

Analysis of Flow around a Ship Propeller using OpenFOAM

Eamonn Colley

Supervised by Dr Tim Gourlay

October 2012

Honours Dissertation

Curtin University

Perth, Western Australia

Abstract

This dissertation analyses a propeller based off the coordinates of model KCD 32 (Emerson and Sinclair, 1967). Using Matlab to recreate the blade and mesh a three dimensional model, analysis was conducted. The study was completed using a computational program, OpenFOAM, and the pressure distribution and effects of cavitation were compared to experimental results. Reasonable results were produced such that the thrust, torque and efficiency trends were in good agreement with experimental data. It was also found that OpenFOAM had good prediction of when and where cavitation would occur on the propeller.

Acknowledgements

I would like to thank Tim Gourlay for the help and knowledge that he has given me over the entire duration of my project.

The Lecturers and staff at Curtin University have been most kind and assisted me when I have needed it.

I would also like to thank my family who have supported me throughout my whole life. My parents have been especially helpful in guiding me on a path that I truly enjoy.

Table of Contents

Analysis of Flow around a Ship Propeller using OpenFOAM	1
Abstract.....	2
Acknowledgements.....	2
Table of Contents.....	3
Figures.....	5
1.0 Introduction	6
1.1. Characteristics of Foils	6
1.2. Ship Propeller Theory.....	9
1.3. Cavitation	10
1.3. Nomenclature	11
1.5. CFD	12
2.0 Aims.....	13
3.0 Literature Review.....	14
4.0 Modelling the Propeller Blade	15
4.1. Getting the coordinates.....	15
4.2. Rotating the Sections.....	15
4.3. Fitting a Leading Edge	16
4.4. Tip, Root and Surface of the Blade	19
4.5. Generating the .STL File	19
5.0 Simulation in OpenFOAM	21
5.1. Constructing the BlockMesh- Single Blade	21
5.2. Meshing the Single Blade with the Block.....	21
5.3. Meshing the Complete Propeller.....	22
5.4. SRFSimpleFoam.....	25
5.5. MRFSimpleFoam	25
6.0 Analysis	28
6.1. Input Values	28
6.2. Computational Results – Thrust and Torque.....	28
6.3. Computation Results – Pressure Distribution.....	33
7.0 Conclusion.....	45
8.0 Recommendations	46

9.0	References	47
10.0	Appendix	48
	A1: SnappyHexMesh Changes.....	48
	A2: surfaceMesh.m	49
	A3: MainFunc.m	50
	A4: Interpolate.m	53
	A5: rotating.m	55
	A6: lealedge.m	56
	A7: surCreate.m.....	57
	A8: STLWriting.m.....	58
	A9: addBlades.m.....	59

Figures

Figure 1: An image showing the various components of a foil.....	6
Figure 2: Flow around a foil with no circulation and circulation (Celli, 1997)	7
Figure 3: Streamline flow around a foil at 6 degrees angle of attack.	8
Figure 4: Streamline flow around a foil at 16 degrees angle of attack.	8
Figure 5: Image of a fixed 3-blade propeller.....	9
Figure 6: Diagram of the local angle of attack in relation to the flow of water.	10
Figure 7: Table of coordinates (A) for the blade, short diagram and labelling (B) and pitch (C) (Emerson and Sinclair, 1967).	15
Figure 8: Fitting a circle to the points A,B,D,E.	17
Figure 9: Seven sections of the blade where Z is the radius outward from the centreline in inches. .	18
Figure 10: A blade of a propeller, meshed with triangular surface elements.	20
Figure 11: The block that was created initially via OpenFOAM.....	21
Figure 12: The blade now meshed with the block.....	22
Figure 13: Showing the complete propeller with a triangle mesh.	23
Figure 14: Representation of the blockMesh in paraView.	24
Figure 15: Jagged edges on the propeller blade(s).....	25
Figure 16: Setup of MRFSimpleFoam.....	26
Figure 17: Graphical representation of the experimental results (Emerson & Sinclair, 1967).....	30
Figure 18: The relationship of the thrust coefficient and J.....	31
Figure 19: The relationship of the torque coefficient and J.....	32
Figure 20: The relationship of the efficiency and J.	33
Figure 21: Back side of the propeller, J value of 0.40	34
Figure 22: Face side of the propeller, J value of 0.40	34
Figure 23: Back side of the propeller, J value of 0.45	35
Figure 24: Face side of the propeller, J value of 0.45	35
Figure 25: Back side of the propeller, J value of 0.50	36
Figure 26: Face side of the propeller, J value of 0.50	36
Figure 27: Back side of the propeller, J value of 0.55	37
Figure 28: Face side of the propeller, J value of 0.55	37
Figure 29: Back side of the propeller, J value of 0.60	38
Figure 30: Face side of the propeller, J value of 0.60	38
Figure 31: Back side of the propeller, J value of 0.65	39
Figure 32: Face side of the propeller, J value of 0.65	39
Figure 33: Back side of the propeller, J value of 0.70	40
Figure 34: Face side of the propeller, J value of 0.70	40
Figure 35: Back side of the propeller, J value of 0.75	41
Figure 36: Face side of the propeller, J value of 0.75	41
Figure 37: Back side of the propeller, J value of 0.80	42
Figure 38: Face side of the propeller, J value of 0.80	42
Figure 39: Closer inspection of the back side of the blade, J = 0.40	43
Figure 40: Closer Inspection of the back side of the blade, J = 0.80.....	44
Figure 41: ScreenCapture from SnappyHexMesh dictionary.....	48

1.0 Introduction

This project investigates propellers and their properties using computational fluid dynamics. A propeller blade can be considered as a stack of cross-sections; each cross-section acting as a lifting surface. A foil is similar to that of an aeroplane wing, such that, it obeys the same laws of physics but with different parts of the propeller blade moving at different speeds and rotational flow. These laws enable us to examine how the flow moves around the foil, and hence propeller blades, as the conditions change (White, 2008).

1.1. Characteristics of Foils

Figure 1 shows a cross-section through a propeller blade. The coordinates of a cross-section/foil are usually given as x (distance along chord) and y (Upper/Lower Camber) and z (the radius away from the hub centreline).

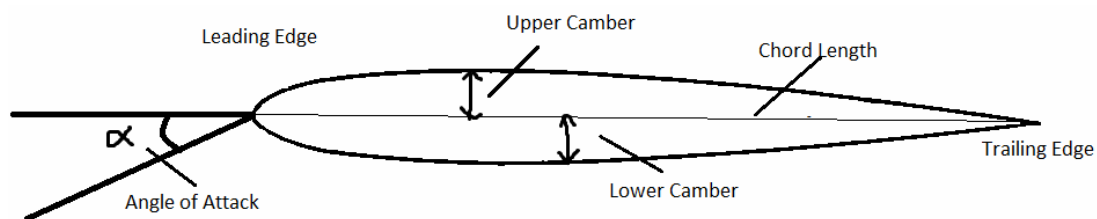


Figure 1: An image showing the various components of a foil.

The point where the fluid is at zero velocity and also where the fluid divides into two is called the stagnation point. This point is related to the angle that the fluid hits the foil. As this angle increases, the stagnation point becomes further down the lower edge. Bernoulli showed that as pressure (P) increases, the velocity (V) of the fluid decreases, and also vice-versa, as shown in Equation 1.

Equation 1: Bernoulli's equation

$$\frac{V^2}{2} + gz + \frac{P}{\rho} = \text{Constant}$$

Initially as the fluid flows around the foil, it has no circulation. That is, the flow leaving the trailing edge of the airfoil is not smooth and there is a stagnation point on the upper edge (Figure 2).

However, the Kutta-Joukowski condition states that the circulation will adjust itself via a starting vortex circulating in the opposite direction (in the wake) and the flow will leave the trailing edge smoothly.

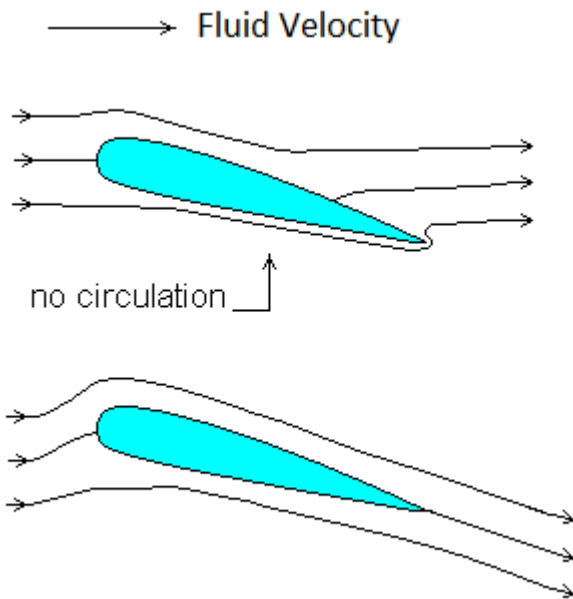


Figure 2: Flow around a foil with no circulation and circulation (Celli, 1997)

As the angle of attack increases, the fluid has further to travel around the upper edge, as the stagnation point is now on the lower edge, and the flow will be pushed around the upper edge at a higher velocity to conserve the Kutta-Joukowski condition. Conservation of Bernoulli's equation requires that the pressure decreases where the velocity increases. Generally, on the lower surface there is lower velocity with higher pressure and on the upper surface there is higher velocity with lower pressure. Figure 3 shows a flow that was computed in OpenFOAM of a foil at 6 degrees angle of attack. The red arrows indicating high velocity are at a low pressure region (blue lines) and there is high pressure at the stagnation point (red lines).

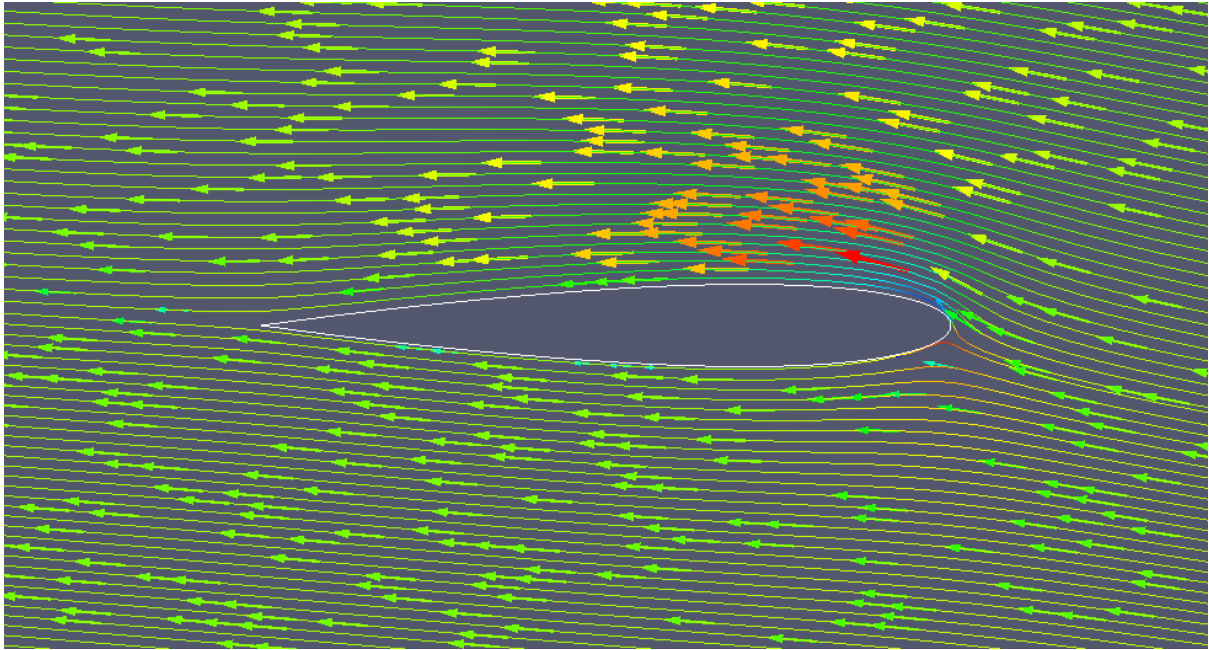


Figure 3: Streamline flow around a foil at 6 degrees angle of attack.

However, if the angle of attack increases too much, then the flow no longer follows the foil and more drag is produced than lift stall occurs (White, 2008). Figure 4 shows a large separation from the foil and stalling.

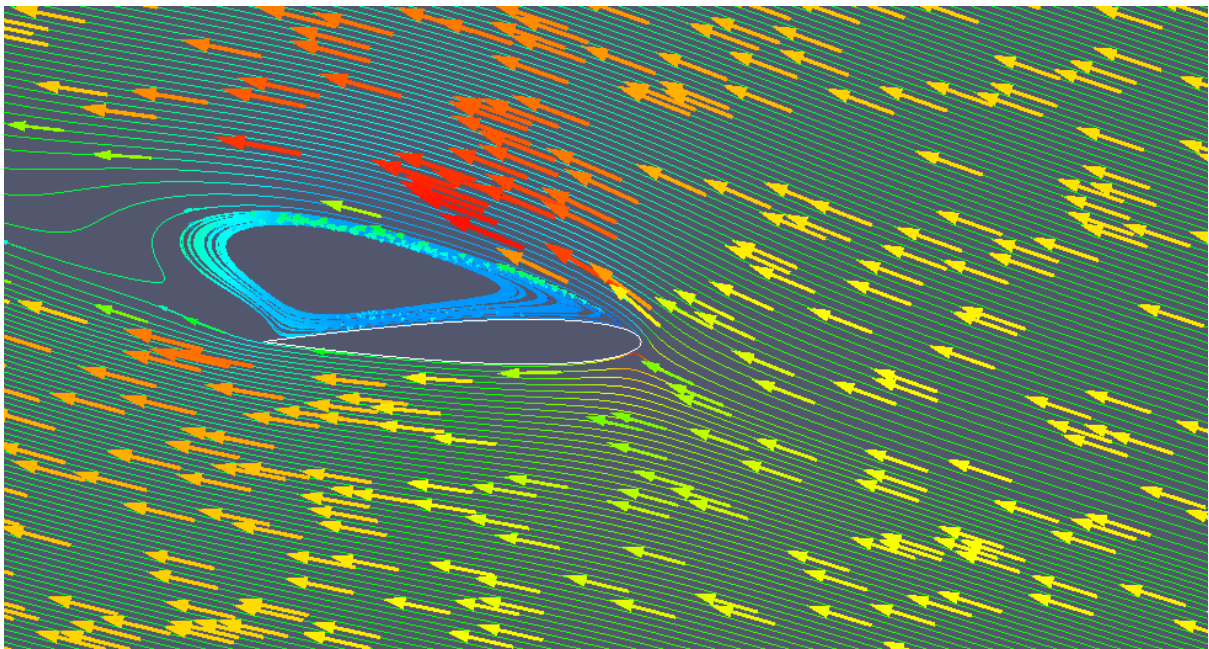


Figure 4: Streamline flow around a foil at 16 degrees angle of attack.

1.2. Ship Propeller Theory

Ship propellers come in various shapes and sizes. These range from different shaped blades, thickness, size and angle. Moreover, the amount of blades attached to the hub can make a variance in the flow and hence lift. Three blades is usually the best compromise between balance, blade area and efficiency. More than three blades help reduce the vibration and are often used on larger ships. However, the ideal amount of propeller blades is one, but is very unbalanced and generally not used.

Propellers are effectively a simple design and only have a few key components (Figure 5). The hub is the centre piece which when sufficiently strong holds the blades in place. The blades are curved in order to slice the water (leading edge) and provide lift to move the ship forward (Gerr, 1989). The blade face which can be seen from the aft of the ship is at high pressure. The blade back, which is in the direction of the ships' motion, is the low pressure side.

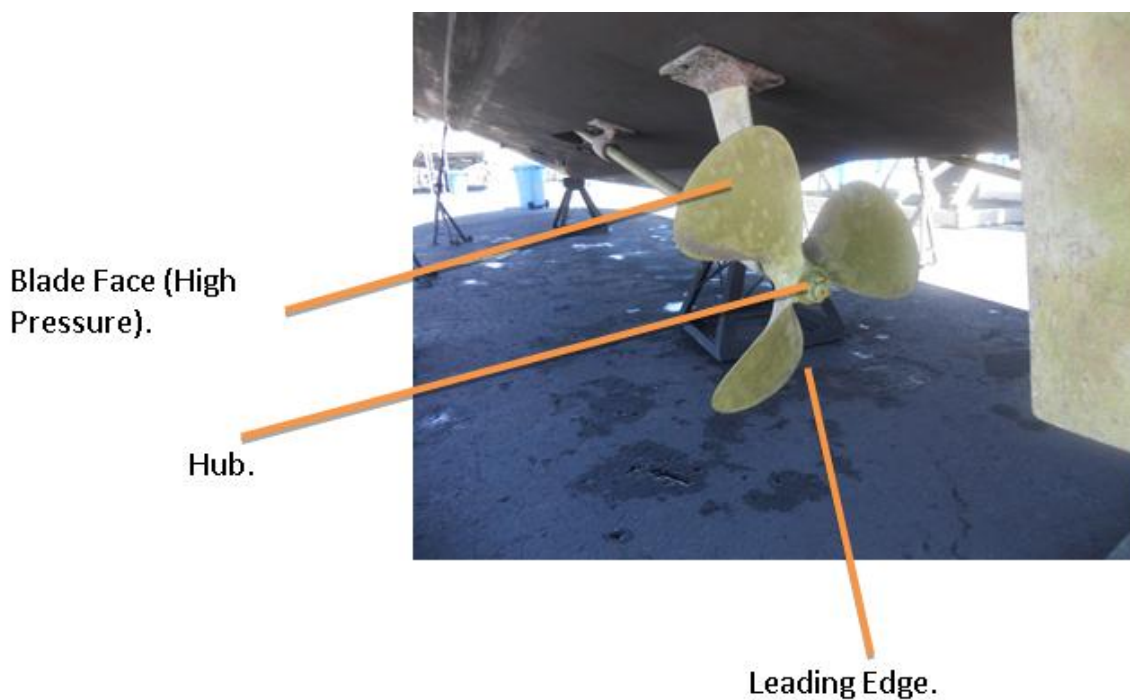


Figure 5: Image of a fixed 3-blade propeller.

An important property of the propeller blade is the local angle of attack. It is dependent on the leading edge of the blade as it cuts through the water as shown in Figure 6. As the boat moves forward through the water, there will be flow over the blade back due to the speed of the boat. However, the local angle of attack is what will define the amount of lift that will be generated. As the propeller spins faster, there will be a larger local angle of attack and this will provide more lift. However, as the angle of attack increases, the flow will also begin to separate and high speeds will produce cavitation.

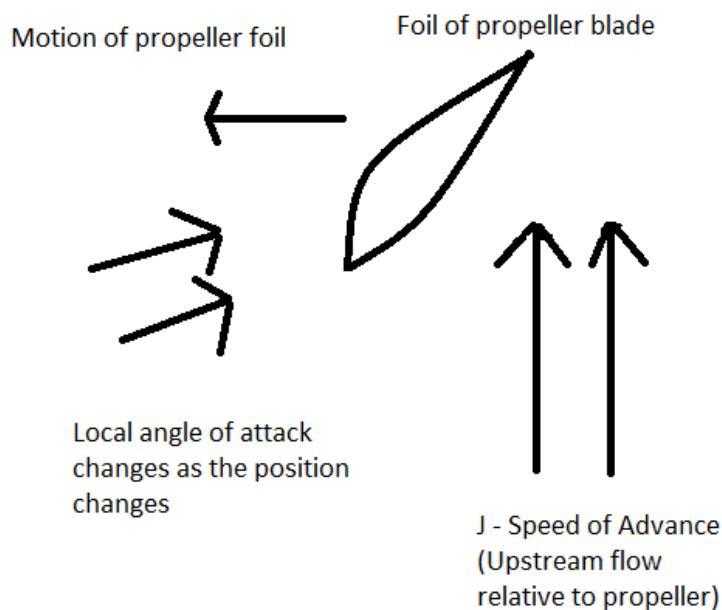


Figure 6: Diagram of the local angle of attack in relation to the flow of water.

