QUEEN'S UNIVERSITY BELFAST

SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING

Final Project Report

AER4002

Computational methods for generating swirling inlet boundary conditions for CFD

Student	Gerico Vidanes [40170738]
Supervisor	Dr. Marco Geron
Programme	MEng Aerospace Engineering
Date	20th March 2021

Summary

Non-uniform flow is present in almost all systems of engineering interest. The analysis of these distorted flows has become increasingly important as the aerospace industry continues to seek increased performance and efficiency within its products. Swirling inlet flows have the potential to be either detrimental or advantageous to aeroengine performance therefore it is important to understand their behaviour and development. In recent years, an effective method for generating swirling inlet flows for wind tunnel experiments (StreamVane™) has been developed. Parallel to this, the CFD space is actively developing more accurate and efficient ways of generating boundary conditions for numeric simulations. At the intersection of these two fields, there is work to be done on expanding the capability of generating realistic swirling inflow boundary conditions for use in numerical analysis.

This report proposes various computational methods designed to aid this goal and integrates them into workflows which can generate incompressible inlet conditions useable within a CFD framework. One such method superimposes the velocity effects of discrete vortex models to parametrically create arbitrary swirling distributions. In addition, a contour translation method has been implemented whereby the underlying numerical data illustrated by a contour plot within an image can be extracted. Both methods have been integrated into a Python workflow to produce swirling inlet condition files which are immediately useable within a 3D SU2 simulation. This end-to-end process is achieved by also providing independent modules capable of extracting the inlet nodes from a 3D mesh (.su2 format) and writing the flow field data in a format recognisable by the SU2 framework. An accompanying open-source Python implementation has also been released on GitHub[1].

Both methods proposed can recreate the idealised swirl cases described in the Aerospace Information Report 5686[2] and write them into inlet condition files for simulation. The resulting simulations are numerically stable and show good agreement with physical reasoning. The capability of modelling a developed boundary layer within the inlet condition has additionally been explored. Integrating these capabilities demonstrates the possibility of allowing the elimination of the buffer region immediately downstream of the inlet plane. More work is needed to develop and validate this; however, it has the potential of allowing the truncation of CFD domains resulting in significant computational savings.

Acknowledgements

Firstly, I would like to acknowledge and thank my supervisor for this project. The independence Dr. Marco Geron encouraged fostered both the project and my own development while also exposing my proficiency and enjoyment of the research and development process. His guidance was invaluable when the project faced crossroads and his comments on drafts of this report helped me to present the work in a clear way. I would also like to acknowledge Donal McCaughey, a PhD student providing support for this Master's project. Donal was always available for my queries through Microsoft Teams and gave indispensable advice on areas like vortex modelling, SU2, and CFD in general. Finally, I would also like to thank Dr. Rob Watson for providing key guidance on boundary layer modelling and helping me to implement this quickly as the project was coming to a close.

Table of Contents

Project Description ..... **Error! Bookmark not defined.**

Summary i

Acknowledgements ii

Nomenclature v

1 Introduction 1

2 Literature/Technology Review 2

2.1 Swirling flow 2

2.1.1 Characterisation 2

2.1.2 Generation 3

2.2 CFD and SU2 4

2.3 Python and its libraries 4

2.4 Requirements specification 5

2.4.1 Goal 5

2.4.2 Problem breakdown 5

3 Proposed Solution 7

3.1 Overview 7

3.2 Method 1 – Vortex Method 7

3.2.1 Generating realistic swirl 7

3.2.2 Extraction of inlet nodes 10

3.2.3 Inlet condition file output 12

3.3 Method 2 – Contour Translation Method 13

3.3.1 Introduction 13

3.3.2 Segmentation 13

3.3.3 Plot sampling 15

3.3.4 Inverse colour mapping 15

3.3.5 Integration with boundary generation 16

3.4 Summary of process 17

3.5 Python implementation 18
4 Development 19
4.1 Image translation 19
4.1.1 Segmentation performance 19
4.1.2 Sampling parameters & accuracy 20
4.2 Fit-for-purpose testing 22
4.2.1 Idealised swirl cases 22
4.2.2 Boundary flux 24
4.2.3 CFD simulations in SU2 25
4.3 Boundary layer modelling 29
4.3.1 Method 29
4.3.2 Validation 30
5 Sustainability 32
6 Future Work 33
6.1 Limitations and suggestions 33
6.2 Going forward 33
7 Conclusion 34
8 References 35
9 Appendix A – Solid boundary modelling need 39
10 Appendix B – Code architecture 40
11 Appendix C – Modelled boundary layer development 42
12 Appendix D – Supervisor Meeting Minutes ..... **Error! Bookmark not defined.**
13 Appendix E – Gantt chart ..... **Error! Bookmark not defined.**

Nomenclature

Symbols:

α	-	Radial flow angle
β	-	Tangential flow angle
V_θ	-	Tangential component of velocity
V_r	-	Radial component of velocity
V_z	-	Axial component of velocity
$\boldsymbol{x}$	-	Cartesian coordinate vector
x, y	-	Orthogonal cartesian components of distance
r	-	Radial component of polar coordinate
θ	-	Tangential component of polar coordinate
Γ	-	Vortex circulation
Ω	-	Vortex strength
a	-	Vortex core radius
u	-	Cartesian x component of velocity
v	-	Cartesian y component of velocity
x_c, y_c	-	2D cartesian coordinates of vortex centre
δ	-	Boundary layer thickness
Π	-	Boundary layer wake parameter
κ, B	-	Constant parameters for boundary layer models
ν	-	Kinematic viscosity
c_f	-	Skin friction coefficient
Re_x	-	Reynold's number at position x
u^+	-	Non-dimensional velocity
${u_\infty}^+$	-	Dimensionless freestream velocity
u_∞	-	Freestream velocity
u_τ	-	Friction velocity
$\hat{\boldsymbol{u}}$	-	Flow direction unit vector
$\boldsymbol{u}$	-	Flow velocity vector
y^+	-	Non-dimensional wall distance

Abbreviations:

10D	-	10 diameters
2D	-	Two dimensional
3D	-	Three dimensional
AIR	-	Aerospace Information Report
ASCII	-	American Standard Code for Information Interchange
BL	-	Boundary layer
CFD	-	Computational Fluid Dynamics
GUI	-	Graphical User Interface
LES	-	Large Eddy Simulation
MIT	-	Massachusetts Institute of Technology
RANS	-	Reynolds Averaged Navier-Stokes
RGB	-	Red, Green, Blue
STEM	-	Science, technology, engineering, and mathematics

1 Introduction

Swirling flow is common within areas of aerospace engineering interest, especially in the domain of engine design – both subsonic and supersonic. Jet engines have been traditionally designed whilst considering ideal, uniform inlet flow. However, as the aerospace industry continues to strive for increased performance and optimisations, the effect of the inevitable presence of distorted flows becomes significant. The complex interaction between an aeroengine and its airframe is a common cause of sustained inlet distortions. A classic case study of this involved the Tornado fighter aircraft[3], where engine surges could occur when flying at high angles of attack or high Mach numbers. Here, the problem was identified and mitigated in the late stages of design. As engine integrations become increasingly complex, the effects of inlet swirl are considered much earlier in the design process. This is illustrated in the analysis of Boeing's hybrid wing body concept[4]; the engines' reaction to the distorted flow caused by boundary layer ingestion is given detailed consideration at the conceptual design stage. Interest on swirling flow extends to supersonic engines and internal aerodynamics; non-uniform flow in scramjet/ramjet combustion chambers induces fuel-air mixing and can significantly affect combustion performance[5]–[9].

It is apparent then that swirling flow is an active and necessary research area, and unsurprisingly much work has been done towards the understanding of the effect and development of swirl. One such significant work outlines an effective method for generating swirling inlet flow for wind tunnel experiments, the StreamVane™ method[10]. The present report outlines work in developing methods which provide analogous capability but within the computational domain. Simulation and numerical analysis offer reduced costs and increased cycle time compared to experimental studies, in addition to allowing the analysis of rare or extreme cases which can be difficult to replicate in an experimental setting.

Within the CFD space, there is ongoing research on the generation of inlet boundary conditions within the context of turbulent LES simulations[11]; this area calls for inflow conditions which are already realistically turbulent. In this work we follow the same idea towards a largely convergent goal but seek to generalise across flow regimes and solver types while seeking specificity towards vortical flows. Having tools to generate boundary conditions which are already swirling realistically (bypassing the need to model, mesh and simulate upstream vortex generators) can potentially result in less computational cost, shorter simulation times, and faster research as a whole.

2 Literature/Technology Review

2.1 Swirling flow

2.1.1 Characterisation

Much work has been done to simulate swirling engine inlet flow from varying causes[3], [4], [12]. To this end, an Aerospace Information Report (AIR 5686) was published by the S-16 Turbine Inlet Flow Distortion Committee which outlines a method for characterising swirl distortion, including quantitative swirl descriptors[2]. The basic angle definitions are utilised in this work.

$$\beta = \tan^{-1}\left(\frac{V_\theta}{V_z}\right), \quad \alpha = \tan^{-1}\left(\frac{V_r}{V_z}\right)$$

Equation 1: Tangential and radial flow angles.

