372         if (m_bNewTipLoss)
373         {
374             double f=sin(phi/360*2*PI);
375             double g=(outer_radius-m_pos.at(i))/m_pos.at(i);
376             double Flt = 2/PI*acos(exp(-blades/2*fabs(g/f))*(exp(-0.15*(blades*
377                         m_lambda_local[i]-21))+0.1)));
378             Fl = Fl * Flt;
379         }
380
381         if (m_bNewRootLoss)
382         {
383             double f=sin(phi/360*2*PI);
384             double g=(m_pos.at(i)-inner_radius)/m_pos.at(i);
385             double Flt = 2/PI*acos(exp(-blades/2*fabs(g/f))*(exp(-0.15*(blades*
386                         m_lambda_local[i]-21))+0.1)));
387             Fl = Fl * Flt;
388         }
389         if (m_bNewTipLoss || m_bNewRootLoss)
390         {
391             Cn = Fl * Cn;
392             Ct = Fl * Ct;
393         }
394     }
395
396     //computation of the local thrust coefficient
397     CT = sigma*pow(1-a_a,2)*Cn/pow( sin (phi/360*2*PI) ,2);
398
399     //computation of the axial induction factor
400     if (CT <= 0.96*F)
401     {
402         a_a=1/((4*F*pow( sin (phi/360*2*PI) ,2))/(sigma*Cn)+1);
403     }
404     else
405     {
406         a_a = (18*F-20-3*pow( fabs(CT*(50-36*F)+12*F*(3*F-4)) ,0.5 ))/(36*F
407                         -50);
408     }
409     //computation of the tangential induction factor
410     //a_r=1/((4*cos(phi/360*2*PI)*sin(phi/360*2*PI))/(sigma*Ct)-1);
411
412     a_r = 0.5 * (pow(fabs(1+4/pow(m_lambda_local.at(i),2)*a_a*(1-a_a)) ,0.5)-1)
413         ;      //CW
414
415     //implementation of the relaxation factor
416     if (count <10)
417     {
418         a_a=a_a;
419     }
420     if (count <11)
421     {
422         a_a=0.25*a_a+0.5*a_a_old+0.25*a_a_older;

```

```

422     }
423     else
424     {
425         a_a=relax*a_a+(1-relax)*a_a_old;
426     }
427
428     //computation of epsilon
429     if (fabs(a_a-a_a_old)>fabs(a_r-a_r_old))
430     {
431         eps=fabs(a_a-a_a_old);
432     }
433     else
434     {
435         eps=fabs(a_r-a_r_old);
436     }
437
438
439 }
440
441
442 //now results are appended in the arrays , if the results are computed later ,
443 //during a
444 //turbine simulation a zero as placeholder is appended
445 m_a_axial.append(a_a);
446 m_a_radial.append(a_r);
447 m_Fa_axial.append(F*a_a);
448 m_Fa_radial.append(F*a_r);
449 double Vrel2 = (pow((1-m_a_axial.at(i)),2)+pow(m_lambda_local.at(i)*(1+
450     m_a_radial.at(i)),2));
451 m_p_normal.append(Cn*0.5*Vrel2*m_c_local.at(i));
452 m_p_tangential.append(Ct*0.5*Vrel2*m_c_local.at(i));
453 m_phi.append(phi);
454 m_alpha.append(phi-m_theta.at(i)-pitch);
455 m_CL.append(CL);
456 m_CD.append(CD);
457 m_LD.append(CL/CD);
458 m_Cn.append(Cn);
459 m_Ct.append(Ct);
460 m_F.append(F);
461 m_Reynolds.append(pow(Vrel2,0.5)*m_c_local.at(i));
462 m_DeltaReynolds.append(Reynolds);
463 m_Roughness.append(0);
464 m_Windspeed.append(0);
465 m_Iterations.append(count);
466 m_Mach.append(0);
467 m_circ.append(0);
468
469 //calculation of power coefficient Cp//
470 double power=0, windenergy, thrust;
471 for (int i=0;i<m_pos.size();i++)
472 {
473     power = power + m_pos.at(i)*m_p_tangential.at(i)*deltas.at(i);
474 }
475 power=power*blades*lambda_global/outer_radius;

```