1.3. Cavitation

Cavitation is vapour bubbles forming at very low pressures. These then implode downstream when damage can occur. As these bubbles implode (collapse), they force energetic liquid into very small volumes: creating a shock wave (Gerr, 1989). There are two types of cavitation which are tip cavitation and blade cavitation. Tip cavitation is at the tip where the flow goes from a high pressure region to a low pressure region and creates a vortex. This type of cavitation is not necessarily bad for the blade/ship as it washes downstream. Blade cavitation is at the lowest pressure area of the blade. As the vapour bubbles implode on this area, it damages the surface (Eisenberg, 2008). This type of cavitation is one to avoid as it affects the propeller by corrosion, loss of efficiency, noise and vibrations and usually occurs if the local angle of attack is too large, e.g. the propeller is spinning too fast. However, in order to gain sufficient lift and operate efficiently, the propeller needs to be at the level that cavitation starts to happen.

1.3. Nomenclature

Equation 2: Propeller advance coefficient (Breslin & Andersen, 1994)

$$J = \frac{V_a}{nD}$$

The propeller advance coefficient, J , is propeller advance velocity on the revolution per seconds multiplied by the diameter of the propeller. When J is large, there is a small local angle of attack and less chance of cavitation. When J is small there will be a larger local angle of attack and an increased chance of cavitation occurring.

Equation 3: Cavitation number (Breslin & Andersen, 1994)

$$\sigma = \frac{p_\infty - p_v}{\frac{1}{2}\rho(nD)^2}$$

The cavitation number, σ , relates to the ambient pressure and vapour pressure over the density of water, revolutions and diameter of the propeller. If σ is large there won't be cavitation and at small cavitation numbers, there is an increased chance of cavitation.

Equation 4: Thrust coefficient (Breslin & Andersen, 1994)

$$K_T = \frac{Thrust}{\rho n^2 D^4}$$

The thrust coefficient, K_T , is a measure of the force output of the propeller, where thrust can be calculated from the lift coefficient.

Equation 5: Torque coefficient (Breslin & Andersen, 1994)

$$K_Q = \frac{Torque}{\rho n^2 D^5}$$

The torque coefficient, K_Q , is a measure of the torque input that is required to rotate the propeller where torque can be calculated from the moment coefficient.

Equation 6: Efficiency (Breslin & Andersen, 1994)

$$\eta = \frac{K_T J}{2 \pi K_Q}$$

The efficiency is essentially a ratio of power input and power output from the propeller.

1.5. CFD

CFD is a Computational Fluid Dynamics program that uses numerical methods to solve and analyse fluid problems. There are generally three parts to solving a problem computationally: Pre-processing, processing (solving) and post-processing.

During the pre-processing, the boundary conditions are defined; specifying the way the fluid should move. The object and surrounding surfaces and volumes are then meshed (discrete cells).

The processing stage is done via a solver. OpenFOAM, an open source CFD, allows many different solvers to be used. The solver that will be examined is steady-state solver for incompressible laminar flow. Different sub modules allow for various types of problems to be solved. In this project, the function called MRFSimpleFoam will be used as it allows a reference-frame rotation while conserving momentum, energy and matter; a key requirement in the accurate observation of cavitation on propellers.

Post-processing is where the analysis of the solution is done. OpenFOAM offers paraView, another open source tool to visualise and investigate the problem.

2.0 Aims

The aim for this project is to successfully model a propeller in OpenFOAM. If this is accomplished, it will allow for good comparison between experimental and computational results. Moreover, further analysis can be conducted in how the flow is affected in different conditions. In order to complete this aim, there are key stages that need to be finished first:

-Model the propeller blade: After acquiring offsets of the blade, they will be mapped and smoothed in Matlab.

-Mesh the propeller blade: Using a meshing program, the surface of the blade will be split into much smaller domains (discrete cells), generally made up of geometric primitive shapes; hexahedra and tetrahedral (CFD-Online, 2012).

-Model and mesh the propeller including all blades and hub: Using programs to model the whole propeller, the surface will be meshed. Using OpenFOAM, the volume will then be meshed to make it a three dimensional problem.

-Set up boundary conditions in OpenFOAM: Using a 360 degree boundary and allow a rotating reference frame.

-Compute flow analysis in OpenFOAM and compare with experimental texts.

3.0 Literature Review

In fluid dynamics, an analytical approach to solve the flow equations is hard to achieve with good precision due to the complexity. Therefore, many of the researchers use CFD (Computational Fluid Dynamics) to simulate the flow and compare to experimental data. In CFD, setting up the boundary conditions correctly is important to simulate the flow as accurate as possible. However, this is not a simple task to do. A large number of things can be changed such as the turbulence model used, number of cells that make up the mesh and the solver.

OpenFOAM is an open source CFD and is the modelling program used in this dissertation. This program is a “free, open source CFD software package... to solve complex fluid flows” (OpenFOAM user guide, 2010). One of the main features it provides is the ability to solve multiphase fluids such as the water and vapour that will be used for a propeller. As it is open source, there are forums in which other users can add to the discussions. These topics also provide useful tutorials in which test cases can be conducted.

There are limited amounts of research papers related to propeller flow analysis due to patented designs. Many of the public papers also don't use standard series propellers with known offsets which makes it hard to reproduce results and/or compare. This is evident in Watanabe et al. (2003) and Bensow (2010). However, these two papers are useful as they do provide insight on how they constructed their mesh, each with a different turbulence model and CFD program.

The key paper that was studied and used in this dissertation was Emerson and Sinclair (1967). Emerson and Sinclair used a cavitation tunnel to run experiments on 5- and 6-Bladed propellers. This was one of the few papers that provided coordinates of the propeller they were using and associated results. This enables a model to be produced using OpenFOAM and a good comparison to be made. It is also suggested in the text that cavitation depends to some extent on the air content of the water “without any kind of proof in connection with tests in nonhomogenous flow... there is always a considerable amount of entrained air in the water”, but as OpenFOAM only models 100% air or 100% water, have to allow for this. It may also be something that can be looked at in further detail. This also leads to ensure that all variables be controlled as best as possible which is outlined in the 23rd ITTC report (2002) as many factors can affect the results.

Other main texts that were used are Watanabe et al. (2003) and Bensow (2010) that simulate cavitating flows with RANS and LES CFD code respectively. The two major differences in these papers is the turbulence model used and the way the boundary conditions are set up. Watanabe (et al.) uses the RANS (Reynolds Averaged Navier-Stokes) turbulence model and has the water rotating around a motionless propeller blade. Using steady and unsteady conditions it was found that the results were in good agreement with experimental values. Bensow used the LES (Large Eddy Simulation) in OpenFOAM to gain an understanding of flow around 3 objects: one including a propeller. However, even though simulations are made, no results or data is in the text, but does provide a good appreciation of how LES can work in OpenFOAM.

4.0 Modelling the Propeller Blade

4.1. Getting the coordinates.

The coordinates of the blades were based on a 5-bladed right hand propeller; model KCD 32 (Emerson and Sinclair, 1967). The coordinates are set up as shown in Figure 7. There are 7 sections (foils) at varying pitch and distance from the hub.

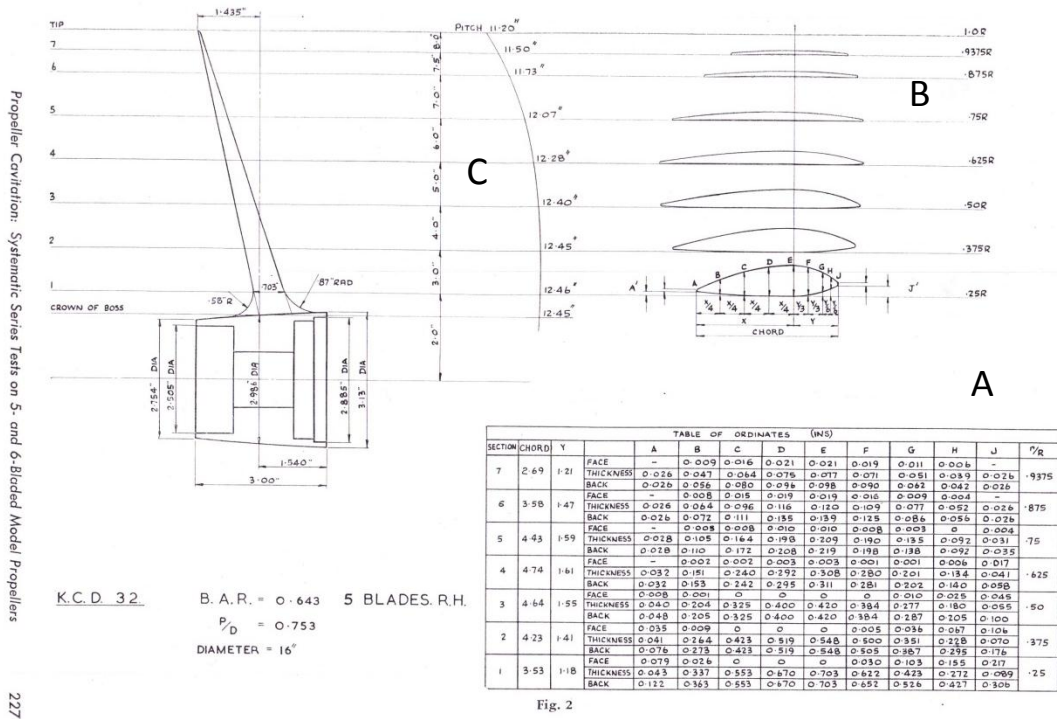


Figure 7: Table of coordinates (A) for the blade, short diagram and labelling (B) and pitch (C) (Emerson and Sinclair, 1967).

Matlab, a product made by MathWorks, is mathematical based computing software that is used in this project. The coordinates of each section were input into x and y vectors in Matlab. A function (interpolate.m) was created to interpolate a specified number of points for the x and y values to make the curves smoother. The numbers of points are able to be changed in mainFunc.m

4.2. Rotating the Sections.

Each section of the blade has a different pitch, measured in inches. The pitch is defined as the distance moved forward in one rotation assuming no slip and is given by a trigonometric ratio (Gerr,

1989). Therefore, the angle that each section is to be rotated by can be calculated by Equation 7: where r is the distance from the hub.

Equation 7: Angle of rotation

$$\theta = \tan^{-1}\left(\frac{Pitch}{2 * \pi * r}\right)$$

Each section is then systematically rotated anti-clockwise in a Matlab function (rotating.m) using a two dimensional rotation matrix.

4.3. Fitting a Leading Edge

From the coordinates, no information is given about the leading edge. For good flow around the foil and obtaining comparable results, the leading edge should be very smooth but curved (Bensow, 2010). NACA foils base the leading edges on circles as it reduces separation and forestalls stalling (McCormic, 1999). Therefore, want a circle that will pass through points B and D and a gradient equal to the magnitude of the average gradient of A:B and E:D (Abbot, 1958). This is shown for the first section (Figure 8).

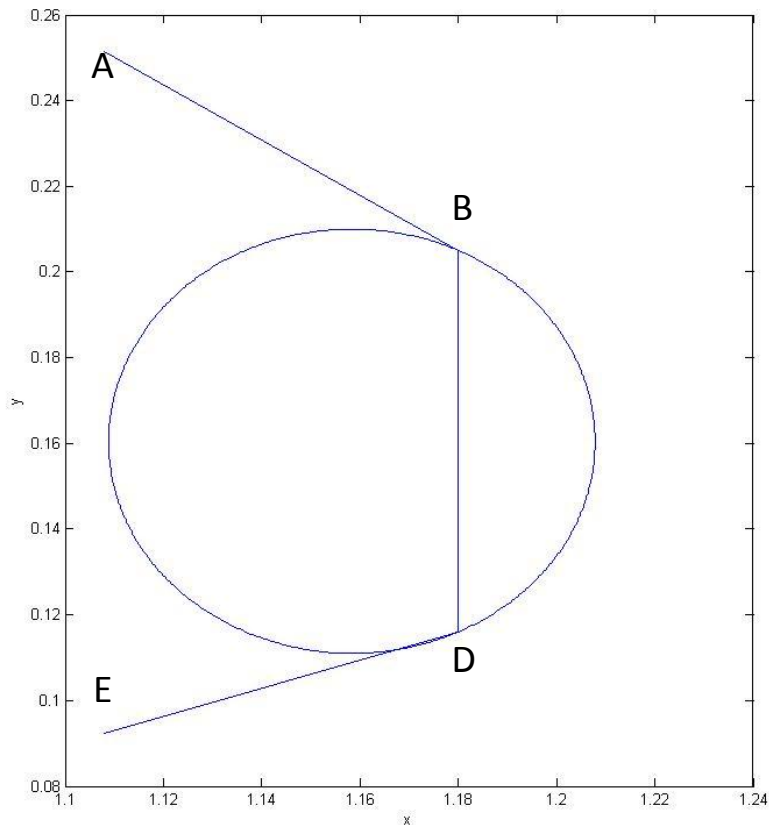


Figure 8: Fitting a circle to the points A,B,D,E.

A function was written in Matlab (leadedge.m) to calculate the points B to D (Figure 8) on the circle and replace them for the direct path (straight line) from B to D. The general equation of the circle is given as Equation 8.

Equation 8: General equation of a circle

$$(x - a)^2 + (y - b)^2 = c^2$$

Knowing the coordinates at points A, B, D and E, the gradient of each was calculated and therefore the magnitude of the average gradient (m) was obtained. Equation 8 was implicitly differentiated and shown in Equation 9 to find the gradient of a circle at a (x,y) point.

Equation 9: The gradient of a circle

$$y' = \frac{(a - x)}{(y - b)}$$

The magnitude of the average gradient, m , and Equation 9 were then equated to each other at point B. Knowing the x and y values of each point, the b value could be calculated as the midpoint of D and B.

Equation 10: Solving for a

$$a = m * (y - b) + x$$

Equation 11: Solving for c

$$c = \sqrt{(x - a)^2 + (y - b)^2}$$

Having solved the circle equation as described above, x and y points were calculated at regular distances along the circle between B and D and added to the current x and y vectors to complete the section/foil.

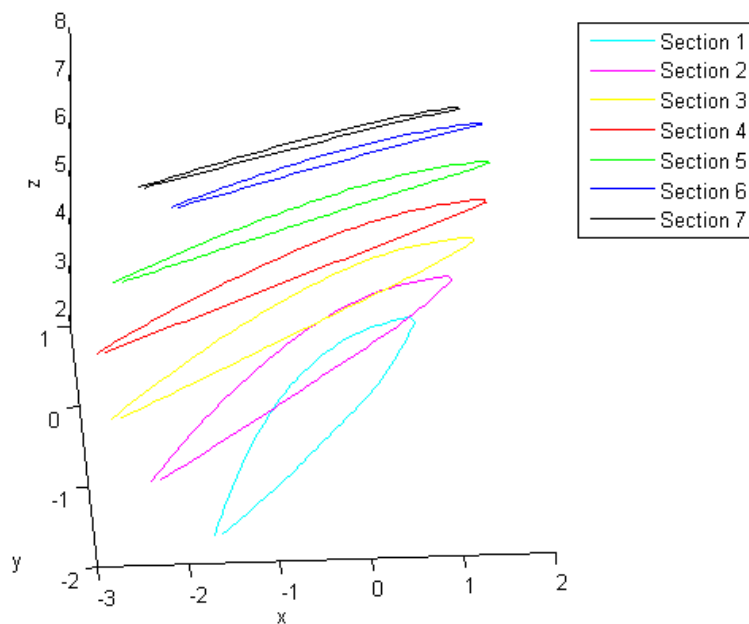


Figure 9: Seven sections of the blade where Z is the radius outward from the centreline in inches.

Figure 9 shows the seven sections of the blade plotted in MatLab. This is a visual representation of the shape the propeller will take.

4.4. Tip, Root and Surface of the Blade

The tip of the blade, which is essentially Section 8, was created using an extrapolation feature of MatLab (surCreate.m). Extrapolating the z value at the leading and trailing edges a linear line could be added so that the propeller blade is now closed at the tip.

The root of the blade (section 0) was extrapolated from the other sections' x and y values. This section is at a height of 1.5 inches, which is below the hub. This is to ensure that the blade meshes correctly with the hub. The function is shown in surCreate.m

To refine the surface of the blade, the z values were interpolated in a MatLab function (surCreate.m) to a specified number of points that can be modified in mainFunc.m.

4.5. Generating the .STL File

The propeller needed to be in a format that can be modified in OpenFOAM. A STL (Standard Tessellation Language) file was the easiest format to convert to from the coordinate system already made.

Firstly, the points are numbered in sequential order starting at section one and going from trailing edge (face side) to leading edge and then back to trailing edge (back side) before moving up the z axis.

Quadrilaterals are then made by joining two adjacent points and two points above (in the z axis). This is iteratively done till all points are part of a quadrilateral. In this process, the trailing edge is now closed. Triangles are created by splitting the quadrilaterals in half. These loops are seen in surfaceMesh.m. The coordinates are also converted to metres (from inches).

The blade is then written to STL file format in STLWriting.m. This function calculates the facet normal of each triangle, which is the cross product of the sides of the triangle, regarding the right-hand rule. A file is then opened for writing and following the STL format, the facet normal is written along with the vertices coordinates for each triangle.

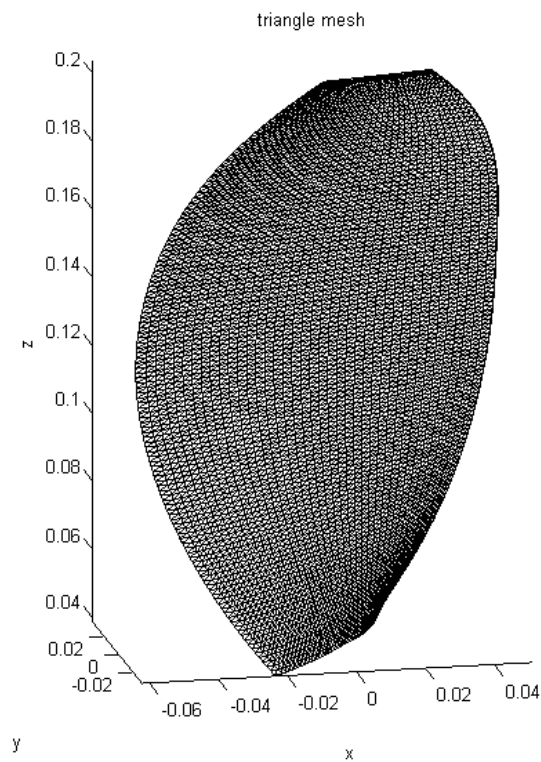


Figure 10: A blade of a propeller, meshed with triangular surface elements.

Figure 10 shows the representation of a blade when meshed with triangular surface elements. The blade can be refined or made coarser by altering the variables at the top of mainFunc.m. This blade has 22 134 surface elements.

5.0 Simulation in OpenFOAM

5.1. Constructing the BlockMesh- Single Blade

In OpenFOAM, the BlockMesh is a three dimensional, 8-cornered block that the flow will be simulated throughout. For a single blade, this will be a 72 degree wedge prism that (1 out of 5 blades) has an inlet, outlet, top and hub. There will also be two sides to the block that will have periodic boundary conditions.