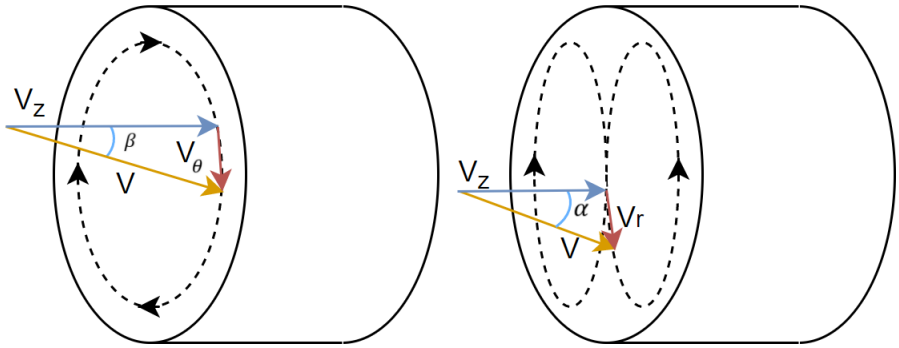

Figure 1: Flow angles relative to velocity components. (Left) Bulk swirl with only tangential flow. (Right) Counter-rotating swirl showing a point with a high radial flow component.

β in Equation 1 represents the tangential flow, or swirl, angle and α represents the radial flow angle; where V_θ is the tangential velocity, V_r is the radial velocity, and V_z is the axial velocity. Figure 1 illustrates the relevance of these as a description of the distortion of the flow. It is common within the literature to report a distorted flow via a contour plot of swirl angle along with the streamline plot as in Figure 2.

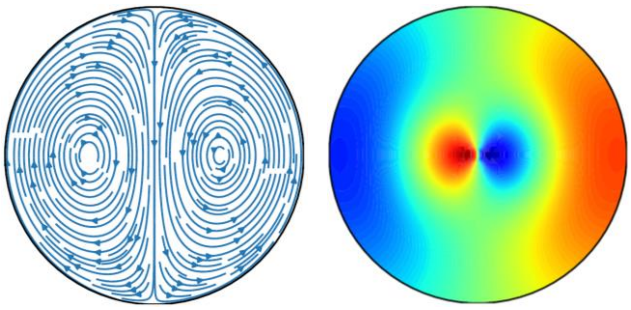

Figure 2: Streamline and swirl angle contour plot of a counter-rotating vortical flow within a pipe.

Further material useful for this work are the idealised swirling test cases detailed by K. Smith[13]; following the lead of the AIR5686[2] and providing simplified swirling cases composed of elementary vortical structures. The four reported swirls consist of a bulk swirl case, created by circumferentially sweeping a linear radial swirl profile, and three dual counter-rotating vortex systems, one of which is symmetrical.

2.1.2 Generation

A relatively recent method has been developed for generating swirling inflows to aid inlet distortion research in the experimental domain. The 'StreamVane™ Method' by K. Hoopes is able to generate a 3D part which, when placed in a wind tunnel, 'turns' the originally uniform flow into a given arbitrary distortion profile[10]. The method begins with a 2D velocity vector field as input; curves are then drawn which are perpendicular to the velocity field at every point, these are the so called 'vane' lines. To then create the 3D geometry for manufacture, an appropriate aerofoil profile is extruded along these lines (along with non-aerodynamic struts for support); these are the main functioning components which distort the incoming air and produce the desired swirl. The original paper shows promising results, and the method has been shown to successfully scale in both size and flow conditions[14]. A representative geometry of a StreamVane™ part is shown in Figure 3.

Figure 3: CAD model of a StreamVane™ part, taken from [39].

While the current work's aim is analogous to Hoopes', the necessary implementation within the computational domain is quite different; after all, if the velocity field is already available, it is relatively trivial to format this in a way that is useable within a CFD framework. Rather, the numerical equivalent is to parameterise the specification and creation of swirling inlet boundary conditions to minimise costly simulation. To this end, Pierce and Moine[15] and Jiang et al.[16] describe numerical and analytical methods respectively for generating inflow boundary conditions. In the first case, a general confined swirling case is solved, and the second case provides formulae for describing swirl produced by round and annular jets. Both works specifically state that the validity of their models is confined to 'equilibrium' swirling flows, i.e., when the flow is fully developed - for instance in the case where the vortex generator is sufficiently far upstream. Pierce agrees that this is a restrictive constraint and as a result will not cover all analyses. The aim in this work is to expand these capabilities utilising powerful computational tools and methods to further aid aeroengine research.

2.2 CFD and SU2

The ongoing exponential improvements to computation have allowed the widespread use of computational fluid dynamics[17]. The ability to simulate cases of engineering interest without costly and sometimes difficult experiments has no doubt advanced the field exponentially. To this end, there exists a growing ecosystem of CFD frameworks, both commercial and open-source, to aid the research and analysis effort. One such software package is SU2[18], an open-source multi-physics simulation suite. SU2 is freely available and is a fully capable CFD framework, allowing the simulation of non-trivial 3D cases with little difficulty. The open-source nature of the code also makes the development of a peripheral tool less complicated.