```
476     windenergy= PI/2*pow(outer_radius ,2) ;
477     cp = power/windenergy;
478
479     // if ( cp < 0) cp = 0;                                //CW
480
481
482     //calculation of thrust coefficient Ct//
483     power=0;
484     for ( int i=0;i<m_pos.size () ; i++)
485     {
486         power = power + m_p_normal.at(i)*deltas.at(i)*blades;
487     }
488
489     thrust = PI/2*pow(outer_radius ,2) ;
490     ct = power/thrust;
491
492     // if ( ct < 0) ct = 0;                                //CW
493
494
495 }
496
497
498
499 void BData::Serialize(QDataStream &ar , bool bIsStoring)
500 {
501     int i ,n,j ;
502     float f ;
503     QString strong ;
504
505     if ( bIsStoring )
506     {
507
508         n=m_pos.size () ;
509
510         if ( m_bIsVisible) ar<<1; else ar<<0;
511         if ( m_bShowPoints) ar<<1; else ar<<0;
512         ar << (int) m_Style;
513         ar << (int) m_Width;
514         ar << (float) elements ;
515         ar << (float) rho;
516         ar << (float) epsilon ;
517         ar << (float) iterations ;
518         ar << (float) relax;
519 //         if ( m_bTipLoss) ar << 1; else ar<<0;
520 //         if ( m_bRootLoss) ar << 1; else ar<<0;
521 //         if ( m_b3DCorrection) ar << 1; else ar<<0;
522         WriteCOLORREF(ar ,m_Color);
523         WriteCString(ar ,m_WingName);
524         WriteCString(ar ,m_BEMName);
525         WriteCString(ar ,lambdaglobal);
526         WriteCString(ar ,windspeedStr );
527
528         ar << (int) n;
529
530         for ( i=0;i<n ; i++)
531     {
```

```

532     ar << (float) m_pos[ i ] << (float) m_c_local[ i ] << (float)
533         m_lambda_local[ i ] << (float) m_p_tangential[ i ];
534     ar << (float) m_p_normal[ i ] << (float) m_a_axial[ i ] << (float)
535         m_a_radial[ i ] << (float) m_theta[ i ];
536     ar << (float) m_alpha[ i ] << (float) m_phi[ i ] << (float) m_CL[ i ] << (
537         float) m_CD[ i ] << (float) m_LD[ i ];
538     ar << (float) m_Cn[ i ] << (float) m_Ct[ i ] << (float) m_F[ i ] << (float)
539         m_Reynolds[ i ] << (float) m_DeltaReynolds[ i ] << (float) m_Roughness
540         [ i ] << (float) m_Windspeed[ i ];
541     ar << (float) m_Iterations[ i ] << (float) m_Mach[ i ] << (float)
542         m_Fa_axial[ i ] << (float) m_Fa_radial[ i ] << (float) m_circ[ i ];
543 }
544
545 }
546 else
547 {
548
549     ar >> f;
550     if (f) m_bIsVisible = true; else m_bIsVisible = false;
551     ar >> f;
552     if (f) m_bShowPoints = true; else m_bShowPoints = false;
553     ar >> j;
554     m_Style = j;
555     ar >> j;
556     m_Width = j;
557     ar >> f;
558     elements = f;
559     ar >> f;
560     rho = f;
561     ar >> f;
562     epsilon = f;
563     ar >> f;
564     iterations = f;
565     ar >> f;
566     relax = f;
567 //     ar >> f;
568 //     if (f) m_bTipLoss = true; else m_bTipLoss = false;
569 //     ar >> f;
570 //     if (f) m_bRootLoss = true; else m_bRootLoss = false;
571 //     ar >> f;
572 //     if (f) m_b3DCorrection = true; else m_b3DCorrection = false;
573 ReadCOLORREF(ar ,m_Color);
574 ReadCString(ar ,m_WingName);
575 ReadCString(ar ,m_BEMName);
576 ReadCString(ar ,lambdaGlobal);
577 ReadCString(ar ,windspeedStr);
578
579     ar >> n;
580
581     for (i=0;i<n;i++)

```