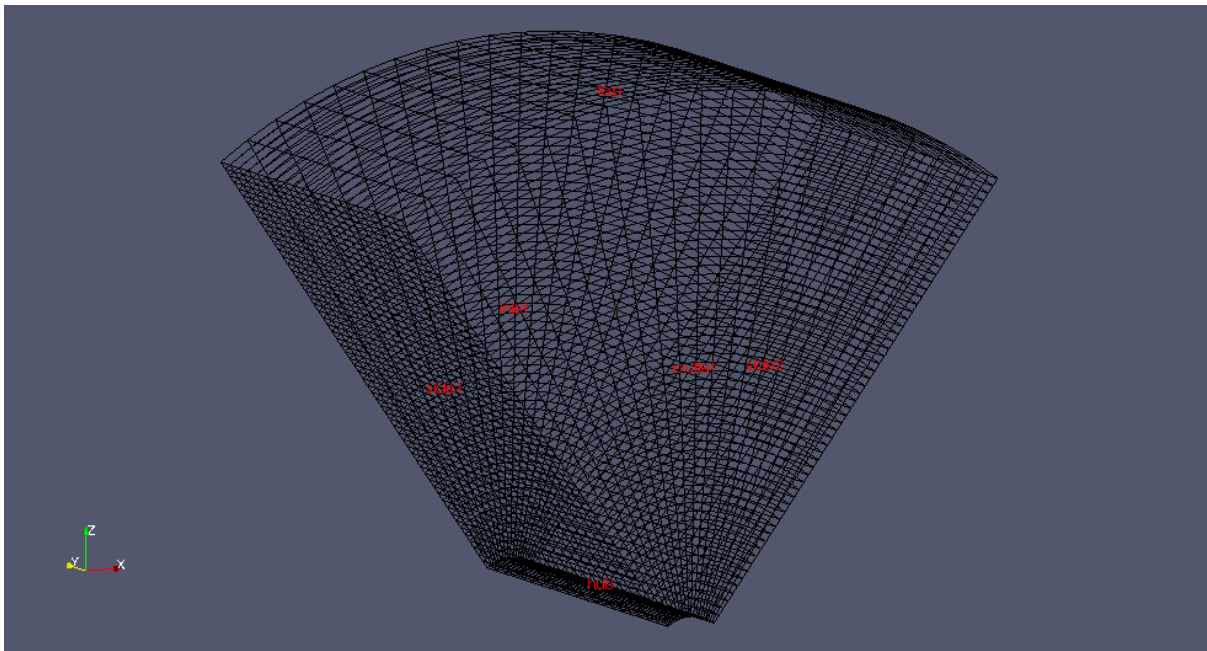


Figure 11: The block that was created initially via OpenFOAM.

Figure 11 shows the blockMesh with side1 and side2 having periodic boundary conditions.

5.2. Meshing the Single Blade with the Block

Using SnappyHexMesh, a utility provided with OpenFOAM, we can combine the blades with the block to make one mesh that can be solved iteratively. However, as seen in Figure 12, side1 and side2 now have more cells than the initial mesh as they are very close to the blade. This leads to problems when creating the periodic (cyclic) boundary conditions for these sides, as they need to be exactly the same for the createPatchDict utility in OpenFOAM to run. Therefore, periodic boundary conditions is not a viable option and all 5 blades and a new boundary mesh will need to be constructed.

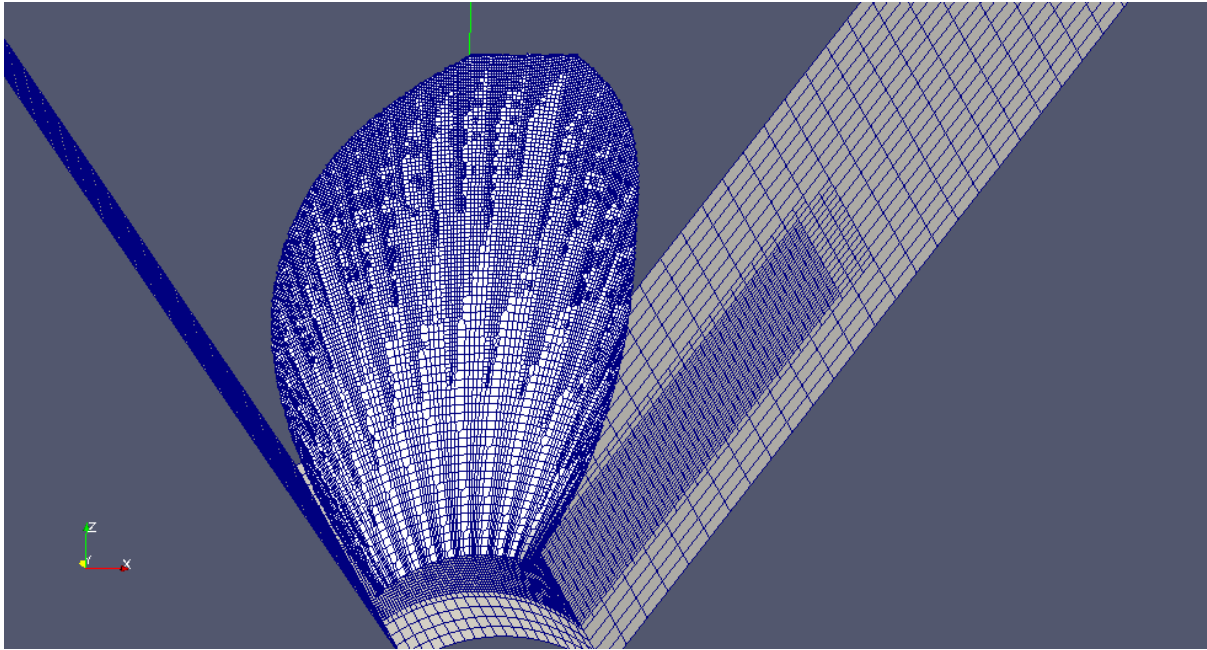


Figure 12: The blade now meshed with the block.

5.3. Meshing the Complete Propeller

In order to mesh the complete propeller, a few changes had to be made to the MatLab files. A new function (addBlades.m) takes the initial points and rotates them 72 degrees using a three dimension rotation matrix. This is done 4 times to add the remaining blades. These are then converted to STL file format individually. Figure 13 shows all 5 blades, each with 8024 surface elements.

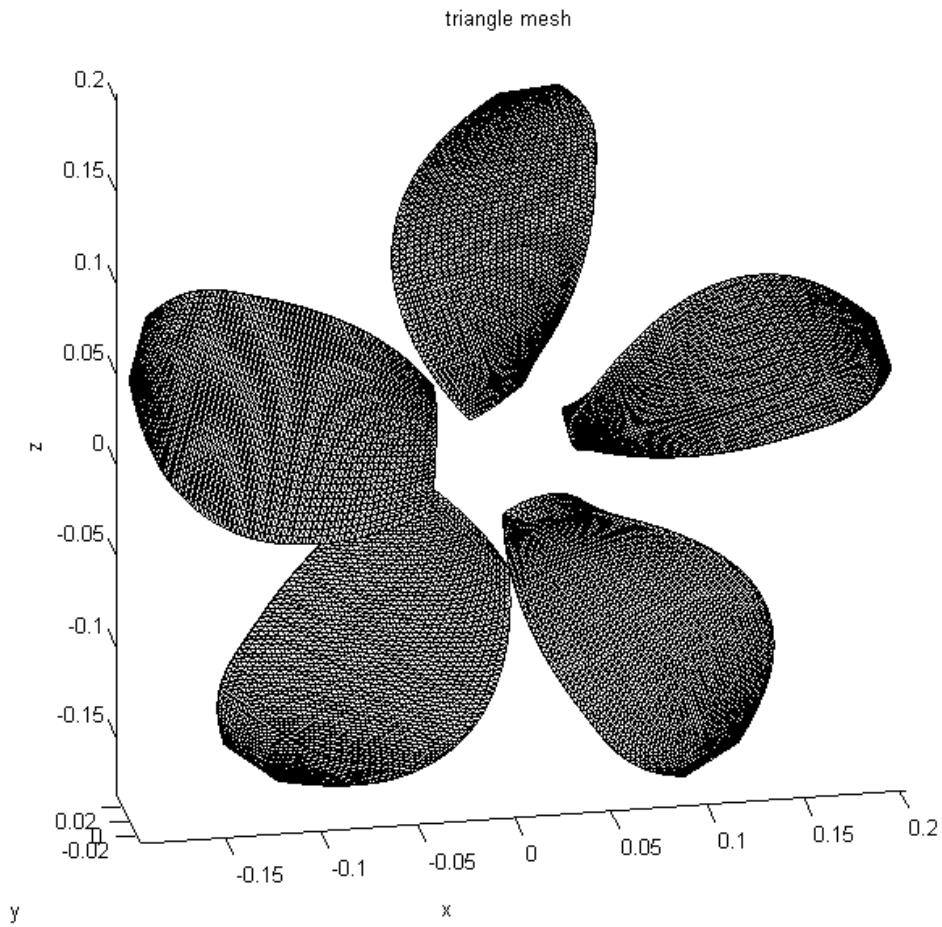


Figure 13: Showing the complete propeller with a triangle mesh.

A full cylindrical boundary mesh was created using blockMesh. It consisted of 3 separate blocks joined together around a 4 inch diameter (the hub) as seen in Figure 14. The inlet is at the positive y direction and the surrounding walls have symmetry plane boundary conditions.

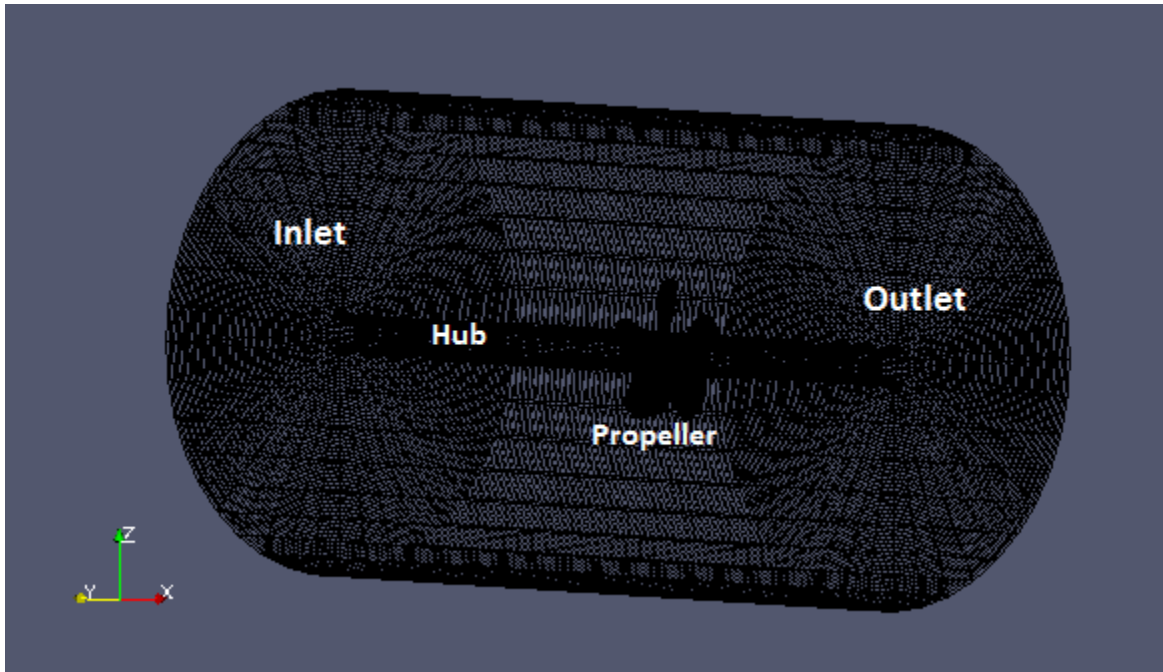


Figure 14: Representation of the blockMesh in paraView.

OpenFOAMs' utility, snappyHexMesh, was also used to attach the five blades to the boundary mesh. There are many parameters that can be varied in order to get a decent quality mesh. The amount that each cell is split is given by the level that is specified for refinement in the snappyHexMesh dictionary. For consistent meshing, a level 4 was used to refine each of the blades' surface. Level 4 was also used in the refining region at distance of 0.008 m. A higher level means a much better refinement. However, using level 5 would not snap the blades within 500 million cells. Even though the leading edge is very smooth when created via MatLab and exported using a STL file, the resulting mesh appears to have jagged edges (Figure 15).

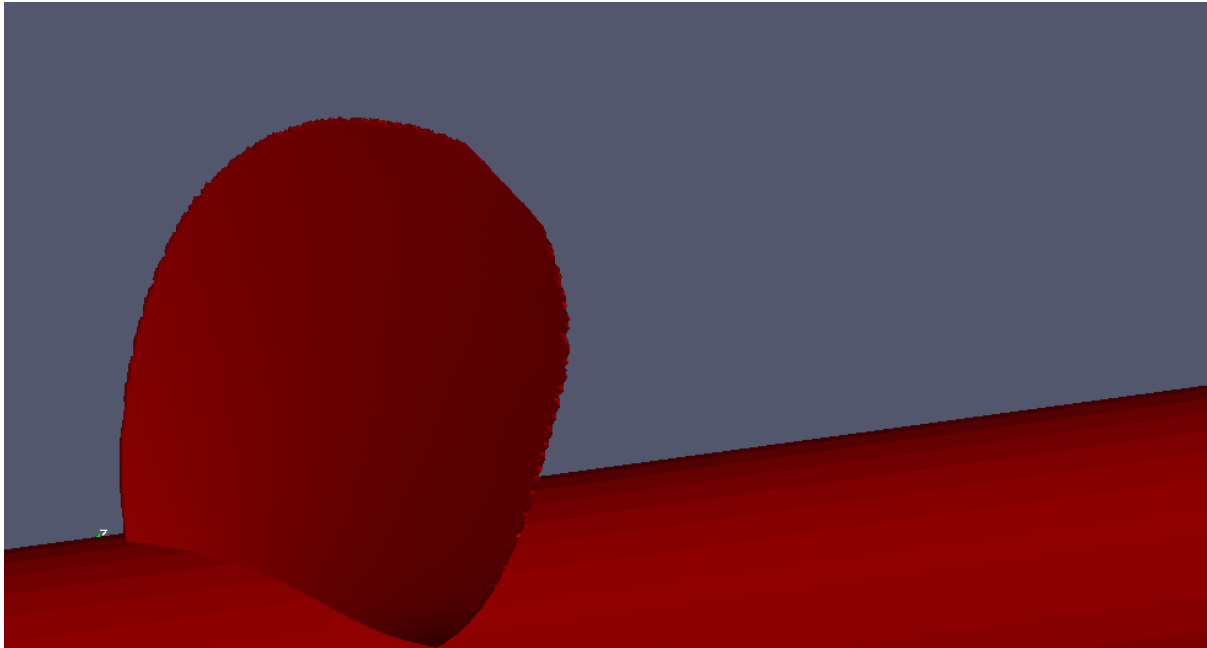


Figure 15: Jagged edges on the propeller blade(s).

The jagged edges in the mesh were attempted to be removed in multiple ways (Appendix A1). Changing the number of cells in the x, y and z direction, would change the position of where the blade would ‘snap’ to in the block. However, as the block is cylindrical, the further away from the centre, the larger the aspect ratio would be and this prohibits a smooth snap. An aspect ratio of 5.8 was found to give the minimum number of jagged edges.

5.4. SRFSimpleFoam

The SRFSimpleFoam (Single Rotating Frame) solver was trialled initially as it was thought to be less computationally intensive and easier to set up than multiple reference frames. The solver was setup so that the flow is rotating and the propeller is stationary. However, the solution would not converge, even after lowering relaxation factors and changing boundary conditions.

5.5. MRFSimpleFoam

The MRFSimpleFoam (Multiple Rotating Frames) solver was setup so that the hub and 5 blades were rotating, while the fluid is stationary (Figure 16).

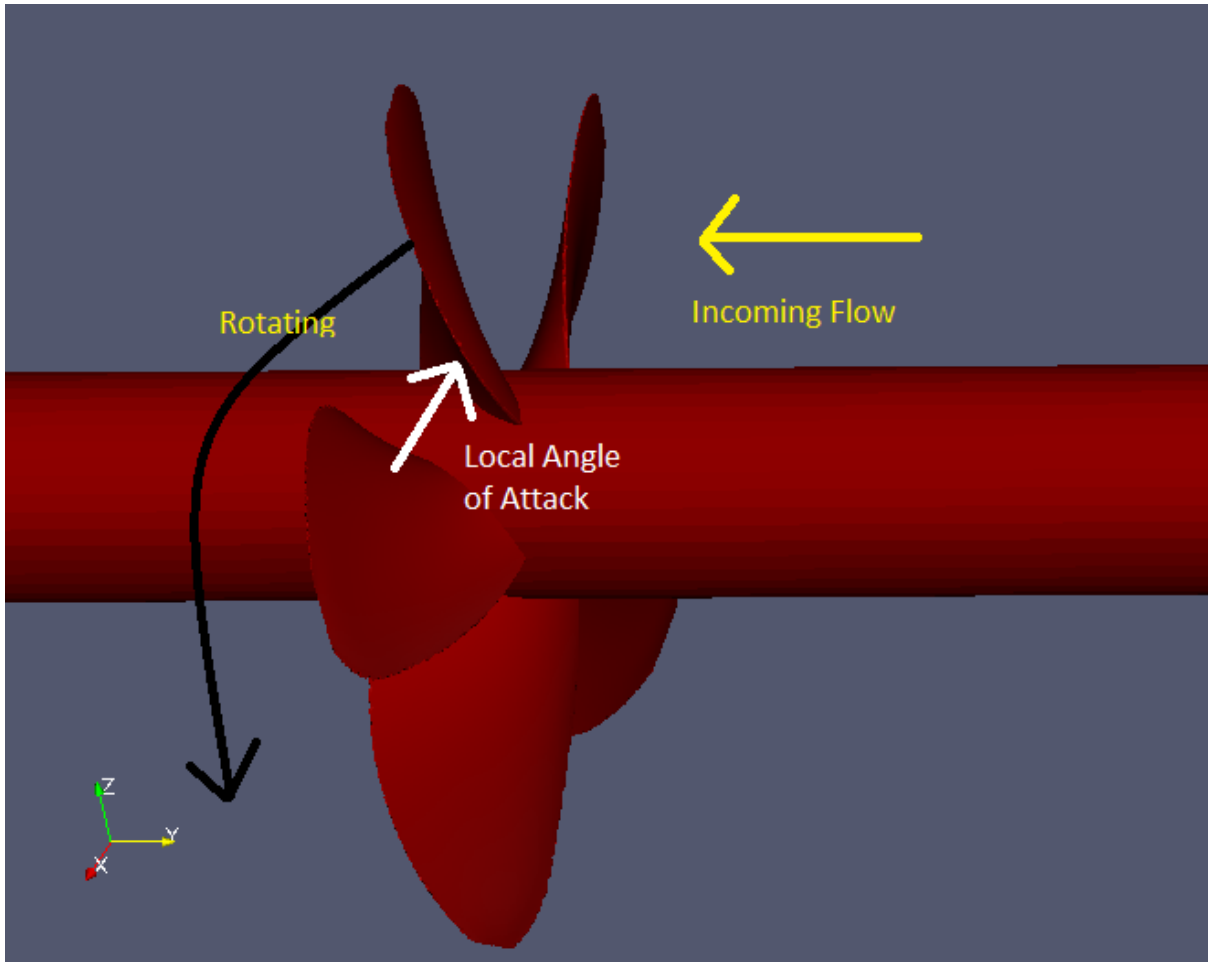


Figure 16: Setup of MRFSimpleFoam

A symmetry plane for the surrounds was used as a boundary condition and fixed velocity at the inlet and outlet. It should be noted that if allowing zero gradient boundary condition at the outlet, the flow would rotate with the blades, and produce little to no lift. There should be some swirl in the fluid, but is too complex for this particular solver. A no-slip condition is used for the hub and blades. The boundary conditions used in the simulation are shown in Table 1.