The fundamental function of CFD codes is the solution of discretised partial differential equations; the discretisation of a traditionally continuous area of mathematics[19] is necessary if the power of computation is to be leveraged. The implications and specifics of discrete calculus are outside the scope of this work but should be noted as the possible cause of non-physical emergent results. Nevertheless, the solution of differential equations requires the specification of boundary conditions. The selection of appropriate boundary conditions effectively defines the physical scenario being modelled and the values given to these boundary conditions have a direct impact on the ultimate solution obtained. One such boundary condition which is implemented in all major CFD packages is the inlet boundary[20]. This allows the movement of transported variables into the domain, either by specifying uniform total conditions or by specifying a profile across the inlet plane. The inlet condition defines the evolution of the rest of the domain and thus accuracy is paramount.

2.3 Python and its libraries

Python is a popular high-level interpreted language with easy-to-understand syntax[21]; this makes resulting programmes generally more useable and expandable by other engineers and can be easily integrated into existing workflows. As a dynamically typed programming language, it is particularly suited to rapid prototyping and is ideal for this type of proof-of-concept work. Python is an ideal integration platform and can be used to build robust and sophisticated programmes by utilising powerful libraries with backends written in more low-level languages.

One such library that is essential for this work is NumPy[22]. It provides support within Python for manipulating multi-dimensional arrays as well as providing a large collection of high-level mathematical functions – similar to MATLAB. The performant matrix operations are essential in this work; trying to do these operations in native python would be prohibitively slow[23]. NumPy exists

within the larger SciPy ecosystem[24], providing open-source tools for STEM; the namesake SciPy library[25] is also used here for its performant optimisation and interpolation functions.

Another library that this work relies on is OpenCV[26], originally developed by Intel. It is an open-source library of computer vision and image processing tools; the relevance of which will be seen later in this report. OpenCV is optimised for real-time applications and is therefore computationally efficient. It implements common low level image processing operations like colour transformations, edge detection, and image resizing; as well as increasingly high-level computer vision operations like object segmentation and face recognition[27]. The widespread use of OpenCV in the computer vision field speaks to the quality of its implementations[28]. However, since the library uses computer vision algorithms from literature, the disadvantages of these algorithms in general become relevant. Ambiguity is the main challenge of computer vision[29]; as a result, most classical rule-based algorithms rely on threshold values to correctly detect structures within images. The onus is on the user to use appropriate threshold values for their application; the implications and selection of these within this work will be discussed.

2.4 Requirements specification

2.4.1 Goal

The aim of this work is summarised as follows:

To develop a tool which can generate realistically swirling 3D flows and write them as boundary conditions useable within a CFD framework.

2.4.2 Problem breakdown

It is necessary to unpack the stated goal and specify requirements which can be used to evaluate the proposed solution. The main ideas are isolated and discussed in the following.

Realistic swirl

Defining realistic flow fields for developed non-elementary flows without simulation is not trivial[15], [16]. An arbitrarily defined flow field may be mathematically correct but represents a non-physical scenario. In this work, CFD simulations are used to judge the realism of the boundary condition but ideally, they should be experimentally validated.

3D Flow

The ultimate simulations will be 3D domains to properly capture the development of the swirl, therefore the inlet boundary is a 2D slice with each node having 3 dimensional properties. It is also noted that this work, as is common in this area[30], focuses on streamwise swirl (i.e., off-angle velocities are concentrated in the plane perpendicular to the flow)

Useable by a CFD framework

To be used as a boundary condition for a simulation, the inlet flow data needs to be written in a format readable by a CFD framework. Since there are many CFD packages, this data formatting should be modularised so that the underlying boundary generation methods can be used across frameworks. Furthermore, the data should be such that it results in a numerically stable solution.

Ultimately, the inlet conditions will need to match the discretisation being used by the CFD simulation. At the time of writing, SU2 requires a 1-to-1 point matching of the values and does not support interpolation[31], therefore the final output of the proposed tool should describe the inlet flow with the same nodes as the mesh being used for simulation. This can be achieved either by interpolating to the desired mesh or by using the same discretisation throughout; in either case, capability for extracting node positions of the inlet from the user's mesh is needed.

General design guidelines

As part of the overall design/analysis process, the properties of the generated flow fields should be easily controllable by the user. Considering the target user base, extensive visualisation and intermediate data availability should also be available to further aid the engineering process. In terms of the code implementation, the author has decided that flexible disclosure of complexity would aid the useability without compromising the user's control – discussed further in this report.