```
582 {  
583     ar >> f;  
584     m_pos.append(f);  
585     ar >> f;  
586     m_c_local.append(f);  
587     ar >> f;  
588     m_lambda_local.append(f);  
589     ar >> f;  
590     m_p_tangential.append(f);  
591     ar >> f;  
592     m_p_normal.append(f);  
593     ar >> f;  
594     m_a_axial.append(f);  
595     ar >> f;  
596     m_a_radial.append(f);  
597     ar >> f;  
598     m_theta.append(f);  
599     ar >> f;  
600     m_alpha.append(f);  
601     ar >> f;  
602     m_phi.append(f);  
603     ar >> f;  
604     m_CL.append(f);  
605     ar >> f;  
606     m_CD.append(f);  
607     ar >> f;  
608     m_LD.append(f);  
609     ar >> f;  
610     m_Cn.append(f);  
611     ar >> f;  
612     m_Ct.append(f);  
613     ar >> f;  
614     m_F.append(f);  
615     ar >> f;  
616     m_Reynolds.append(f);  
617     ar >> f;  
618     m_DeltaReynolds.append(f);  
619     ar >> f;  
620     m_Roughness.append(f);  
621     ar >> f;  
622     m_Windspeed.append(f);  
623     ar >> f;  
624     m_Iterations.append(f);  
625     ar >> f;  
626     m_Mach.append(f);  
627     ar >> f;  
628     m_Fa_axial.append(f);  
629     ar >> f;  
630     m_Fa_radial.append(f);  
631     ar >> f;  
632     m_circ.append(f);  
633 //     ar >> f;  
634 //     m_between.append(f);  
635  
636 //     ReadCString(ar, strong);  
637 //     m_polar.append(strong);
```

```
638 //      ReadCString(ar, strong);  
639 //      m_foil.append(strong);  
640 }  
641  
642  
643  
644  
645  
646  
647 }  
648 }  
649 }
```

## A.2 catiasort.py

```
1 #Dies ist ein Skript zum einlesen zylindrische schnitten die von Catia erzeugt  
2      wurden und diese dann sortiert  
3 #in einem ebenen Koordinatensystem in einer .dat datei ausgiebt, die von QBlade  
4      als airfoil eingelesen  
5 #werden kann.  
6 #  
7 #Christian Weiß  
8 #1030438  
9 #Graz 24.11.2013  
10 #für das Institut für thermische Turbomaschinen und Maschinendynamik an der TUGraz  
11 #in einer Bachelorarbeit betreut von Dr.-Ing Oliver Borm  
12 #####  
13 #diese Werte anpassen:  
14  
15 #Speicherort der Eingangsdatei und Ausagabedateiverzeichnis:  
16 #eingangsdatei:  
17 pathin="/media/KINGSTON/bakk/data/WEISS_PROPELLER4_1_1/"  
18 #augabeordner für die .dat (der Ordner muss bereits angelegt sein!)  
19 pathout="/media/KINGSTON/bakk/data/slicedfoils/testt/"           #mit / am Ende  
20  
21 #Schnittparameter:( die entsprechenden Dateien müssen vorhanden sein )  
22 startradius=30  
23 schrittweite=5  
24 endradius= 220  
25  
26 #Aerodynamikdaten:      (standards mit http://www.peacesoftware.de/eingabewerte/luft  
27 .html für 1 bar und 25°C berechnet)  
28 rho=1.168                 #Dichte= 1.168[kg/m^3]  
29 nue=15.821917808219e-6    #kinematische Viskosität= 15.821917808219[m^2/s]  
30 van=50                    #Fluggeschw= 22–77 [m/s]  
31 n=9000                     #Drehzahl= 9000 [rpm]  
32 #kontrollieren ob alle obigen werte auf die aktuelle Anwendung angepasst sind!  
33 #####  
34  
35 import math  
36 import os  
37 import sys  
38  
39 modellname=pathin.strip().split("/")[-2].split(".")[0]  
40  
41 #Die log datie wird auch ins Ausgabeverzeichnis gespeichert und gibt zum einen ü  
42      ber Fehler im Programm auskunft  
43 #und listet für die Simulation wichtige Parameter für jedes Profil auf  
44 logpath=pathout+modellname+".log"  
45 log=open(logpath,"w")  
46 log.write("log for slicing the model: "+modellname)  
47 log.write("\nstored in: "+pathin+"\nslices stored in: "+pathout)  
48 log.write("\nStartradius: "+str(startradius)+"\nSchrittweite: "+str(schrittweite)  
49      +"\nEndradius: "+str(endradius))  
50 log.write("\nrho_air= "+str(rho)+"\nnü_air= "+str(nue))  
51 log.write("\nFluggeschw= "+str(van)+"\nn= "+str(n))
```