Table 1: Boundary Conditions

Patch	Type	U	ρ
Inlet	Patch	fixedValue uniform (0 -3.9624 0)	zeroGradient
Outlet	Patch	fixedValue uniform (0 -3.9624 0)	fixedValue uniform 0
Hub	Wall	fixedValue uniform (0 0 0)	zeroGradient
Surrounds	symmetryPlane	symmetryPlane	symmetryPlane
Blades	Wall	fixedValue uniform (0 0 0)	zeroGradient

The flow is modelled as laminar. Turbulence based upon a flat plate is approximated to start at a Reynolds number of 10^6 (White, 2008); the Reynolds number at the longest section of the propeller, $x = 0.12\text{m}$, is $4.74 * 10^5$ from Equation 12.

Equation 12: Reynolds number along a flat plate.

$$Re_x = \frac{U x}{\nu}$$

Using the full cylindrical block meshed with the 5 propeller blades, the face and cell zones of the mesh that would rotate need to be specified. However, OpenFOAM does not yet have a utility that converts patches to faceZones. Therefore, using the topoSetDict utility, the faces and cells within a cylinder defined by coordinates could be made into sets. These sets are then converted to cell zones and face zones.

The MRFZones utility is used to define which patches inside the zone rotate. By default all of them rotate. However, it is important to state which patches do not rotate (even outside the zone). Within this utility, ω (rad/s) is also specified. This value is varied to produce different values of J . Each case ran for 1250 to 1500 time steps with delta t equal to 1; till the force coefficients were converged. This was equivalent of approximately 3 hours for each case. It should be noted that to get the solution to converge, the simulation was run to timestep 100. Then, without changing any conditions, the simulation was run to timestep 1250 or 1500 to converge; essentially just stopping at 100 and starting again. If the solution was run to the end timestep without stopping, the solution would not converge; an explanation for this could not be reached.

The lift coefficient and moment coefficient were calculated using a function in the controlDict. This was calculated for blade1 only and assumed (and also tested) that the force is the same on each blade as the flow is uniform. Therefore, the coefficients were multiplied by five and calculated to get the thrust and torque coefficients.

6.0 Analysis

6.1. Input Values

The input values from the experimental data (Emerson & Sinclair, 1967) have been converted to SI units. The values they have used have not been specific, as they tested a number of different shaped propellers and stated only cavitation number but not speed or pressure. The cavitation tank was changed at certain stages to allow for different tank widths and also upstream pressures and velocities. However, they have given a range of 10 to 16 feet per second.

For this project, a fixed linear (upstream) velocity was chosen (13 feet per second) and the revolutions were varied in accordance with J . The solver required an input of the angular velocity the propeller would be rotating at. These values are shown in Table 2.

Table 2: Values used in the computational model.

Diameter (m)	Linear Velocity (m/s)	J	Revolutions (1/s)	Angular Velocity (Rad/s)
0.4064	3.9624	0.40	24.38	153.15
0.4064	3.9624	0.45	21.67	136.14
0.4064	3.9624	0.50	19.50	122.52
0.4064	3.9624	0.55	17.73	111.38
0.4064	3.9624	0.60	16.25	102.10
0.4064	3.9624	0.65	15.00	94.25
0.4064	3.9624	0.70	13.93	87.52
0.4064	3.9624	0.75	13.00	81.68
0.4064	3.9624	0.80	12.19	76.58

6.2. Computational Results – Thrust and Torque

The lift coefficient and moment coefficient are obtained from an OpenFOAM forces function. ForceCoeffs.C calculates the coefficients based upon the input in the controlDict from the solver and computes the forces on the specified patch. These can then be used to find the thrust and torque coefficients (Equation 4 and Equation 5). The computational results obtained from OpenFOAM are shown in Table 3.

Table 3: Results from the computational model.

Thrust Coefficient	Torque Coefficient	Cavitation Number	Efficiency
0.234	0.032	2.02	0.46
0.218	0.031	2.55	0.50
0.201	0.030	3.15	0.54
0.182	0.028	3.81	0.56
0.160	0.027	4.54	0.57
0.136	0.025	5.33	0.57
0.108	0.022	6.18	0.55
0.078	0.019	7.09	0.49
0.043	0.015	8.07	0.35

The experimental data (Emerson & Sinclair, 1967) is represented graphically (Figure 17). The values were extracted and then compared to the computational values. The experimental values used are the higher cavitation number of the KCD 32 blade as the solver cannot account for loss of torque and thrust due to cavitation.

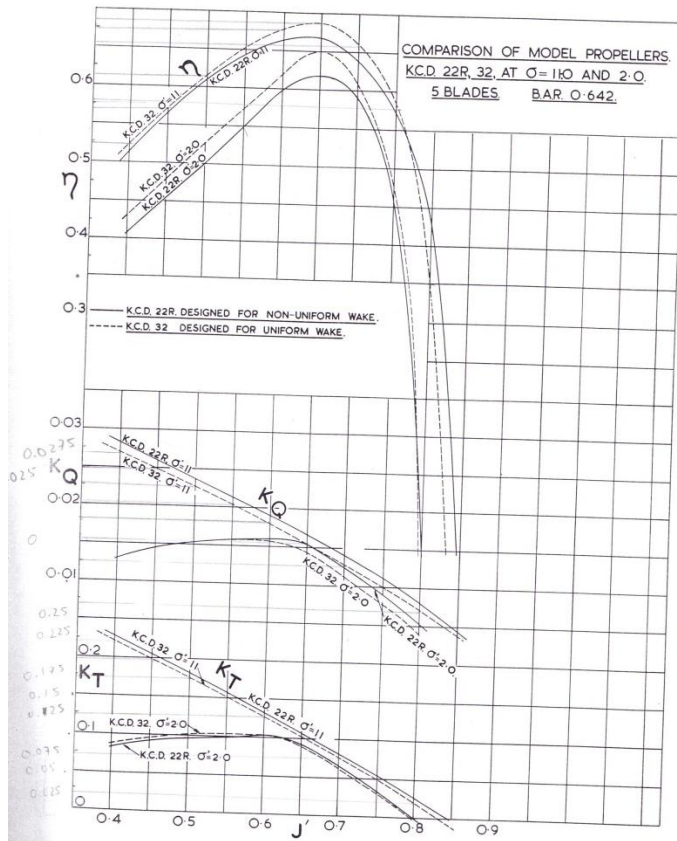


Fig. 16

Propeller Cavitation: Systematic Series Tests on 5- and 6-Bladed Model Propellers

241

Figure 17: Graphical representation of the experimental results (Emerson & Sinclair, 1967).

The comparison between experimental and computation thrust coefficients can be seen graphically in Figure 18. The experimental values seem to have more of a linear trend than what is displayed of the computational trend. However, still to be noted that a slight increase about the middle J values. The computational results are a maximum of 26% difference at J value 0.65.

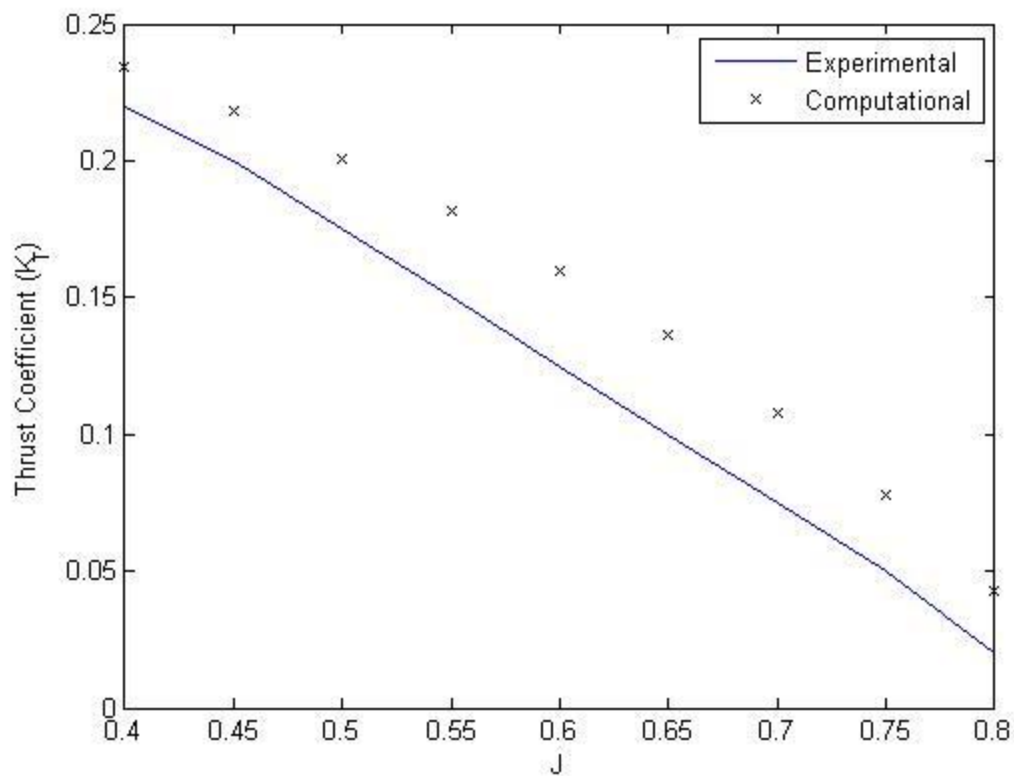


Figure 18: The relationship of the thrust coefficient and J.

The torque coefficient comparisons are shown in Figure 19. The computational results are larger than the experimental with a maximum difference of 37% at 0.65. However, the computational results don't seem to show any outliers, and depicts a rather smooth curve.

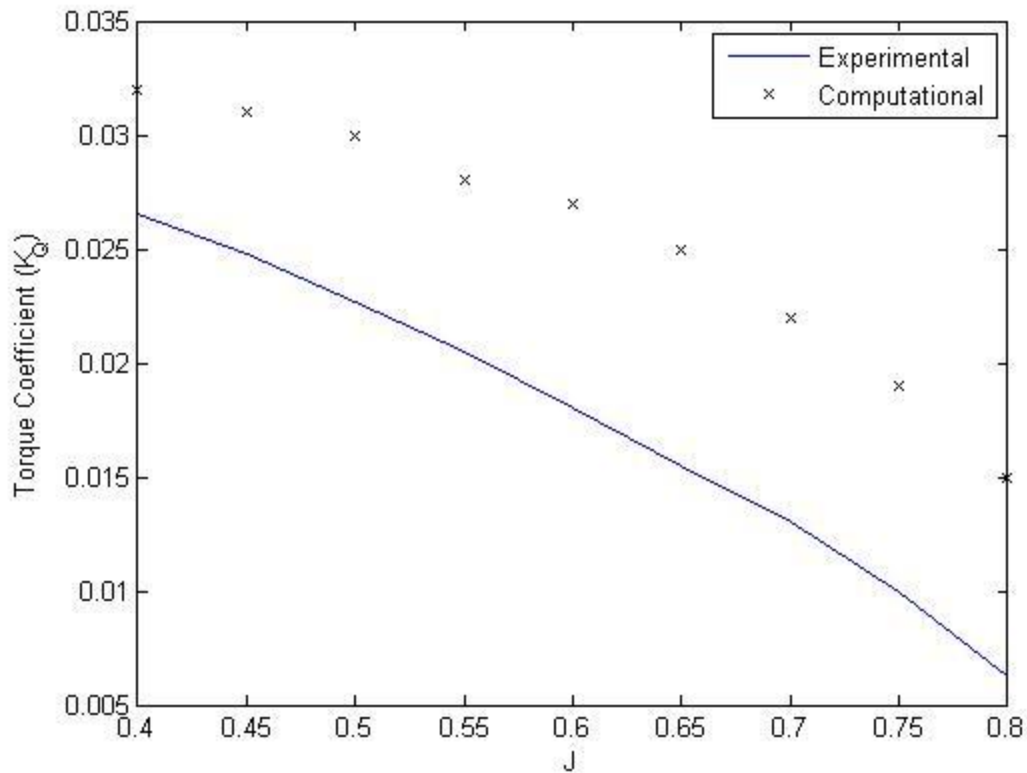


Figure 19: The relationship of the torque coefficient and J.

The efficiency comparisons (Figure 20) show a fairly large discrepancy between the two data sets. It is observable that the efficiency is at a maximum when J is between 0.60 and 0.65. However, the computational results don't reflect this peak at those J values, and is actually rather flat. Yet, the other features of the efficiency curve are maintained; a large drop in efficiency at higher values of J and a linear decrease in efficiency at lower values of J.

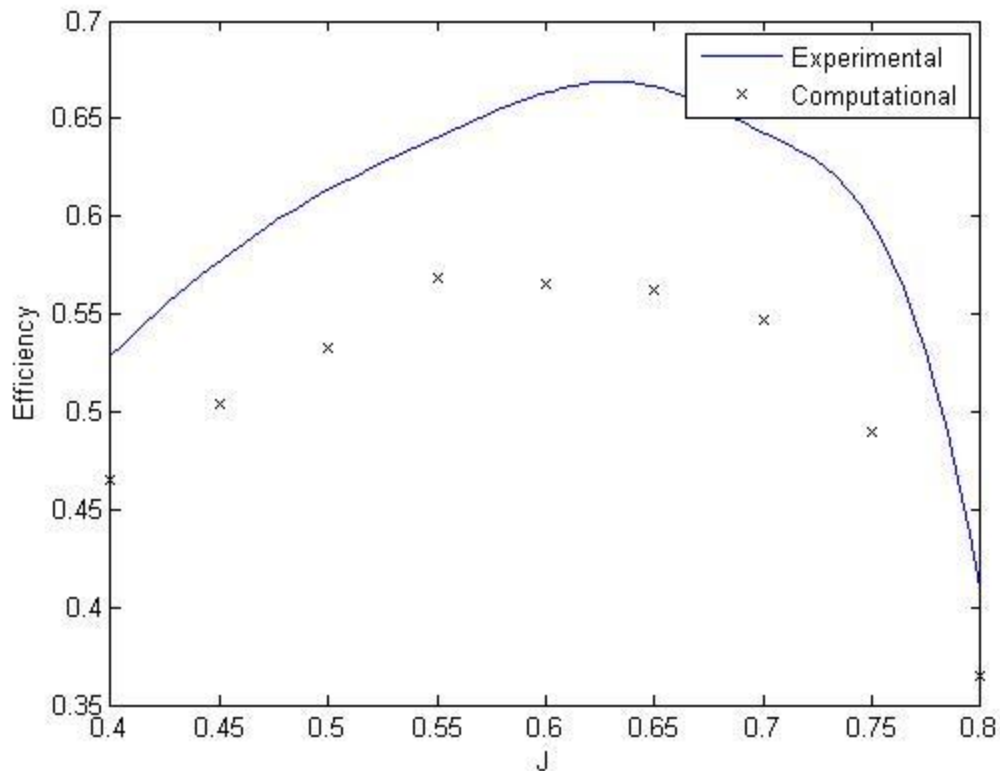


Figure 20: The relationship of the efficiency and J.

The computational results seem to over predict the thrust and torque of the propeller. This seems to be a prevalent observation in RANS CFD simulations (Rhee, 2005; Watanabe et al, 2003). Rhee suggests this is due to viscous flow scale effect and also under-predicted tip vortex strength. In the paper, they have used a fully turbulent simulation even though “50% to 60% of the flow region is laminar”, due to having no recognised method to handle the transition. However, turbulence would only be (dominantly) impacting the torque as this is more sensitive to the viscous flow. In this project, it is evident there is a larger discrepancy in the torque than the thrust and this could possibly be due to not modelling any turbulence.

Moreover, as the solver cannot predict the effect on lift and torque due to cavitation, the computational data is expected to be higher than the experimental results that do allow for this.

6.3. Computation Results – Pressure Distribution

The pressure at the back and face of the blade were monitored at each J value via paraView. It allowed prediction of cavitation and also the features of the pressure along the blade. It should be noted that the legend values shown in the figures are given as P/ρ . When comparing to model tests, the density is 1000 kg/m^3 and therefore, the legend on each figure can essentially be read as kPa.

The vapour pressure in 20 °C water is 2.3 kPa (White, 2008). Cavitation occurs when the localised pressure falls below the vapour pressure. In the simulation, 0 is set to atmospheric pressure of 101.3 kPa and therefore, cavitation will occur at 99 kPa below atmospheric pressure.