3 Proposed Solution

3.1 Overview

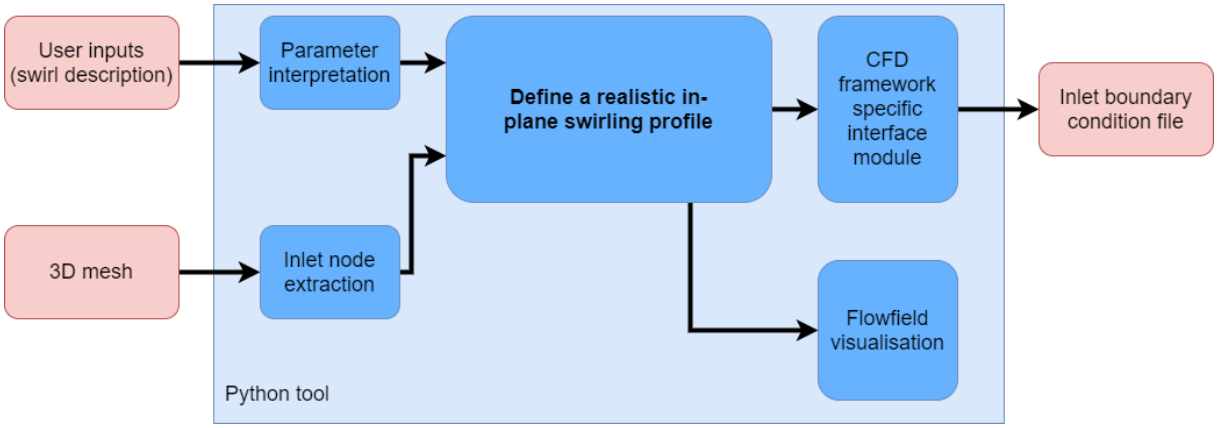

Figure 4: Top-level functional architecture.

Figure 4 shows the basic functional architecture necessary to achieve the identified requirements. The remainder of this report section will detail the methods underlying these functions and their specific implementations. Note that as a proof-of-concept, boundary conditions for incompressible flow were generated therefore only a velocity profile was needed to fully define the inlet. In addition, while the underlying methods proposed are theoretically agnostic to domain shape, the implementations presented here are for a circular inlet for simplicity.

3.2 Method 1 – Vortex Method

3.2.1 Generating realistic swirl

Discrete Vortices

Generation of streamwise swirl can be simplified into specifying a 2D swirling flow and imposing a uniform axial velocity to complete the 3D flow field. This is analogous to what a StreamVane™ part does within a wind tunnel. Therefore, the definition of a 2D velocity profile is the subject of this section. As discussed previously, the AIR5686 attempts to characterise swirl distortions[2] and shows that they can be largely broken down into elementary vortical structures. The same approach is taken here; it is assumed that an arbitrary continuous vorticity distribution can be modelled by superimposing the velocity effects of discrete vortices. With this presumption, the problem of realism can be implicitly solved by utilising vortex models from literature[32]. The Lamb-Oseen vortex model has been chosen in this work as is common in literature[10], [13], [33], in addition to its ease of computation and variation (controlling variables are explicit within the equations). This method is similar to that used by Virginia Tech's in-house vortex generator function 'nVort', mentioned in Smith's paper[13], and as

such results from this can be used for validation. Note that the following methods outlined in this report are agnostic to the vortex model used and the accompanying code modularises this allowing for the implementation of other models, as will be seen in section 4.2.1.

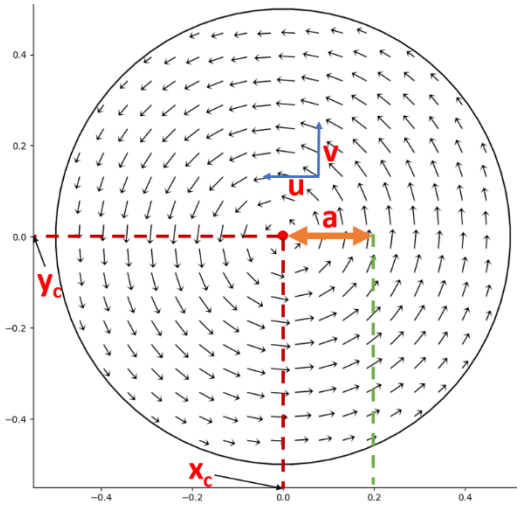

Figure 5: A single Lamb-Oseen vortex with its parametric descriptions shown.

$$u = \frac{1}{2}\frac{a^2\Omega(y-y_c)}{(x-x_c)^2+(y-y_c)^2}\left(1-e^{-\frac{(x-x_c)^2+(y-y_c)^2}{a^2}}\right)$$

Equation 2: x-component of velocity for a Lamb-Oseen vortex.

$$v = -\frac{1}{2}\frac{a^2\Omega(x-x_c)}{(x-x_c)^2+(y-y_c)^2}\left(1-e^{-\frac{(x-x_c)^2+(y-y_c)^2}{a^2}}\right)$$

Equation 3: y-component of velocity for a Lamb-Oseen vortex.