```

50 log . write ( " \nwritten by: "+os . path . basename ( sys . argv [ 0 ] ) )
51 log . write ( " \n\ncreatet slices:\nradius chord Phi
      reynolds Vrel mach" )
52
53 radius=startradius
54 scale=1
55 while radius<=endradius:
56
57     temppath=pathin+str ( radius ) +".txt"
58     checkempty=False
59     foilin=open ( temppath , " r " )
60     foilin . seek ( 0 )
61     if foilin . read ( 5 ) =="":
62         checkempty=True
63     foilin . seek ( 0 )
64     if foilin . read ( 5 ) =="\n":
65         checkempty=True
66     if checkempty:
67         print ( str ( radius ) +"Error while reading .txt" )
68         log . write ( " \n"+str ( radius / scale ) +" Error while reading .txt" )
69     else:
70         foilin . seek ( 0 )
71         #flag für die erste Zeile , die ignoriert werden kann
72         ii=0
73         #Datei einlesen
74         punkte=[]
75         for line in foilin :
76             if ii==0:
77                 ii=1
78             else:
79                 rawline=line . strip () . split ( "\t" )
80                 temp2=float ( rawline [ 1 ] . replace ( " , " , ". " ) ) #sollte die x-Achse
81                     sein
82                 temp3=float ( rawline [ 2 ] . replace ( " , " , ". " ) ) #sollte die y-Achse
83                     sein
84                 temp1=float ( rawline [ 3 ] . replace ( " , " , ". " ) ) #sollte die z-Achse
85                     sein
86                     #abwickeln
87                     tempx=((temp1**2+temp2**2)**0.5)*math . atan ( temp1/temp2 )
88                     punkte.append (( tempx , temp3 ))
89                     #zähler für die anzahl der Koordinaten. Diese dürfen für QBlade max 300
90                     sein
91                     linecount=len ( punkte )
92
93                     #Suche die zwei Punkte die den grüßten Abstand zueinander haben
94                     dis=0.
95                     a=0
96                     b=0
97                     while a<=linecount -1:
98                         b=0
99                         while b<=linecount -1:
100                             if dis<=((punkte [ a ] [ 0 ] -punkte [ b ] [ 0 ] ) **2+(punkte [ a ] [ 1 ] -punkte [ b
101                                 ] [ 1 ] ) **2)**0.5:
102                                 dis=((punkte [ a ] [ 0 ] -punkte [ b ] [ 0 ] ) **2+(punkte [ a ] [ 1 ] -punkte [ b
103                                     ] [ 1 ] ) **2)**0.5
104                                 pkta=punkte [ a ]

```