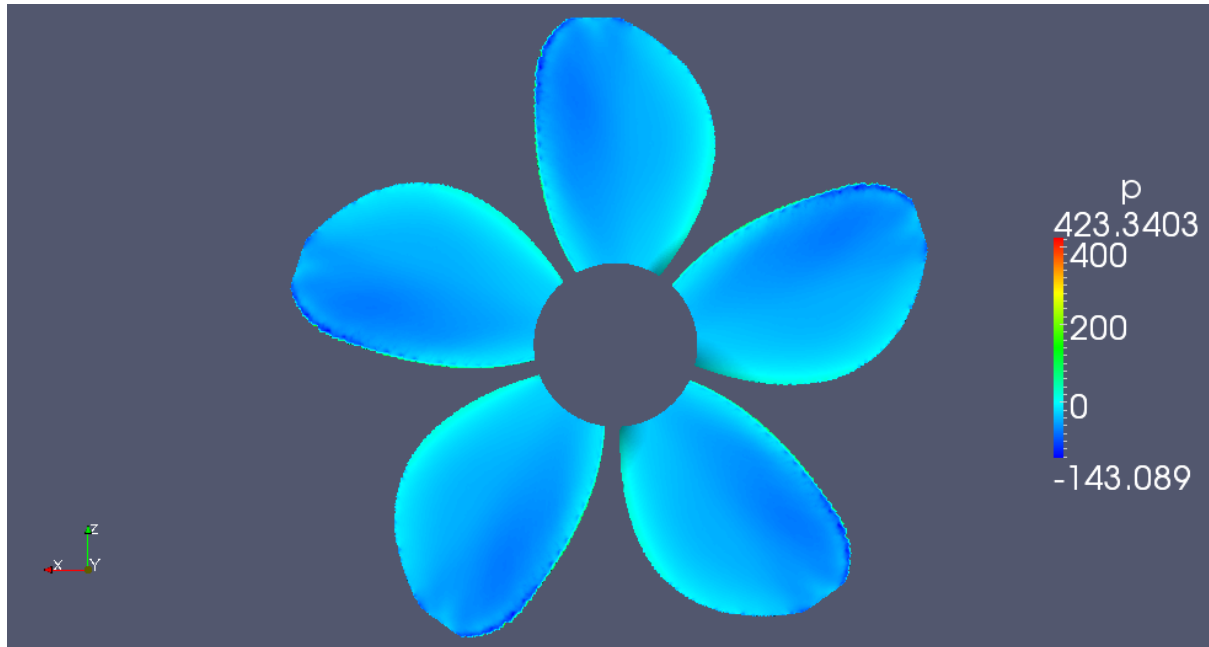


Figure 21: Back side of the propeller, J value of 0.40

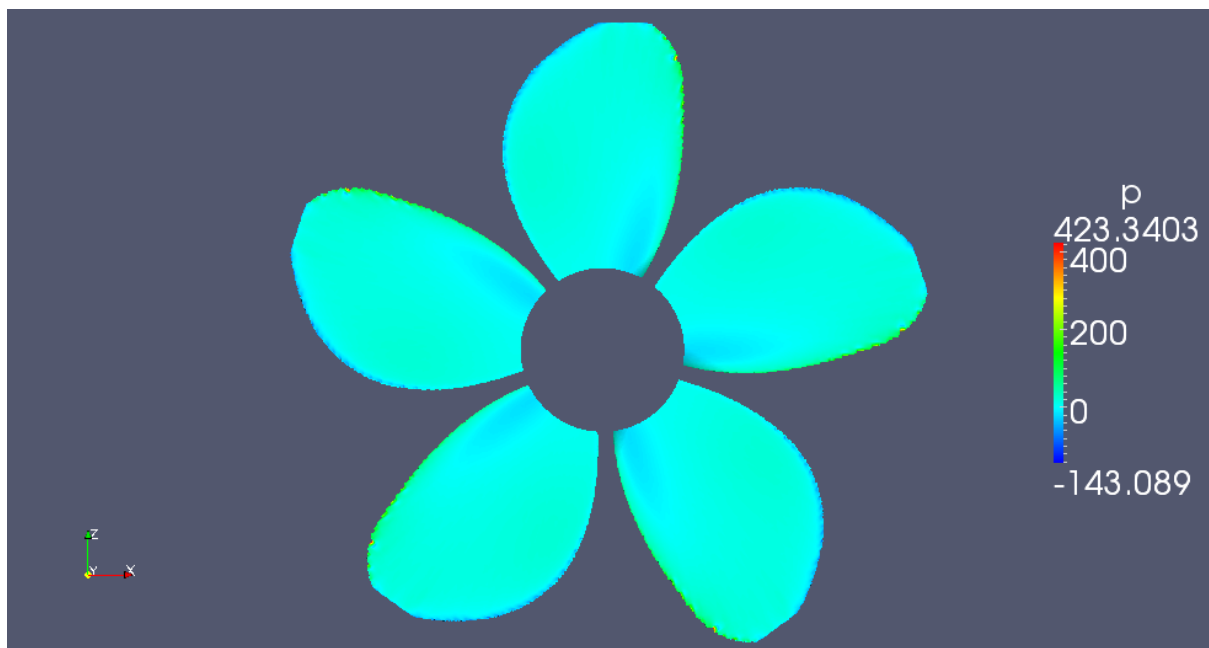


Figure 22: Face side of the propeller, J value of 0.40

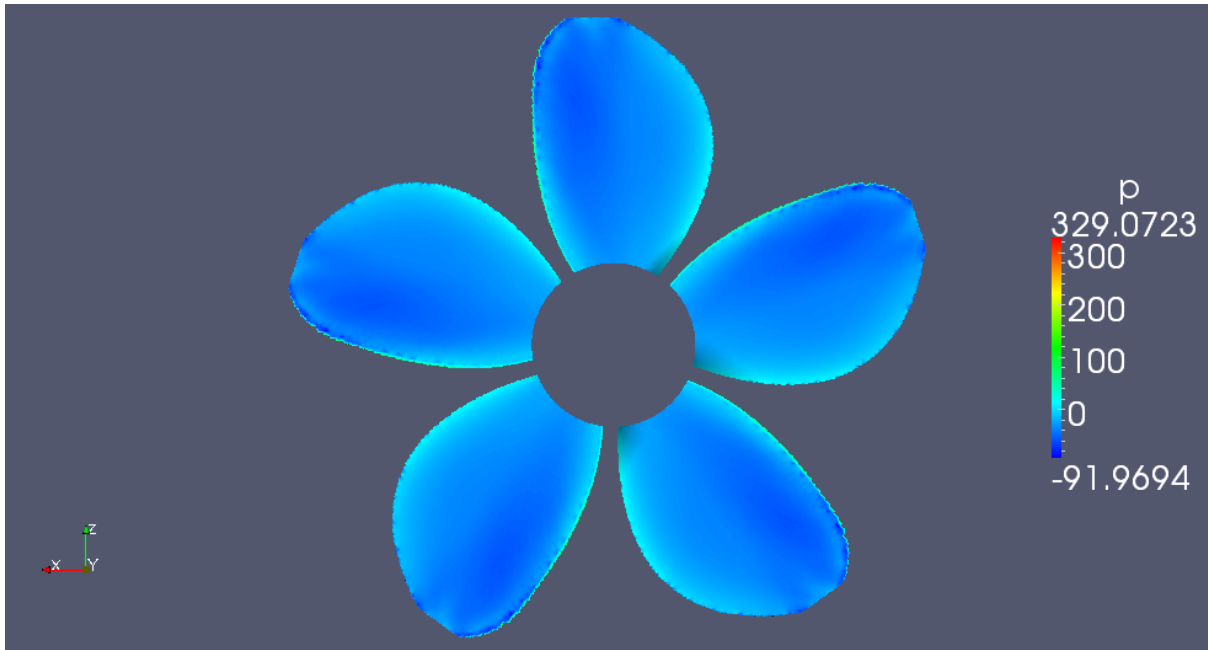


Figure 23: Back side of the propeller, J value of 0.45

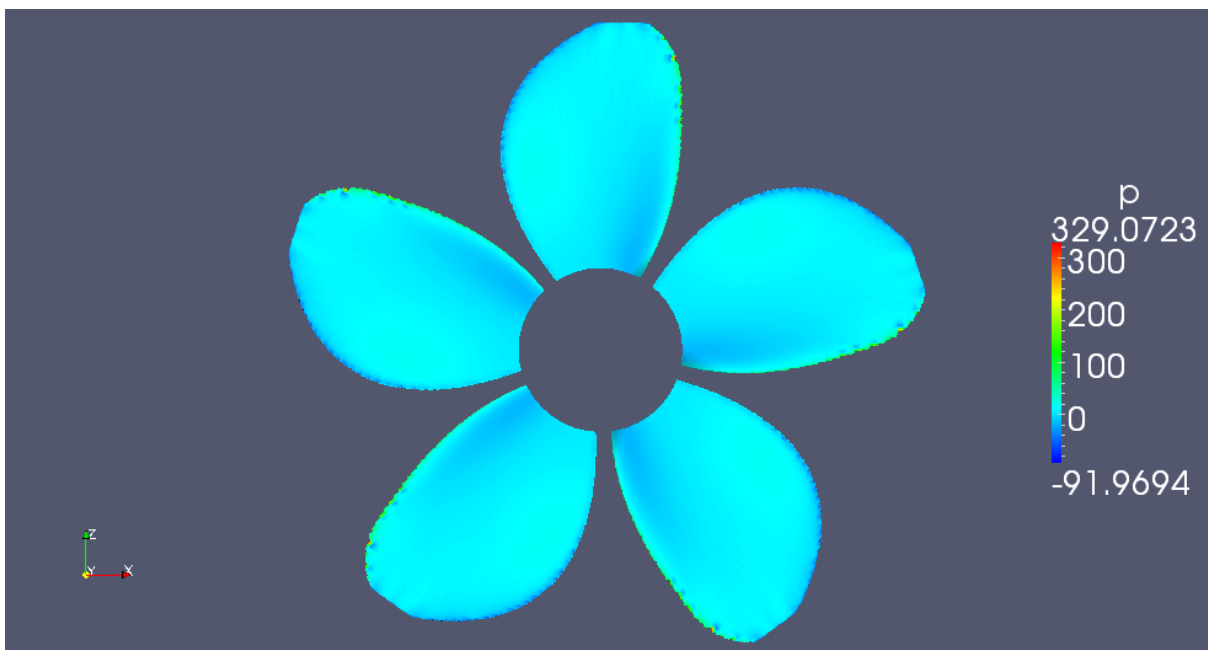


Figure 24: Face side of the propeller, J value of 0.45

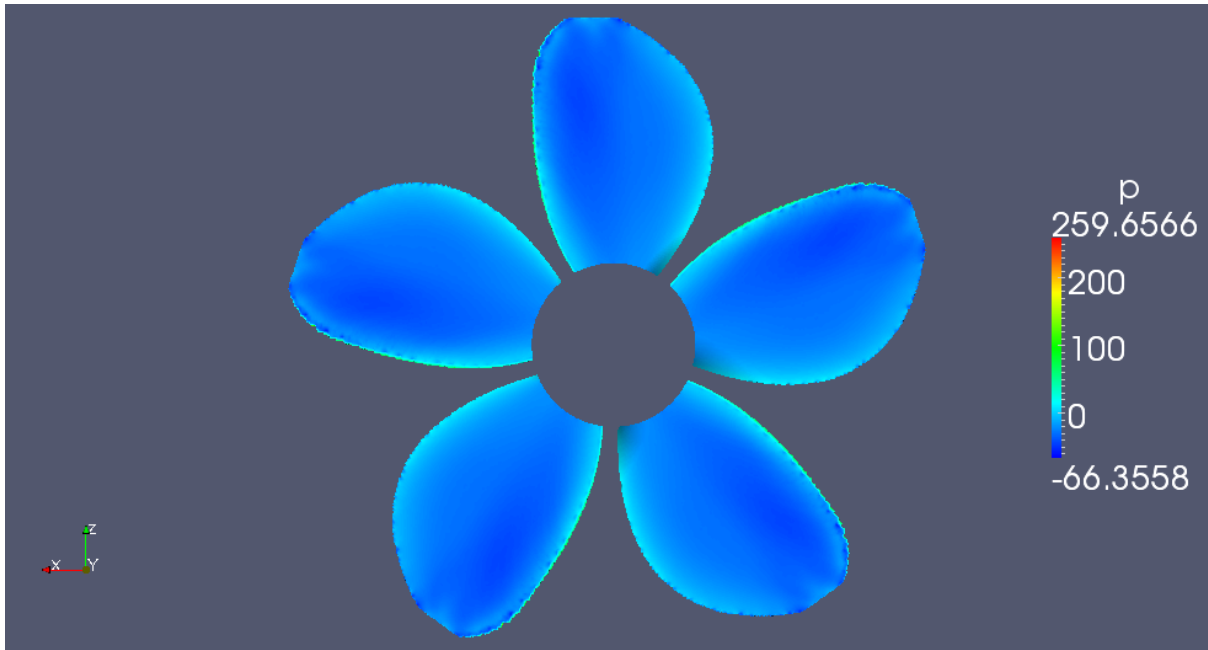


Figure 25: Back side of the propeller, J value of 0.50

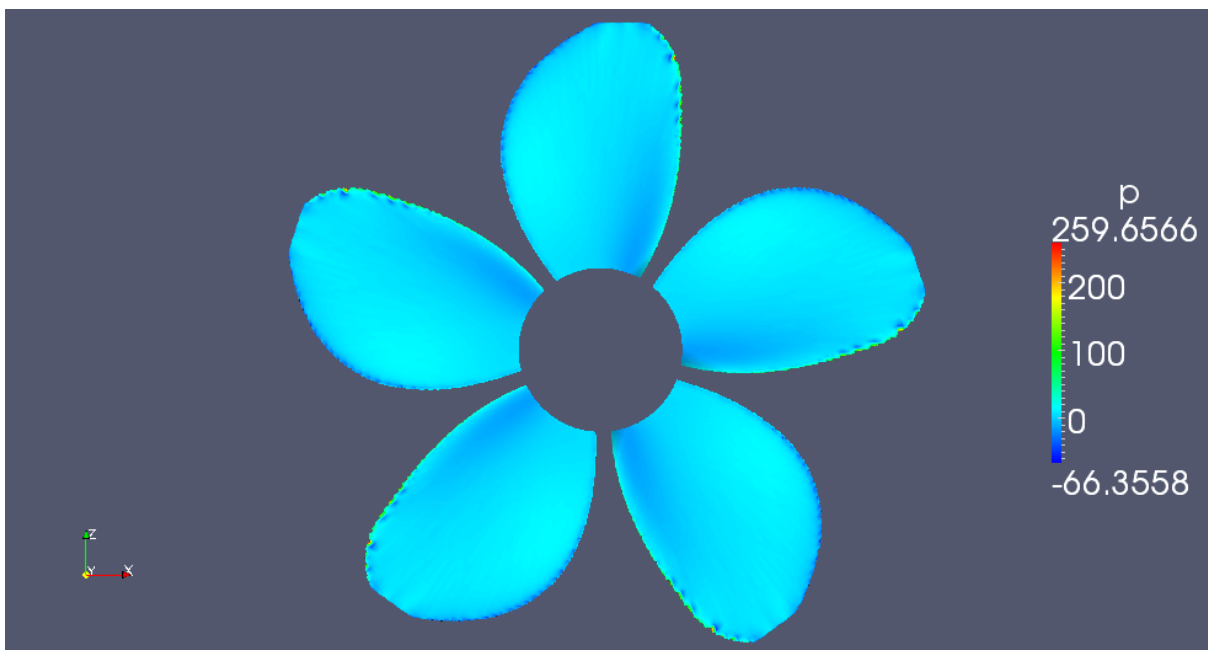


Figure 26: Face side of the propeller, J value of 0.50

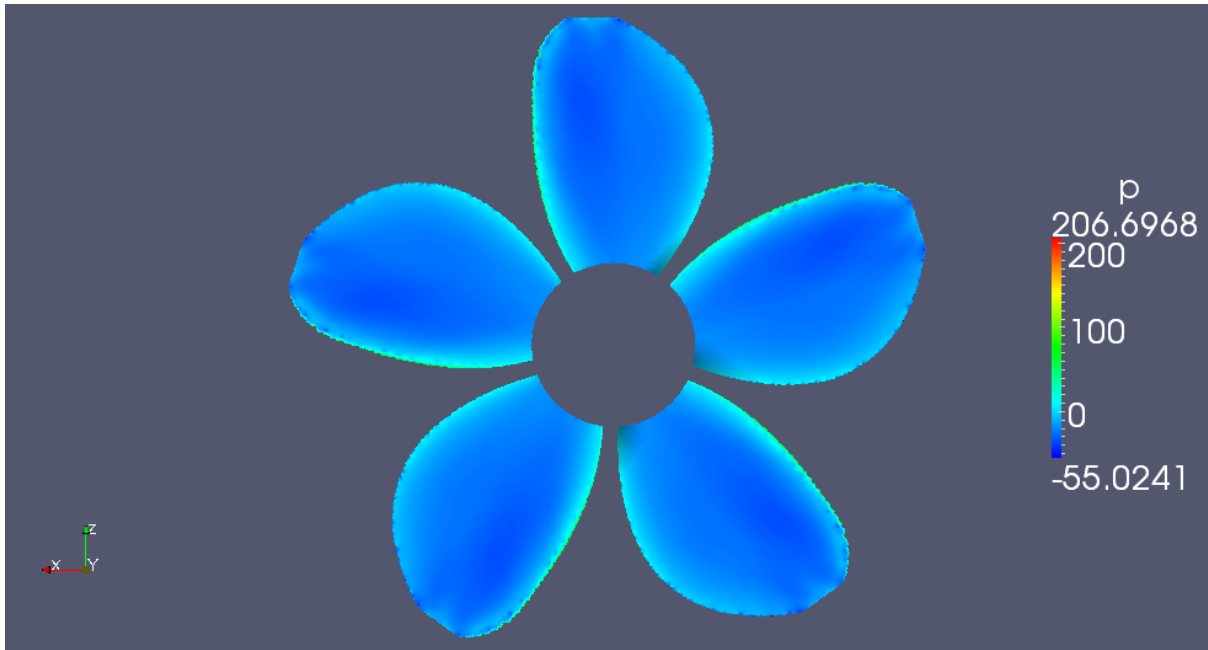


Figure 27: Back side of the propeller, J value of 0.55

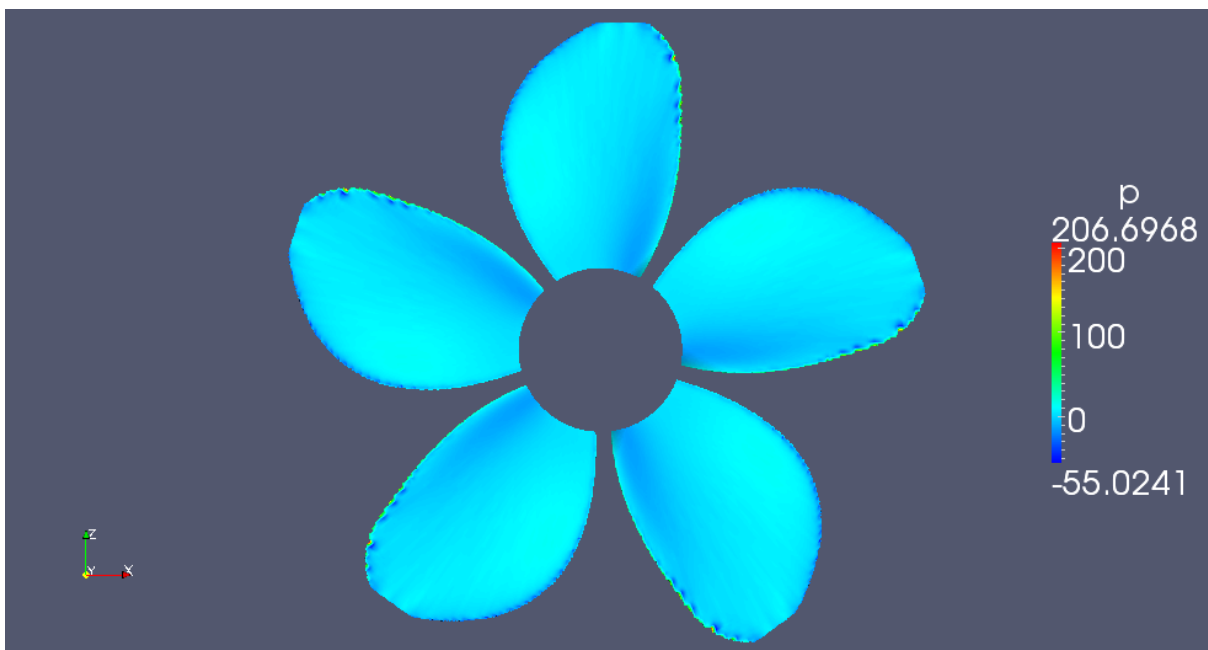


Figure 28: Face side of the propeller, J value of 0.55

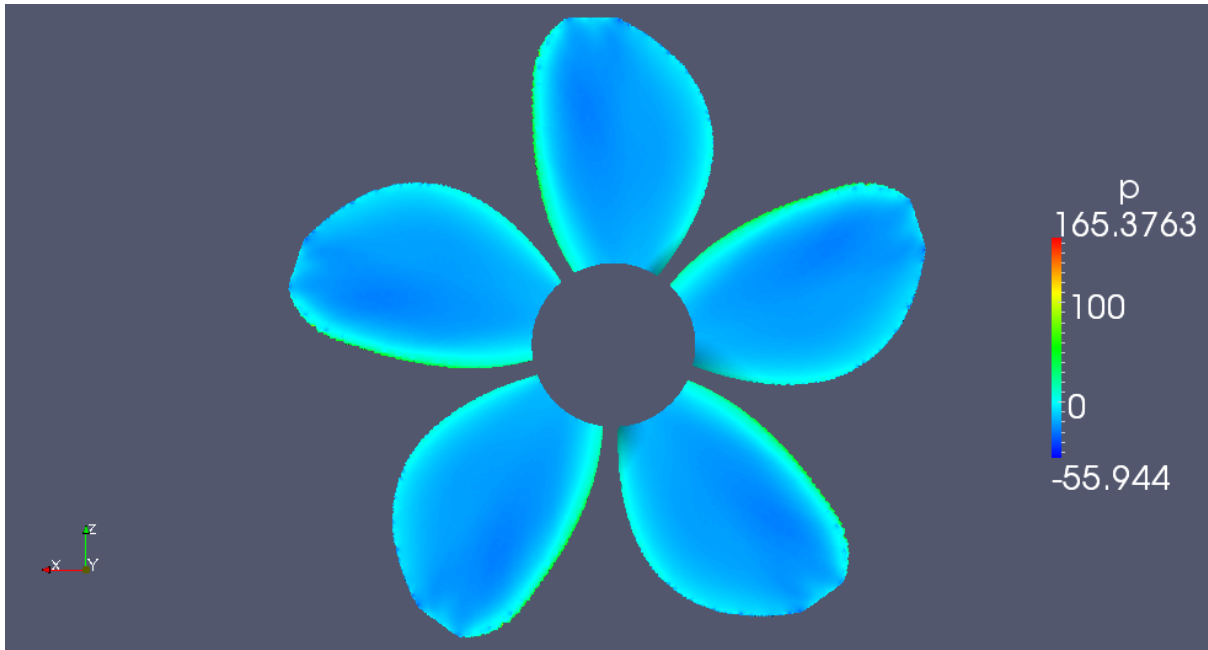


Figure 29: Back side of the propeller, J value of 0.60

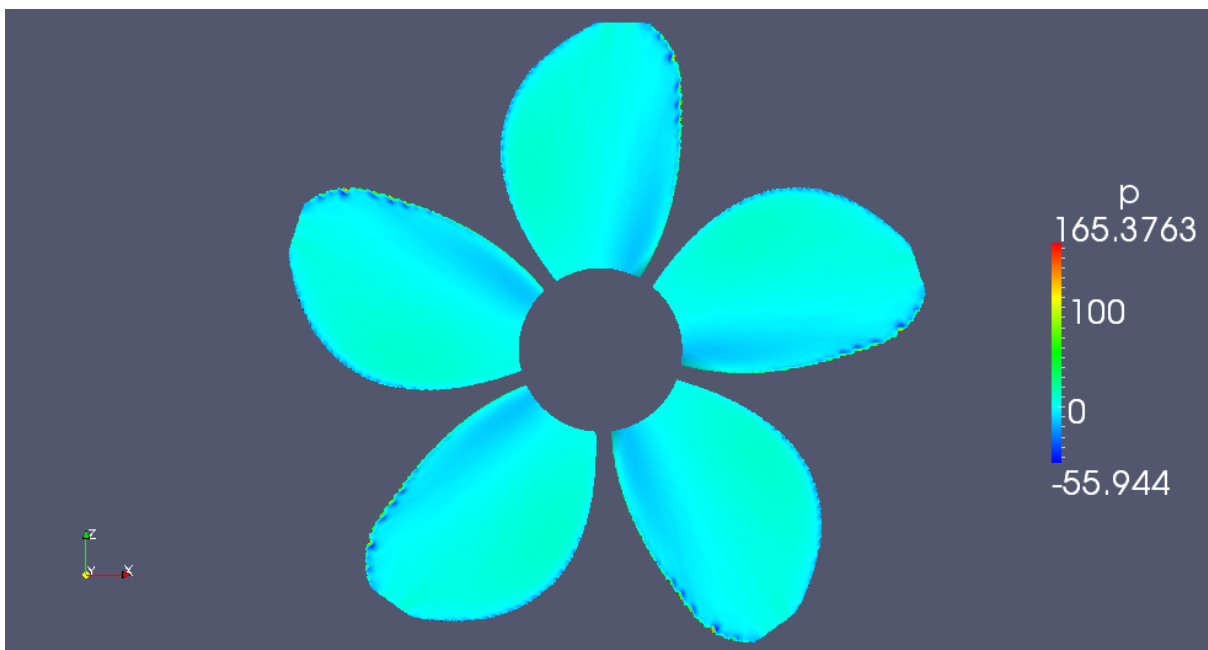


Figure 30: Face side of the propeller, J value of 0.60

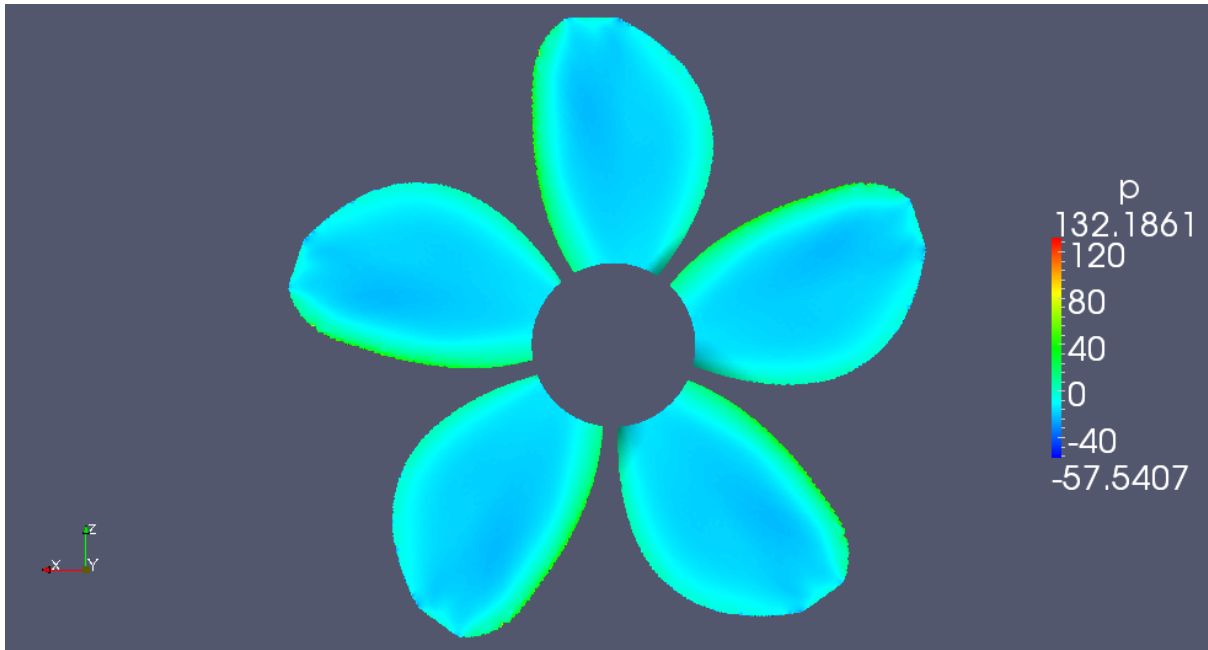


Figure 31: Back side of the propeller, J value of 0.65

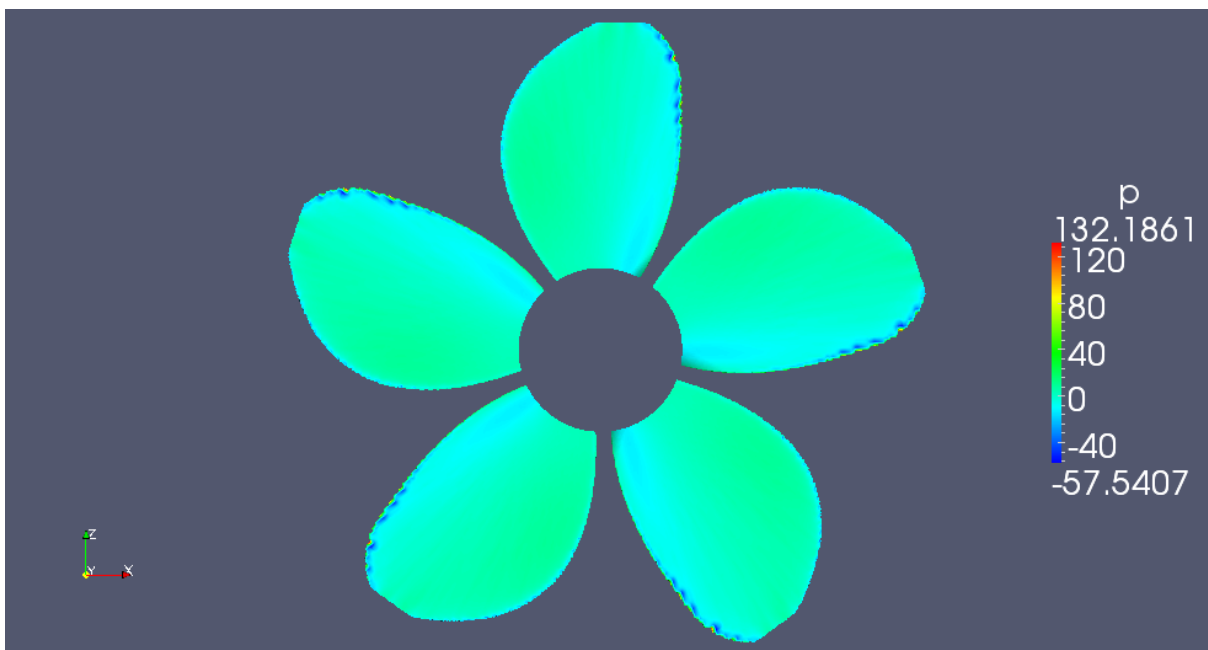


Figure 32: Face side of the propeller, J value of 0.65

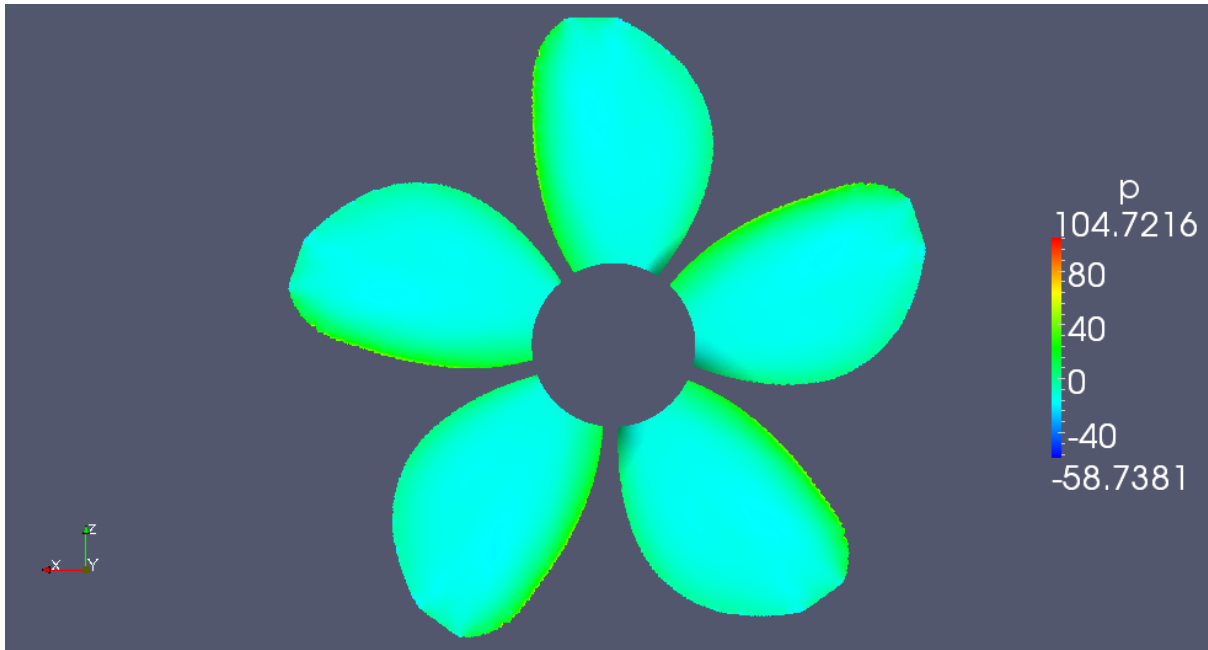


Figure 33: Back side of the propeller, J value of 0.70

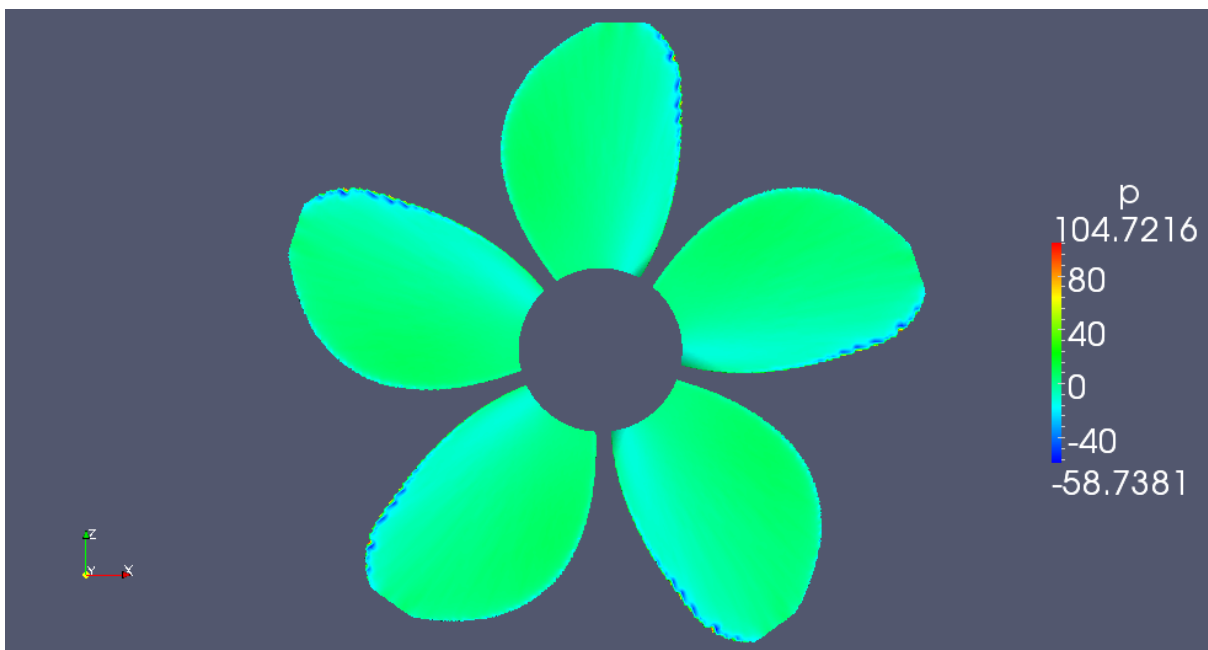


Figure 34: Face side of the propeller, J value of 0.70

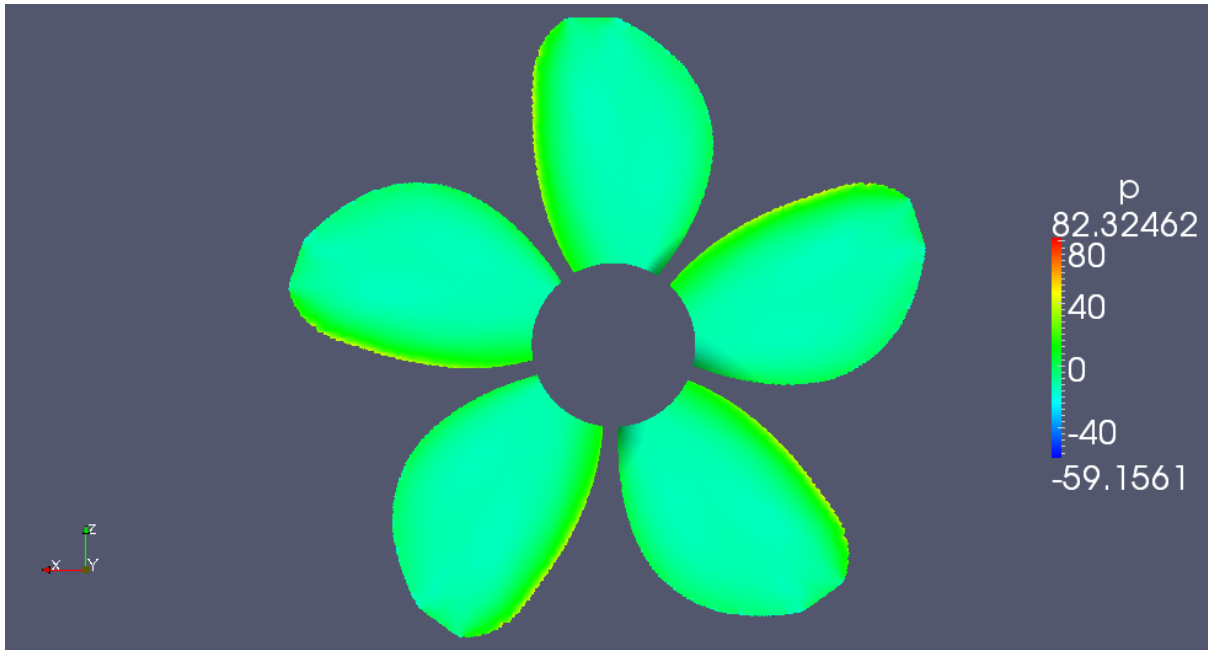


Figure 35: Back side of the propeller, J value of 0.75

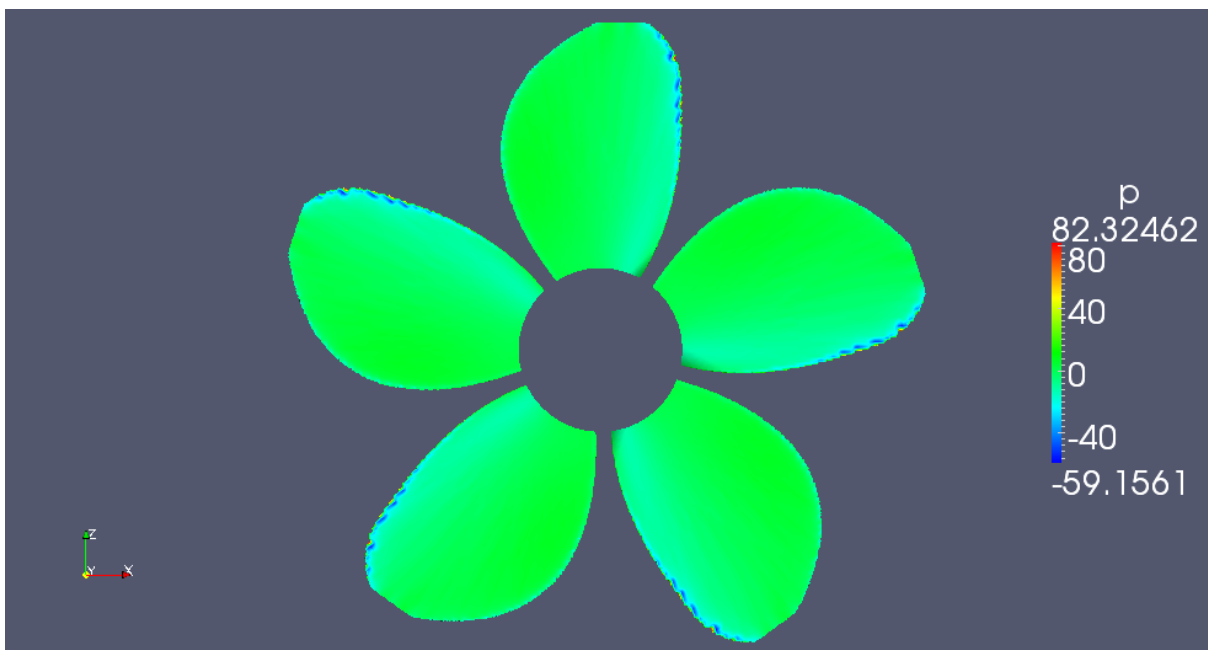


Figure 36: Face side of the propeller, J value of 0.75

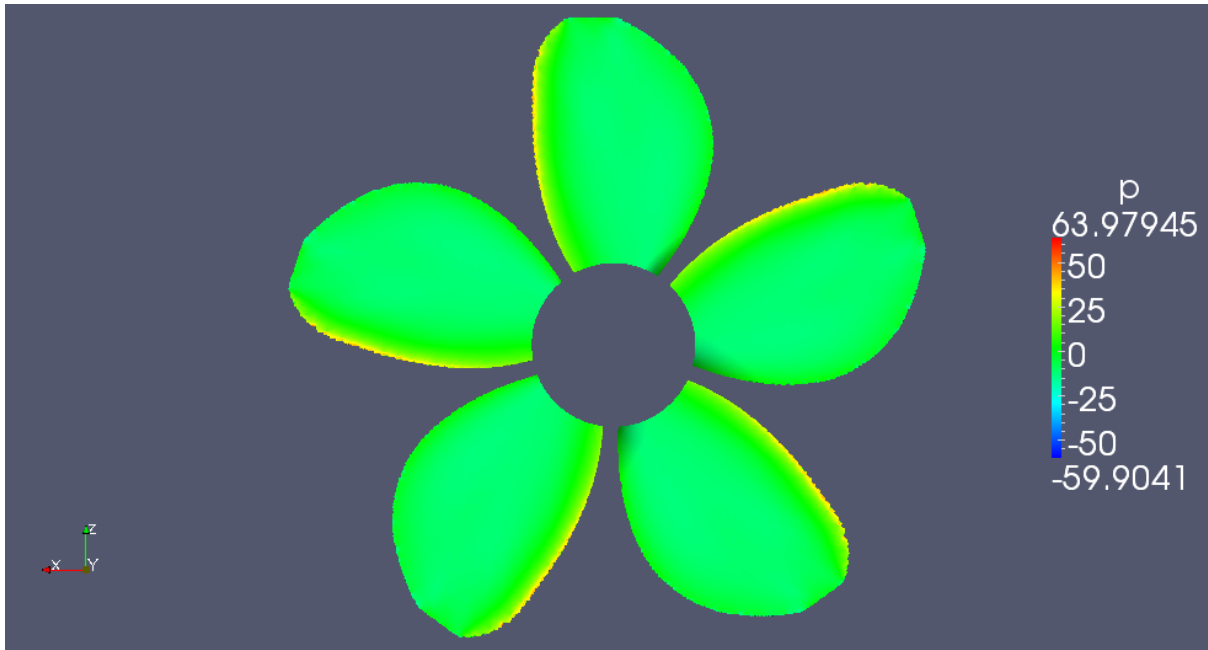


Figure 37: Back side of the propeller, J value of 0.80

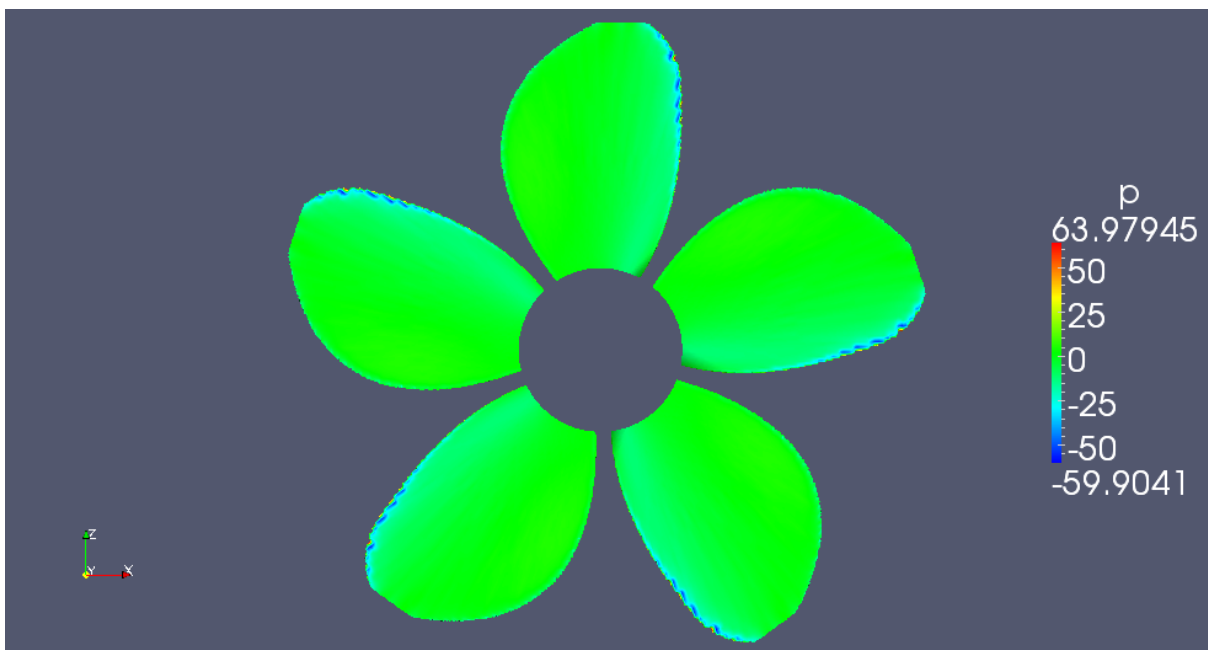


Figure 38: Face side of the propeller, J value of 0.80

Figure 21 to Figure 38 show the pressure distribution of the face of the blades and back of the blades from low J values to high J values respectively. From inspection, they tend to agree with the typical pressure distribution. That is, higher pressure at the centre of the face and lower pressure on the back with a predominant low pressure region near the tip (Rhee, 2005).

Figure 21 shows the back side of the propeller with a lowest pressure of -143 kPa (143 kPa below atmospheric pressure). This is a low enough pressure for those very low pressure regions to be cavitating. On closer inspection of Figure 21 (Figure 39), it shows that these low regions occur near the leading edge and tip, indicating the origin of the tip vortex and quite possibly blade cavitation. Higher pressure regions are towards the trailing edge and this is where the cavitation damage would most probably occur; the vapour bubbles implode as they move into a higher pressure region (White, 2008). It is also interesting to note that there are also low pressure regions on the blade face near the tip and trailing edge, which would also contribute to the tip cavitation (tip vortex).

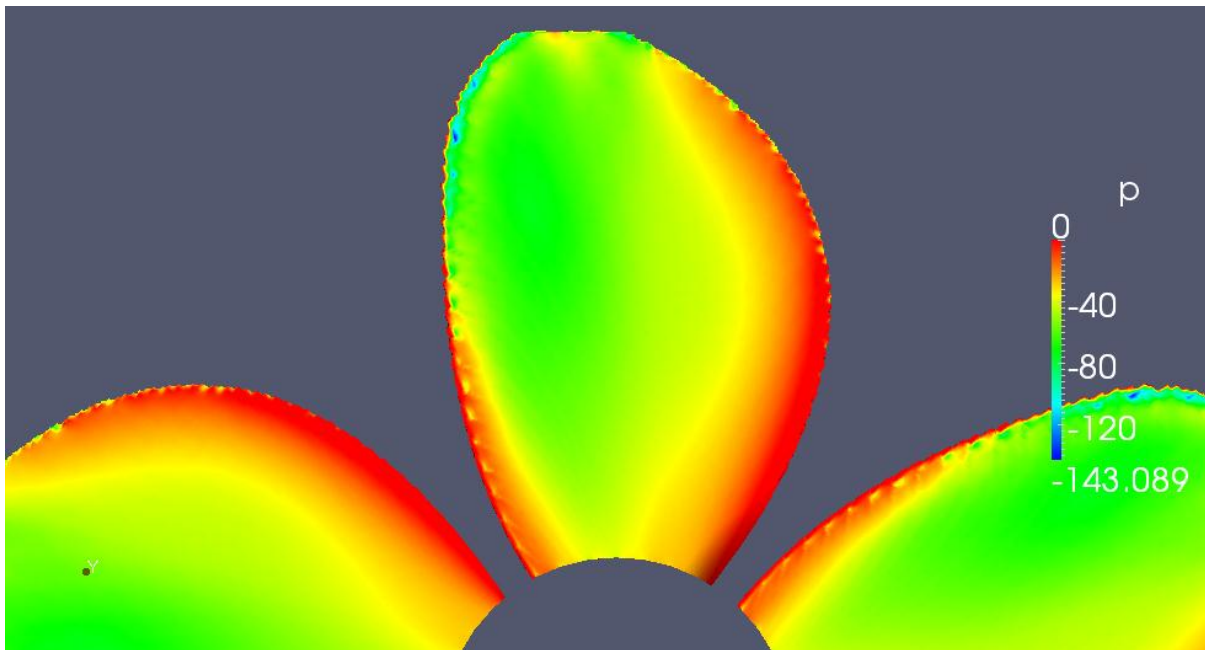


Figure 39: Closer inspection of the back side of the blade, $J = 0.40$

The other simulations for J values of 0.45 to 0.80 and a cavitation number range of 2.55 to 8.07 (Table 3) show no regions that have low enough pressure to cavitate. It is interesting to note the differences in pressure distribution between low and high J values on the back of the blades (Figure 39 and Figure 40 respectively). The dominant high pressure region is now on the leading edge (high J value) rather than the trailing edge (low J value).

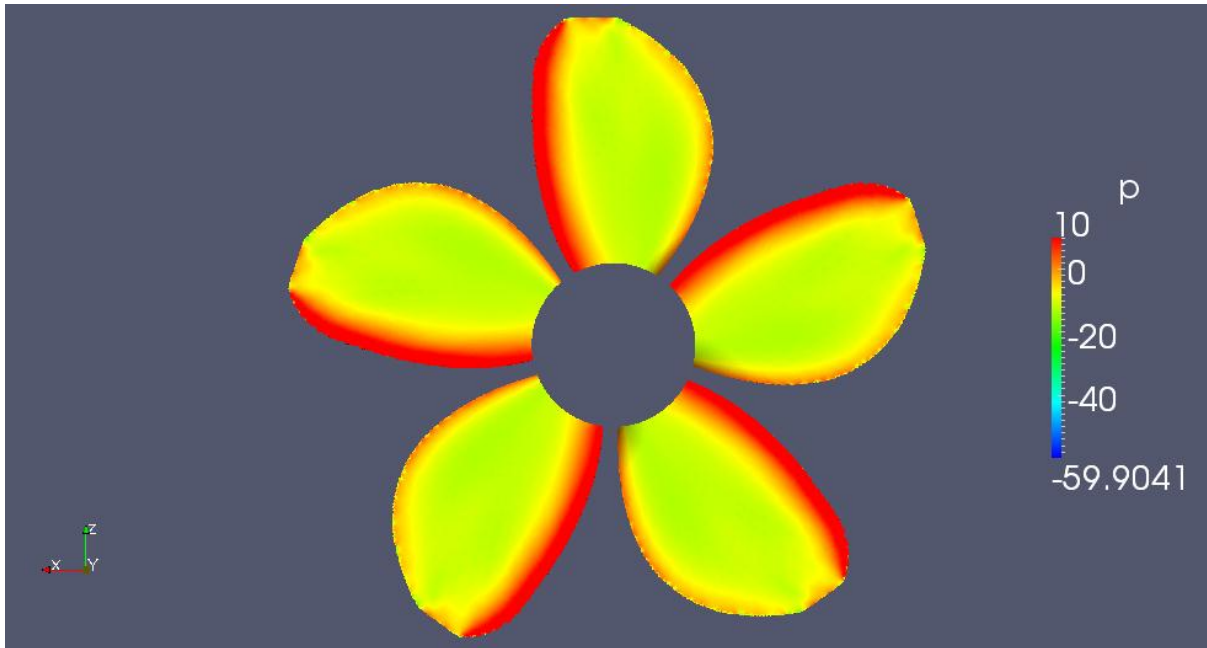


Figure 40: Closer Inspection of the back side of the blade, $J = 0.80$

Due to how the solver is computed, streamlines about the propeller cannot be seen clearly. This makes it difficult to view the flow patterns about the leading edge: the effect the local angle of attack has on the propeller blade(s) and whether flow separation will occur.