The governing equations have been reproduced above. Equation 2 and Equation 3 define the velocity field of a 2D domain under the influence of a Lamb-Oseen vortex; where a is the vortex core radius, $\Omega = -\frac{\Gamma}{\pi a^2}$, Γ is the circulation (vortex strength), and x_c and y_c are the coordinates of the vortex centre. As shown in Figure 5, the 'core' described by the radius a does not define the borders of the vortex. The term vortex core has no consistent definition within the literature[33]. In this work it is only used in so far as it is relevant to Equation 2 and Equation 3, and is observed to mean the radius after which the vorticity effect of the vortex starts to decrease.

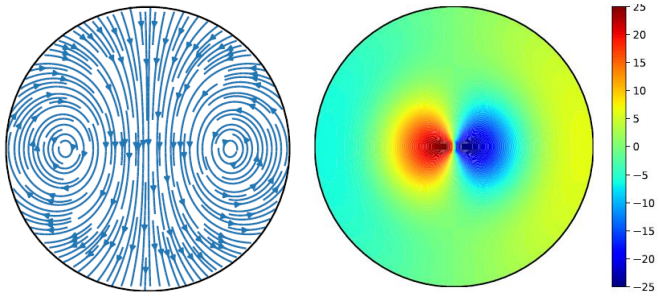

Figure 6: Streamlines (Left) and swirl angle distribution (Right) for two counter-rotating Lamb-Oseen vortices.

A flow field produced by superimposing two Lamb-Oseen vortices with equal and opposite strengths can be seen in Figure 6. No solid boundary has been modelled and the effect of the vortices is infinite, albeit diminishing. In the original StreamVane™ paper this was a non-issue since the velocity profile

was simply used to generate the geometry – boundary effects are enforced by the real fluid interacting with the actual walls. Here, it is necessary to explicitly enforce this boundary to ensure there is no flux across the walls; this was presumed by intuition, but proof can be seen in Appendix A.

Solid boundary

To model the solid walls of the duct surrounding the vortex system, the method of images[34] is utilised following the lead of nVort[13]. Essentially, the effect of a solid boundary can be modelled by the effect of an equal and opposite vortex at the 'inverse' position to the original vortex. For a simple straight wall case, a solid boundary at a distance *L* from a vortex can be modelled by an equal and opposite vortex at a distance *2L* from the original vortex – illustrated in Figure 7.

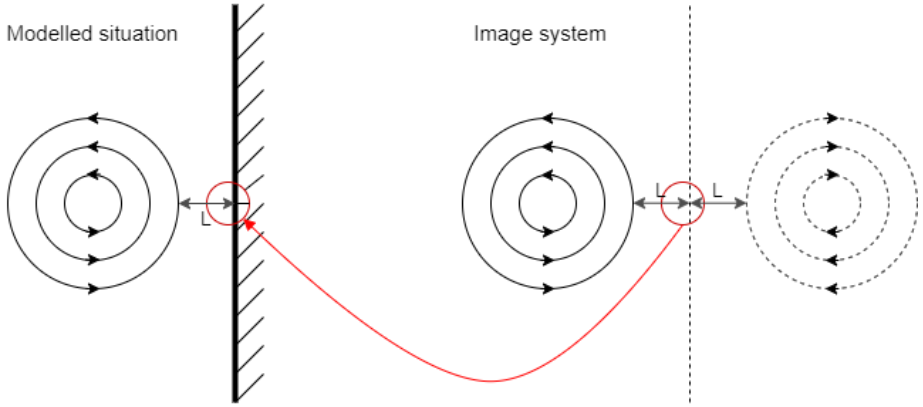

Figure 7: How the conditions at a solid boundary can be modelled with an image vortex.

For the case of a vortex system enclosed by a circular wall, circle theorem[34] is used which defines the inverse position as in Equation 4; where $\boldsymbol{x_i}$ is the 2D vector coordinate of the image vortex, $\boldsymbol{x_o}$ is the 2D vector coordinate of the original vortex, R is the circle radius, and r is the radial position of the original vortex relative to the circle's centre. Figure 8 illustrates these variables in relation to the vortices and the solid boundary; it is assumed that the coordinate system used is coincident with the circle's centre.

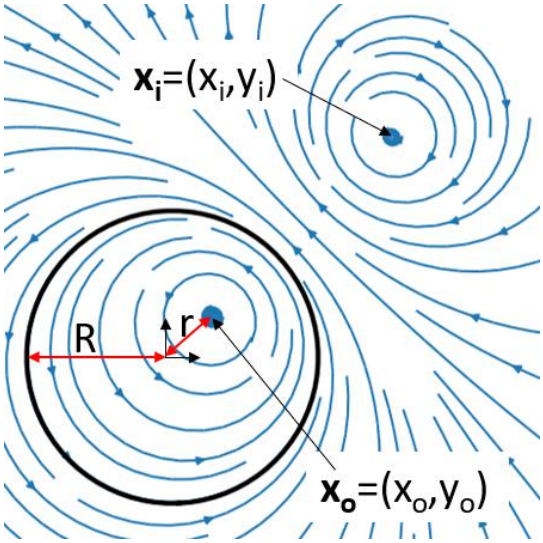

Figure 8: Circle theorem variables illustrated.

$$\boldsymbol{x_i} = \frac{R^2}{r^2}\boldsymbol{x_o}$$

Equation 4: Calculation of the inverse position across a circle.