```
99         pktb=punkte [b]
100        b+=1
101        a+=1
102        print(a)      #eine Ausgabe rein um zu sehen ob das Programm noch
103        arbeitet
104        if pkta[0]>pktb [0]:
105            kante=pkta
106            vorne=pktb
107        else:
108            kante=pktb
109            vorne=pkta
110
111        #gerade in der mitte des profils (vorderster und hinterster Punkt)
112        k=(-vorne[1]+kante[1])/(-vorne[0]+kante[0])
113        d=vorne[1]-k*vorne[0]
114
115        #Aufteilen in Punkte die über bzw unter der Geraden liegen
116        oben=[]
117        unten=[]
118        obenlen=0
119        for line in punkte:
120            if line[1]>=(d+k*line[0]) or 0.0000000001 > -line[1]+(d+k*line[0]):
121                oben.append(line)
122                obenlen+=1
123            else:
124                unten.append(line)
125        #die obere Hälfte wird sortiert in den von der vorderkante aus immer der n
126        #ächstgelegene Punkt gesucht wird
127        #bis die Hinterhanke erreicht ist
128        leno=len(oben)
129        iii=0
130
131        target=vorne
132        sortoben=[]
133        breakflag=False
134        while iii<leno:#für die jeweiligen plätze in der neuen liste
135            if breakflag:
136                break
137            iii+=1
138            bestabstand=999999999999999999999999
139            while v<len(oben):#zum suchen des punkts für den platz
140                abstand=((target[0]-oben[v][0])**2+(target[1]-oben[v][1])**2)**0.5
141                if abstand<=bestabstand:
142                    tari=v
143                    bestabstand=abstand
144                v+=1
145            sortoben.append(oben[tari])
146            target=oben[tari]
147            del(oben[tari])
148            if target[0]==kante[0] and target[1]==kante[1]:
149                break
150                breakflag=True
151        #die verbeibenden Punkte von oben sind eigentlich punkte der Unterkante
152        die durch eine starke Wölbung
```

```

152     #des Profils über die Verbindungsgerade ragen
153     for line in oben:
154         unten.append(line)
155
156     iv=0
157     lenu=len(unten)
158     sortunten=[]
159     while iv <lenu:#für die jeweiligen plätze in der neuen liste
160         vi=0
161         tarii=0
162         iv+=1
163         bestabstand=999999999999999999999999
164         while vi<len(unten):#zum suchen des punkts für den platz
165             abstand=((target[0]-unten[vi][0])**2+(target[1]-unten[vi][1])**2)
166                         **0.5
167             if abstand<=bestabstand:
168                 tarii=vi
169                 bestabstand=abstand
170                 vi+=1
171                 sortunten.append(unten[tarii])
172                 target=unten[tarii]
173                 del(unten[tarii])
174                 for line in sortoben:
175                     sortunten.append(line)
176 #verschieben von vorne in den ursprung
177                 output=[]
178                 for line in sortunten:
179                     output.append((line[0]-vorne[0],line[1]-vorne[1]))
180 #output.append((line[0],line[1]))    #würde die Verschiebung aus dem
181 #Programm nehmen
182
183 #drehen des profils in die xAchse
184 winkel=math.atan(k)
185 winkeldeg=winkel*180/math.pi+90
186 sortunten=[]
187 for line in output[::-1]:
188     sortunten.append((line[0]*math.cos(-winkel)-line[1]*math.sin(-winkel),
189                         line[0]*math.sin(-winkel)+line[1]*math.cos(-winkel)))
190 #sortunten.append((line[0],line[1]))          #würde die Drehung aus dem
191 #Programm nehmen
192
193 # schreiben der oberen und der unteren kurvenhälfte in die Ausgabedatei
194 # und anschlie ßend noch den ersten punkt wieder einfügen um
195 # eine geschlossene Kurve zu erhalten
196 dateiname=pathout+str(radius/scale)+"_"+modellname+".dat"
197 foilout=open(dateiname,"w")
198
199 outline=str(radius/scale)+" "+str(modellname)+"\n"
200 foilout.write(outline)
201
202
203

```