Ideally would want to see how much influence the mesh generation has on the computational results. In particular, the jagged edges on the leading and trailing edge that could not be removed (at time of writing). Mesh resolution and generation has been a major topic in CFD publications (Rhee 2005; Bhaskaran 2012; Biswas, 1998) and contributes to a large portion of the accuracy of the solution. Therefore, if the jagged edges are non-aesthetic, then they would definitely have a negative effect on the flow (Bensow, 2010).

However, the results seem to be within reasonable range of the experimental tests conducted in a cavitation tunnel (Emerson & Sinclair, 1967). The minor discrepancy in the results may be partly due to lack of precise experimental data to replicate, given the wide range of variables used or mesh resolution. A small difference in the linear velocity, and hence cavitation number produced quite large differences in the thrust, torque and also pressure distribution in OpenFOAM. Figure 20 shows a flatter efficiency curve in the computational results than the experimental, but has a peak in approximately the same place.

It was observed that the experimental tests were conducted in non-homogenous flow, "...a considerable amount of entrained air in the water" and was discussed in the paper to a fair extent. Hans Edstrand, a member of the 'The Society of Naval Architects and Marine Engineers' (Emerson & Sinclair, 1967) made a remark that the trailing edge cavitation erosion never occurred in homogenous flow tests. However, the clarity of detecting cavitation was noted to be more difficult in non-homogenous flow.

7.0 Conclusion

This dissertation has discussed the method of constructing a propeller from coordinates and simulating the flow in OpenFOAM. From the analysis, the simulation could predict where and when cavitation would occur via pressure distribution figures, yet lacked in calculating the affected thrust and torque coefficients due to it. The computational results were compared to experimental results (Emerson & Sinclair, 1967) and although had small discrepancies, they were reproduced reasonably well.

It was found that there are many variables in a CFD simulation that can produce non-desirable results. The MRFSimpleFoam solver was chosen as the SRFSimpleFoam solver did not work as intended. Careful design and implementation of the mesh and choosing correct boundary conditions had to be considered. Modelling a single blade using periodic boundary conditions should work theoretically. However, it was found that during the meshing stage, that this was not practical as the blades on the propeller were close together.

8.0 Recommendations

There are many variations that could be altered in this project to produce different results.

Further trials should be conducted over varying linear velocities (varying cavitation numbers) in order to have conclusive results with comparison to experimental results. This would also give additional analysis of cavitation at varying angles of attack and the distribution of pressure over the blades.

More research can be conducted in relation to the refinement of the mesh, and in particular, an improved way of combining the blade and hub of propeller to the block. In hindsight, using a square block would possibly produce a better quality 'snap' when using SnappyHexMesh as it would have a much lower aspect ratio; "the cell aspect ratio should be approximately 1, at least near surfaces at which the subsequent snapping procedure is applied..." (OpenFOAM User Guide, 2010). However, there would also be extra complexity issues in designing suitable boundary conditions and attaching the propeller to the mesh. A cylindrical block which is used in this project has expanding cells in x direction, which results in a non-smooth snap at the edges of each propeller blade. Using cylindrical surroundings, but a square block, which is equal in x, y and z directions near the propeller may fix the jagged edges and could also provide conclusive evidence of the effect the jagged edges had on the simulation but would also cause other problems snapping to the outside cylinder.