This method suggests a general applicability to duct shapes; however, when parallel walls are introduced, an infinite series of image vortices are needed to satisfy the symmetry condition as illustrated by MIT[35]. This could be approximated numerically by iteratively applying symmetry and adding vortices until the flux across the boundaries is decreased to some tolerance; this has not been implemented in this work. Modelling solid boundaries for non-circular inlets with low computational cost is left for future work; the underlying method described in this work is again agnostic to this and should be useable regardless of changes to the modelled flow field.

The method of images implemented here allows for computationally efficient consideration of the solid wall as there is no need to simulate the wall adjacent flow and the already present vortex models can be utilised. Figure 9 shows the same twin vortex system from Figure 6 but with the method of images applied; the image vortices act to compress the streamlines and enforce the solid wall.

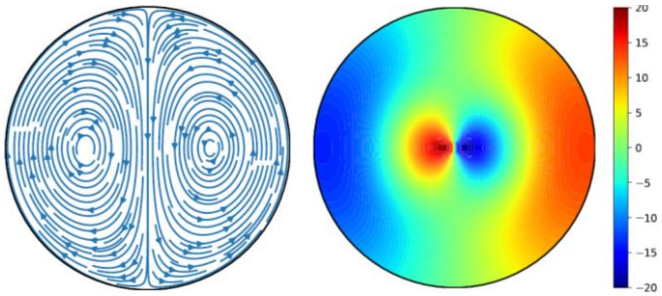

Figure 9: Streamlines (Left) and swirl angle distribution (Right) for two counter-rotating Lamb-Oseen vortices with the boundary modelled with image vortices.

So far, only the no-penetration boundary condition has been modelled, not the no-slip condition. In the first instance, it was decided that the inclusion of this within the generated boundary condition was not crucial; a proper boundary layer can instead be allowed to develop along a buffer region within the ultimate CFD simulation. The implications of this are seen and addressed in section 4.3.

3.2.2 Extraction of inlet nodes

Being in the computational domain, the operations for defining the flow field are applied on a discretised domain of nodes. As discussed in section 2.4.2, at some point before file writing, the flow field discretisation needs to match that of the 3D mesh to be used in the target CFD simulation. In this work, it was decided that the same discretisation would be used to represent the flow field internally – i.e., the values of the fields which represent the swirling flow slice is defined at the same points as that of the user's mesh. This avoids any additional interpolation errors and makes the final data output easier.

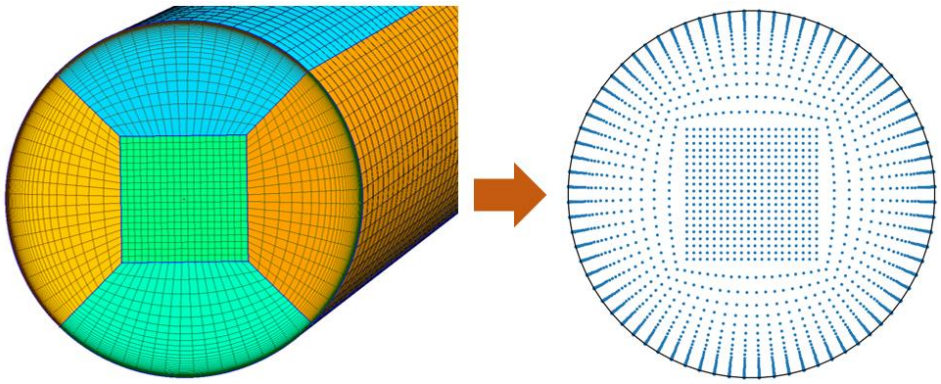

Figure 10: The nodes forming the inlet boundary (Right) extracted from a 3D mesh (Left) made using Gmsh[36].

To achieve this, the positions of the nodes which form the inlet plane of the 3D mesh needs to be extracted, as in Figure 10. The method here, illustrated best as pseudocode shown in Figure 11, and the implementation in the accompanying Python code, is aimed at the *.su2* mesh file format and assumes that the inlet face is a physical group within the domain labelled 'inlet' (case insensitive). The details of the *.su2* file format are not discussed here but can be easily found within the SU2 documentation[37]. Generalising to other mesh file formats is out of the scope of this project, however, the underlying boundary condition generation method outlined in this work is independent of the mesh format.