```

204
205     # /dis normiert die Größe so dass die Länge des foils genau 1 hat
206     viii=0      #zähler für die ite zeile die geschrieben werden soll
207     vii=0      #ich will die erste und letzte zeile immer schreiben
208     for line in sortunten:
209         if viii==0 or viii==len(sortunten):
210             outline=str(line[0]/dis)+" "+str(line[1]/dis)+"\n"
211             foilout.write(outline)
212             viii+=1
213         else:
214             if viii==(linecount-2)/298:
215                 outline=str(line[0]/dis)+" "+str(line[1]/dis)+"\n"
216                 foilout.write(outline)
217                 viii=0
218             else:
219                 viii+=1
220     if linecount>300:
221         print("Achtung bei Radius "+str(radius/scale)+" sind mehr als 300
222             punkte vorhanden.\n Es gehen Daten verloren .")
223
224     outline=str(sortunten[0][0]/dis)+" "+str(sortunten[0][1]/dis) #
225             schreibt den ersten punkt noch ein mal ans ende
226     foilout.write(outline)
227     foilin.close()
228     foilout.close()
229     u=(van**2+(radius/scale/1000.*2*math.pi*n/60)**2)**0.5
230     dd=dis/scale/1000.
231     reyn=u*dd/nue
232
233     log.write("\n"+str(radius/scale)+" "+str(dis/scale)+" "+str(
234             winkeldeg)+" "+str(reyn)+" "+str(u)+" "+str(u/343))
235     if linecount>300:
236         log.write(" Achtung es sind mehr als 300 punkte vorhanden. Es
237             gehen Daten verloren ")
238     radius+=schrittweite
239
240     log.close()
241
242     print " fertig "
243
244 #####/#####
245 #TODO:      Modellnamen richtig bezeichnen
246 #####/#####
247 #V1:      Dieses Programm entstand aus dem programm cut3.py und
248 #              soll die nun mit Catia erstellten Profile sortieren
249 #####/#####
250 #V2:      Modellname wird aus dem Einlesefolder genommen
251 #              einige verbessерungen in der Ausgabe
252 #              bereinigen von variablenüberschreibungen
253 #####/#####
254 #V3:      Korrektur der Kommentare
255 #              van und n werden auch in der log datei ausgegeben
256 #              korrekte Winkelangabe

```



# Literaturverzeichnis

- [1] <http://www.rc-network.de/forum/archive/index.php/t-1709.html>
- [2] *airfoiltools*. <http://airfoiltools.com/>
- [3] *Makro zum Exportieren von Punkten in Catia*. [ww3.cad.de/foren/ubb/Forum133/HTML/002640.shtml#l000003](http://ww3.cad.de/foren/ubb/Forum133/HTML/002640.shtml#l000003)
- [4] *python*. <http://www.python.org/>
- [5] *Blade element momentum theory*. [http://en.wikipedia.org/wiki/Blade\\_Element\\_Momentum\\_Theory](http://en.wikipedia.org/wiki/Blade_Element_Momentum_Theory). Version: 2014
- [6] ABEDI, H. : *Aerodynamic Loads On Rotor Blades*. Version: 2011. <http://publications.lib.chalmers.se/records/fulltext/147853.pdf>
- [7] BRENN, G. ; MEILE, W. : *Strömungslehre und Wärmeübertragung 1*, 2012
- [8] BUHL, M. L.: *A New Empirical Relationship between Thrust Coefficient and Induction Factor for the Turbulent Windmill State*. <http://www.nrel.gov/docs/fy05osti/36834.pdf>. Version: August 2005
- [9] DRELA, M. ; YOUNGREN, H. : *XFOIL 6.94 User Guide*, 2001
- [10] MARTEN, D. ; WENDLER, J. : *QBlade*. <http://sourceforge.net/projects/qblade/>. Version: 2013
- [11] MARTEN, D. ; WENDLER, J. : *QBlade Guidelines*, 2013
- [12] MORIARTY, P. ; HANSEN, A. C.: *AeroDyn Theory Manual*. Version: January 2005. <http://www.nrel.gov/docs/fy05osti/36881.pdf>
- [13] SCHENK, H. : *PropCalc*. <http://www.drivecalc.de/PropCalc/>. Version: 2007