A fully laminar simulation was conducted in this project as no solver had good abilities to replicate the laminar to turbulent transition. It would be interesting to see if there are any variations in the results if different methods to get a fully turbulent solution to converge were used. The turbulence model, $k-\omega$, is widely used as it has shown better potential to predict key features of rotating flows than other models (Rhee, 2005). However, in order to resolve the turbulent boundary layers on solid surfaces, it is best to have growing prismatic cells from the blade and hub surface (Watanabe et al, 2003).

This would mean that the propeller should be meshed using dedicated meshing software. TGrid and GAMBIT are commercial meshing tools that seem to be a popular choice (Watanabe et al, 2003; Rhee, 2005).

Once open water simulation is working well, could look at non-uniform flow. Having a slower linear velocity near the top, would give a higher local angle of attack and therefore, different pressure distributions on each blade.

Additional research on reproducing the thrust and torque coefficients accurately when affected by cavitation is important. Cavitation leads to loss of efficiency (White, 2008; Emerson & Sinclair, 1967) and being able to analyse this effect using CFD would bring a huge advancement to the design of propellers. A method to do this in OpenFOAM is a solver named MultiPhase which models the transformation of water to vapour. However, the process is much more computationally intensive and very sensitive to mesh refinement (Bensow, 2010).

9.0 References

Abott, Doenhoff, 1958, Theory of Wing Sections, Dover Publications

Bhaskaran, Collins, 2012, Introduction to CFD Basics

Biswas, Strawn, 1998, Tetrahedral and Hexahedral Mesh Adaptation for CFD Problems, Applied Numerical Mathematics, Vol. 26, Pages 135-151

Breslin, J.P., *Theoretical and Experimental Techniques for Practical Estimation of Propeller-Induced Vibratory Forces*, TRANS. Vol 78. 1970

Breslin, Andersen, 1994, Hydrodynamics of Ship Propellers, Series 3, Press Syndicate of the University of Cambridge

Celli, Vittorio, 1997, The Kutta-Joukowski Condition, <http://galileo.phys.virginia.edu/classes/311/notes/aero/node4.html> (Accessed July 2012)

CFD-Online, 2012, www.cfd-online.com (Accessed 2011)

Dave Gerr, 1989, Propeller Handbook, Nautical Books

Eisenberg, 2008, Cavitation, <http://web.mit.edu/hml/ncfmf/16CAV.pdf> (Accessed July 2012)

Emerson, Sinclair, 1967, *Propeller Cavitation: Systematic Series Tests on 5- and 6-Bladed Model Propellers*, TRANS. Vol 75, 1967

McCormic, Barnes, 1999, Aerodynamics of V/Stol Flight, Dover Publications

OpenFOAM User Guide. 2010. <http://openfoam.com/docs/user/index.php> (Accessed November 2010)

Rhee, Joshi, 2005, Computational Validation For Flow around a Marine Propeller Using Unstructured Mesh Based Navier-Stokes Solver, JSME International Journal, Series B, Vol 48, No.3

Rickard E. Bensow and Göran Bark, 2010, *Simulating Cavitating Flows With Les In OpenFOAM*, ECCOMAS CFD.

Watanabe, Kawamura, Takekoshi, Maeda, Rhee, 2003, *Simulation of Steady and Unsteady Cavitation on a Marine Propeller Using a Rans CFD Code*, International Symposium on Cavitation, Japan.

White, Frank. 2008. *Fluid Mechanics 6th edition*. New York: McGraw-Hill.

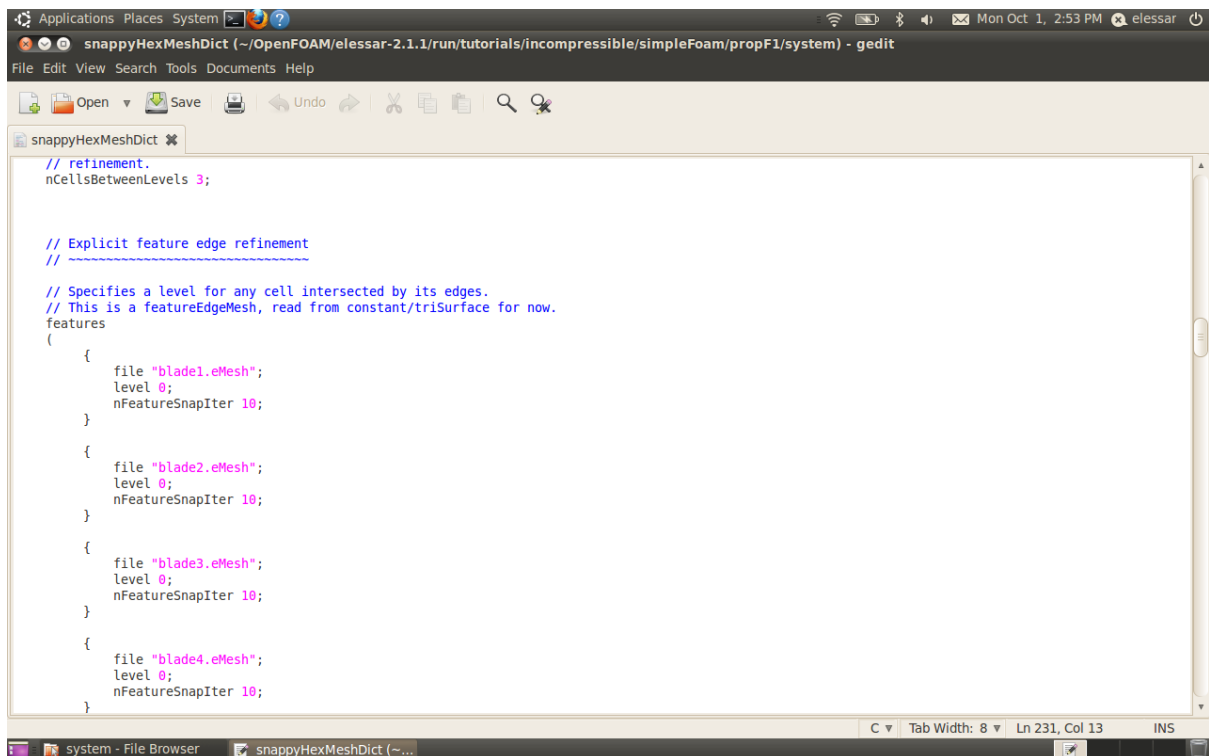
10.0 Appendix

A1: SnappyHexMesh Changes

There were many changes made to the SnappyHexMesh dictionary to remove the jagged edges:

- Refinement of various levels: Up to level 4. Level 5 could not be meshed within 500 million cells
- Used a new Explicit Feature Edge Refinement. From the STL files, the edges were taken out up to a certain angle. 179 took most of them out. The function in SnappyHexMesh (Figure 41) then constructed them back into the mesh.
- Refinement parameters were changed: increasing the max local and global cells and having no minimum refinement.

These changes made little difference to the jagged edges as they are probably the result of a high aspect ratio mesh.



```
// refinement.
nCellsBetweenLevels 3;

// Explicit feature edge refinement
// ~~~~~

// Specifies a level for any cell intersected by its edges.
// This is a featureEdgeMesh, read from constant/triSurface for now.
features
(
  {
    file "blade1.eMesh";
    level 0;
    nFeatureSnapIter 10;
  }

  {
    file "blade2.eMesh";
    level 0;
    nFeatureSnapIter 10;
  }

  {
    file "blade3.eMesh";
    level 0;
    nFeatureSnapIter 10;
  }

  {
    file "blade4.eMesh";
    level 0;
    nFeatureSnapIter 10;
  }
)
```

Figure 41: ScreenCapture from SnappyHexMesh dictionary

A2: surfaceMesh.m

```
%-----  
%E.Colley October 2012  
% To define the mesh in terms of quadrilaterals and then Triangles. Using  
mainMesh as an example (Created by Tim Gourlay)  
%Points ordered anticlockwise (RH rule)  
%writes into a .stl file  
%can refine surface in mainFunc.m  
%-----  
clear  
  
%read in mainFunction  
mainFunc  
  
%number of points - along the x axis  
[m,n]=size(xNew);  
nx = n;  
  
%number of z values (nz)  
nz= length(zNew);  
  
ipp = 0; %initialize total point number  
  
%Assign the points with x,y,z values  
for iz=1:nz  
  
    for ix=1:nx  
  
        ipp= ipp+1;  
        Pts(ipp,:) = [xNew(iz,ix) yNew(iz,ix) zNew(iz)];  
  
    end  
end  
%Pts(ipp,:) = [xNew(iz,ix) yNew(iz,ix) sqrt(((zNew(iz))^2-  
(xNew(iz,ix))^2))];  
%Make the quadrilaterals  
  
iqq = 0;  
  
for iz =1:nz-1  
    for ix=1:nx-1  
  
        iqq = iqq+1;  
        Qua(iqq,:) = [(iz-1)*nx+ix (iz-1)*nx+ix+1 iz*nx+ix+1 iz*nx+ix];  
  
    end  
end  
%Close the Trailing Edge  
iqq = iqq+1;  
Qua(iqq,:) = [(iz-1)*nx+(nx) (iz-1)*nx+1 iz*nx+1 iz*nx+(nx)];  
end  
  
%% split quadrilaterals to form triangles  
TriQua = [Qua(:,1) Qua(:,2) Qua(:,4); Qua(:,2) Qua(:,3) Qua(:,4)];
```

```

%stlwrite(Pts, TriQua)

%convert inch to m
Pts=Pts*0.0254;

%add the other 4 blades
[Pts2, Pts3, Pts4, Pts5] = addBlades(Pts);

%Write each blade to a STL file
nTri = iqq*2;
%blade1 to 5
STLWriting(TriQua, Pts, nTri, 'blade1.stl')
STLWriting(TriQua, Pts2, nTri, 'blade2.stl')
STLWriting(TriQua, Pts3, nTri, 'blade3.stl')
STLWriting(TriQua, Pts4, nTri, 'blade4.stl')
STLWriting(TriQua, Pts5, nTri, 'blade5.stl')

if 0
%plot quads
figure;
hold on
title('quadrilateral mesh')
patch('Faces', Qua, 'Vertices', Pts, 'FaceColor', 'w');
view(3); axis equal;
xlabel('x')
ylabel('y')
zlabel('z')
hold off
end

%plot triangles
figure;
hold on
title('triangle mesh')
patch('Faces', TriQua, 'Vertices', Pts, 'FaceColor', 'w');
patch('Faces', TriQua, 'Vertices', Pts2, 'FaceColor', 'w');
patch('Faces', TriQua, 'Vertices', Pts3, 'FaceColor', 'w');
patch('Faces', TriQua, 'Vertices', Pts4, 'FaceColor', 'w');
patch('Faces', TriQua, 'Vertices', Pts5, 'FaceColor', 'w');
view(3); axis equal;
xlabel('x')
ylabel('y')
zlabel('z')
hold off

```

A3: MainFunc.m

```

%E.Colley October 2012
%List of coordinates of 7 sections of propeller
%runs the other functions

```

%-----

%CHANGEABLE NUMBERS FOR FINER OR COARSER MESH

%number points for back/face (x axis)
nBody= 80;
%number of points around leading edge
nLEdge = 20;
%number of points to tip (z axis)
nHeight = 240;

%offsets

s1 = [0.000 0.021 0.588 0.262 1.175 0.452 1.763 0.569 2.350
0.602 2.743 0.551 3.137 0.425 3.333 0.326 3.530 0.205
0.000 -0.022 0.588 -0.075 1.175 -0.101 1.763 -0.101 2.350 -
0.101 2.743 -0.071 3.137 0.002 3.333 0.054 3.530 0.116];

shiftV1=2.35;
pitch1=12.46;
r1=2;

s2 = [0.000 0.020 0.705 0.217 1.410 0.367 2.115 0.463 2.820
0.492 3.290 0.449 3.760 0.331 3.995 0.239 4.230 0.120
0.000 -0.021 0.705 -0.047 1.410 -0.056 2.115 -0.056 2.820 -
0.056 3.290 -0.051 3.760 -0.020 3.995 0.011 4.230 0.050];

shiftV2=2.82;
pitch2=12.45;
r2=3;

s3 = [0.000 0.020 0.773 0.177 1.545 0.297 2.318 0.372 3.090
0.392 3.607 0.356 4.123 0.259 4.382 0.177 4.640 0.072
0.000 -0.020 0.773 -0.027 1.545 -0.028 2.318 -0.028 3.090 -
0.028 3.607 -0.028 4.123 -0.018 4.382 -0.003 4.640 0.017];

shiftV3=3.09;
pitch3=12.40;
r3=4;

s4 = [0.000 0.016 0.783 0.137 1.565 0.226 2.348 0.279 3.130
0.295 3.667 0.265 4.203 0.186 4.472 0.124 4.740 0.042
0.000 -0.016 0.783 -0.014 1.565 -0.014 2.348 -0.013 3.130 -
0.013 3.667 -0.015 4.203 -0.015 4.472 -0.010 4.740 0.001];

shiftV4=3.13;
pitch4=12.28;
r4=5;

s5 = [0.000 0.014 0.710 0.096 1.420 0.158 2.130 0.194 2.840
0.205 3.370 0.184 3.900 0.124 4.165 0.078 4.430 0.021
0.000 -0.014 0.710 -0.009 1.420 -0.006 2.130 -0.004 2.840 -
0.004 3.370 -0.006 3.900 -0.011 4.165 -0.014 4.430 -0.010];

shiftV5=2.84;

```

pitch5=12.07;
r5=6;

s6 = [0.000 0.013 0.528 0.059 1.055 0.098 1.583 0.122 2.110
0.126 2.600 0.112 3.090 0.073 3.335 0.043 3.580 0.013
0.000 -0.013 0.528 -0.005 1.055 0.002 1.583 0.006 2.110
0.006 2.600 0.003 3.090 -0.004 3.335 -0.009 3.580 -0.013];

shiftV6=2.11;
pitch6=11.75;
r6=7;

s7 = [0.000 0.013 0.370 0.043 0.740 0.067 1.110 0.083 1.480
0.085 1.883 0.077 2.287 0.049 2.488 0.029 2.690 0.013
0.000 -0.013 0.370 -0.004 0.740 0.003 1.110 0.008 1.480
0.008 1.883 0.006 2.287 -0.002 2.488 -0.007 2.690 -0.013];

shiftV7=1.48;
pitch7=11.50;
r7=7.5;

[x1Val y1Val] = interpolate(s1,shiftV1,nBody);
[x1All y1All] = leaedge(x1Val,y1Val,nLEdge,nBody);
[x1RAll, y1RAll] = rotating(x1All, y1All, pitch1,r1);
L1=length(x1RAll);
z1All(1:L1)=2;

[x2Val y2Val] = interpolate(s2,shiftV2,nBody);
[x2All y2All] = leaedge(x2Val,y2Val,nLEdge,nBody);
[x2RAll y2RAll] = rotating(x2All, y2All, pitch2,r2);
L2=length(x2RAll);
z2All(1:L2)=3;

[x3Val y3Val] = interpolate(s3,shiftV3,nBody);
[x3All y3All] = leaedge(x3Val,y3Val,nLEdge,nBody);
[x3RAll y3RAll] = rotating(x3All, y3All, pitch3,r3);
L3=length(x3RAll);
z3All(1:L3)=4;

[x4Val y4Val] = interpolate(s4,shiftV4,nBody);
[x4All y4All] = leaedge(x4Val,y4Val,nLEdge,nBody);
[x4RAll y4RAll] = rotating(x4All, y4All, pitch4,r4);
L4=length(x4RAll);
z4All(1:L4)=5;

[x5Val y5Val] = interpolate(s5,shiftV5,nBody);
[x5All y5All] = leaedge(x5Val,y5Val,nLEdge,nBody);
[x5RAll y5RAll] = rotating(x5All, y5All, pitch5,r5);
L5=length(x5RAll);
z5All(1:L5)=6;

[x6Val y6Val] = interpolate(s6,shiftV6,nBody);
[x6All y6All] = leaedge(x6Val,y6Val,nLEdge,nBody);
[x6RAll y6RAll] = rotating(x6All, y6All, pitch6,r6);
L6=length(x6RAll);
z6All(1:L6)=7;

```

```

[x7Val y7Val] = interpolate(s7, shiftV7, nBody);
[x7All y7All] = lealedge(x7Val, y7Val, nLEdge, nBody);
[x7RAll y7RAll] = rotating(x7All, y7All, pitch7, r7);
L7=length(x7RAll);
z7All(1:L7)=7.5;

xTemp = [x1RAll; x2RAll; x3RAll; x4RAll; x5RAll; x6RAll; x7RAll];
yTemp = [y1RAll; y2RAll; y3RAll; y4RAll; y5RAll; y6RAll; y7RAll];

[xNew, yNew, zNew] = surCreate(xTemp, yTemp, nHeight, nBody, nLEdge);

```

A4: Interpolate.m

```

%E.Colley October 2012
%function to sort the coordinates interpolate the body values

function [xAll, yAll] = interpolate(vector, shift, nBody)

shiftV = [shift];

%grabs the x and y values respectively- X values haven't been shifted yet

X1backT = vector(1,1:2:end-1); % x values of back, from trailing to leading
edge
Y1back = vector(1,2:2:end); % y values of back
X1faceT = vector(2,1:2:end-1); % x values of face, from trailing to leading
edge
Y1face = vector(2,2:2:end); % y values of face

```

```
%shifts the x values to the pivot point - moves the centre point to 0

X1back = X1backT - shift+0.5;
X1face = X1faceT - shift+0.5;

%Interpolates 50 points along the x axis (is the same for back and front)
X1int = linspace(X1back(1),X1back(end),nBody);

%interpolates 50 points for the y front and back using the x value ones
Y1backint = interp1(X1back,Y1back,X1int,'spline');
Y1faceint = interp1(X1face,Y1face,X1int,'spline');

%creates the vector and flips the array of the face(bottom).
xAll = [X1int fliplr(X1int(2:end))];
yAll = [Y1backint fliplr(Y1faceint(2:end))];
```

A5: rotating.m

```
%E.Colley October 2012
%function to rotate the values according to pitch. Angle = tan-
1(p/(2*pi*r))

function [xRAll, yRAll] = rotating(xAll,yAll,pitch,radius)

angle = atand(pitch/(2*pi*radius));

xyMatrix = [xAll;yAll];

%Rotates anti-clockwise through angle.
R= [cosd(angle) -sind(angle); sind(angle) cosd(angle)];

xyRotAll = R*xyMatrix;

xRAll= xyRotAll(1,1:end);

yRAll = xyRotAll(2,1:end);
```

A6: lealedge.m

```
%E.Colley October 2012
%function to create the leading edge values

function [xAll, yAll] = lealedge(xVal, yVal,nLEdge,nBody)

%get the x and y values around the leading edge.
ex1 = [xVal(nBody-1:nBody)];
ex2 = [xVal(nBody+1:nBody+2)];
ex = [ex1 ex2];

why1 = [yVal(nBody-1:nBody)];
why2 = [yVal(nBody+1:nBody+2)];
why = [why1 why2];

%The value of b in the circle equation (midpoint of y values)
b = (why(2)+why(3))/2;

%gradient at x,y (point before leading edge): top, bottom, average
m1 = (why(2)-why(1))/(ex(2)-ex(1));
m2 = (why(3)-why(4))/(ex(3)-ex(4));
m = (m1-m2)/2;

x = ex(2);
x1 = ex(3);
y = why(2);
y1 = why(3);
%get the value of a
a = x + m*(y-b);

%get the value of c
c = sqrt((x-a)^2 + (y-b)^2);

%get the leading edge values:

%grab the angle of the circle to match up with points
angle = atan((y-b)/(x-a));
angle2 = atan((b-y1)/(x1-a));

%counter used to store the values
ii = 1;

%from the bottom x,y value get values from the circle (leading edge) to the
top x,y value
for jj=-angle2:0.01:angle
    temp = [a+c*cos(jj)];
    temp1 = [b+c*sin(jj)];

    xCirT(ii) = temp;
    yCirT(ii) = temp1;
    ii = ii+1;
end
```



```

%values stored in xCirT and yCirT are from bottom to up
%Reverse this vector so that it is top to bottom

xCir =xCirT(end:-1:1);
yCir =yCirT(end:-1:1);

%evenly space out the y values to get the same amount of x values
%-->same amount of values for each section

yCirN = linspace(yCir(1), yCir(end), nLEdge);
xCirN = interp1(yCir, xCir, yCirN, 'spline');

xAll = [xVal(1:nBody-1) xCirN xVal(nBody+1:end)];
yAll = [yVal(1:nBody-1) yCirN yVal(nBody+1:end)];

```

A7: surCreate.m

```

%E.Colley October 2012
%A function to create the surface of the propeller blade. Adds the tip
values.

function [xNew, yNew, zNewL] = surCreate(xVal, yVal, nHeight, nBody,
nLEdge)

zVal =[2 3 4 5 6 7 7.5];

zNew = linspace(zVal(1), zVal(end), nHeight);

%-----
%TIP SURFACE
%Getting the Z tip
zValT =[2 3 4 5 6 7 7.5 8];

zNewT = linspace(zValT(1), zValT(end), nHeight);
%at trailing edge
zTX = interp1(zVal, xVal(:,1), zNewT, 'spline', 'extrap');
zTY = interp1(zVal, yVal(:,1), zNewT, 'spline', 'extrap');
%at leading edge
zLX = interp1(zVal, xVal(:,nBody), zNewT, 'spline', 'extrap');
zLY= interp1(zVal, yVal(:,nBody), zNewT, 'spline', 'extrap');

%interpolating the x and y values at tip
tipLineX = linspace(zTX(end), zLX(end), nBody);
tipLineY = linspace(zTY(end), zLY(end), nBody);

%Add the values of the leading edge at the tip
zTipEdgeX(1:nLEdge) = tipLineX(end);
zTipEdgeY(1:nLEdge) = tipLineY(end);

%reverse the body values (opposite/face)
tipLineRX=tipLineX(end:-1:1);
tipLineRY=tipLineY(end:-1:1);

```

```

%adding the tip together with the other values
tipLineX = [tipLineX(1:end-1) zTipEdgeX tipLineRX(2:end)];
tipLineY = [tipLineY(1:end-1) zTipEdgeY tipLineRY(2:end)];
%-----
%LOWER SURFACE

zValL = [1.5 2 3 4 5 6 7 7.5 8];

zNewL = linspace(zValL(1), zValL(end), nHeight);

for jj = 1:1:length(xVal)
    xBottom(:,jj) = interp1(zVal,xVal(:,jj),zNewL,'spline','extrap');
    xBot(:,jj)=xBottom(1,jj);
    yBottom(:,jj) = interp1(zVal,yVal(:,jj),zNewL,'spline','extrap');
    yBot(:,jj)=yBottom(1,jj);
end

%-----
%add to the x/y matrix to obtain the surface
xValN = [xBot;xVal;tipLineX];
yValN = [yBot;yVal;tipLineY];
%interpolate and create the surface.

for ii = 1:1:length(xValN)

    xNew(:,ii) = interp1(zValL, xValN(:,ii), zNewL, 'spline');

    yNew(:,ii) = interp1(zValL, yValN(:,ii), zNewL, 'spline');

end
end

```

A8: STLWriting.m

```

%E.Colley October 2012
%function to write the .stl file for triangle mesh

function STLWriting(TriQua, Pts, nTri, fileName)

%get the xyz coordinates for 1st, 2nd, 3rd point at each triangle.
a = Pts(TriQua(:,1),:);
b = Pts(TriQua(:,2),:);
c = Pts(TriQua(:,3),:);

%the facet normal is the cross product. RH rule
fn = cross((b-a),(c-a));

size(fn)

%Open file for writing

% fid = fopen('triangleProp.stl','w+');
fid = fopen(fileName,'w+');

```

```

% Write the file contents
% Write HEADER
    fprintf(fid, '%s\n', 'solid prop');

% Write DATA

    for ii=1:nTri
        fprintf(fid, '%s\n', ['facet normal ', num2str(fn(ii, :))]);
        fprintf(fid, '%s\n', 'outer loop');

%Write the vertex number for each point in the triangle
        for jj=1:1:3
            fprintf(fid, '%s\n', ['vertex
', num2str(Pts(TriQua(ii, jj), :))]);
        end

        fprintf(fid, '%s\n', 'endloop');
        fprintf(fid, '%s\n', 'endfacet');
    end

% Write FOOTER
    fprintf(fid, '%s\n', 'endsolid prop');

% Close the file
    fclose(fid);

```

A9: addBlades.m

```

%E.Colley October 2012
%function to add the blades of the propeller

function [Pts2, Pts3, Pts4, Pts5] = addBlades(Pts)

angle2=72;

angle3=144;

angle4=216;

angle5=288;

%Rotates anti-clockwise through angle using a 3D rotation matrix about the
y axis

R2= [cosd(angle2) 0 sind(angle2); 0 1 0; -sind(angle2) 0 cosd(angle2)];
R3= [cosd(angle3) 0 sind(angle3); 0 1 0; -sind(angle3) 0 cosd(angle3)];
R4= [cosd(angle4) 0 sind(angle4); 0 1 0; -sind(angle4) 0 cosd(angle4)];
R5= [cosd(angle5) 0 sind(angle5); 0 1 0; -sind(angle5) 0 cosd(angle5)];

Pts2 = Pts*R2;

```

```
Pts3 = Pts*R3;  
Pts4 = Pts*R4;  
Pts5 = Pts*R5;
```