Inlet nodes extraction from mesh file (pseudocode)

```
Lines ← all lines in file as an array
# Find section markers
For line in Lines:
        If line is 'inlet':     inlet_index  ← line number
        If line is 'NPOIN': point_index ← line number
        If both found:      break from loop
number_of_elements ← Lines[inlet_index+1] as a number
# Information on which nodes make up the inlet is stored by specifying the vertices of each mesh element which make up a named group
For i = inlet_index+2 to inlet_index+2+numElem:
        Inlet_element ← Lines[i] as numbers
        # First number in tuple is element type
        Append Inlet_element[2nd to end] to Node_indices
Discard duplicates in Node_indices
For each index in Node_indices:
        # Get 3D coordinates
        Append first 3 numbers of Lines[point_index+1+index] to Nodes
```

Figure 11: Pseudocode for extraction inlet nodes from a .su2 file.

A simple cylindrical mesh was used throughout this work, to emulate the development of the swirling flow within an internal domain. A multi-structure mesh was applied to the 2D circular faces, as shown in Figure 10. This preserves a good circumferential resolution at the edges while avoiding unnecessarily high node concentrations within the centre as would be the case for a purely radial distribution. A bias is applied to the radial node distribution to allow for decreasing cell size towards the walls for tuning the dimensionless wall distance[38], y^+. This mesh cross section is extruded along the duct and the lengthwise cells are uniformly distributed. The numerical details of the mesh and domain will be discussed as relevant within this work as context arises.

3.2.3 Inlet condition file output

To achieve the ultimate purpose of this work, it is necessary to write the flow field data to a file format that is useable by the chosen CFD framework as an inlet boundary condition, SU2 in this case. This is relatively trivial since the flow field has been defined at points which match the discretisation of the user's 3D mesh – no interpolation or transformation is necessary. The SU2 framework allows the specification of non-uniform velocity profiles for the inlet by way of an ASCII file with specific headings[31] – values at each node are listed as rows. The method to create this file from the data defined in the previous sections is shown in Figure 12. Again, the generalisation to other CFD frameworks is out of the scope of this project but the modularity of these methods and implementations is such that only this interface should need to be changed.

Inlet boundary condition file writing (pseudocode)

```
x, y, z                 ← positions of nodes as flat lists for each dimension
u, v, w                 ← velocity components at each node as flat lists
velocity_magnitude      ← calculate vector norms from u, v, w of each node
nodes_size              ← number of elements in velocity_magnitude array
# SU2 requires the components of the unit vector in the direction of the velocity
flow_direction          ← u, v, w divided by velocity_magnitude for each node
# Temperature needs to be defined but will be ignored for incompressible simulations
temperature             ← array of zeroes of size nodes_size
# Collate data
text    ← "NMARK = 1
          MARKER_TAG = inlet
          NROW = [nodes_size]
          NCOL = 8
          [node data with columns: x, y, z, temperature, velocity_magnitude, flow_direction]"
Write text to a file
```

Figure 12: Pseudocode for writing the inlet boundary condition file compatible with SU2.

3.3 Method 2 – Contour Translation Method

3.3.1 Introduction

The method discussed in section 3.2 provides powerful control of the velocity profile being generated and allows the creation of novel and arbitrary swirling cases. However, difficulty arises when there is a need to study (and recreate) a distorted flow profile from empirical data. Within the literature, it is common to find these distorted flow fields reported as contour plots or streamline plots[10], [13], [14], [39], [40]; rarely is direct numerical data easily available. Therefore, in this work the capability to translate plots (from an image) to the values originally used to plot them has been explored. Contour plots have been used as they are significantly less ambiguous than streamline plots as will be apparent in the following.

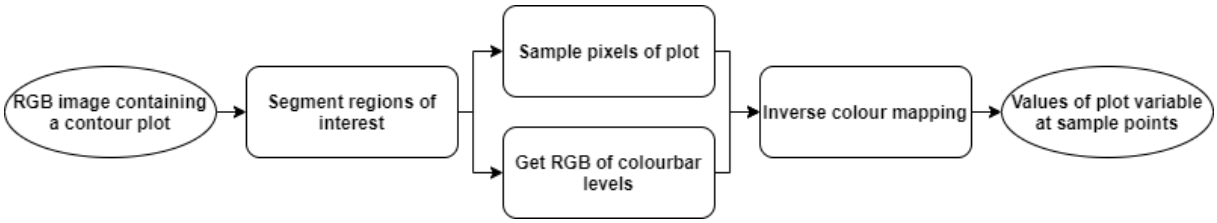

Figure 13: Top level description of the contour translation workflow.

Figure 13 illustrates the top-level steps needed to extract the numerical data which an RGB image reports. These will be explained in detail in the following but essentially the process consists of segmenting the actual contour plot from the image, sampling the colours from this, and performing inverse colour mapping[41] to estimate the values used to create the contour plot. Most of the image processing steps required for this workflow was powered by the OpenCV library.

3.3.2 Segmentation

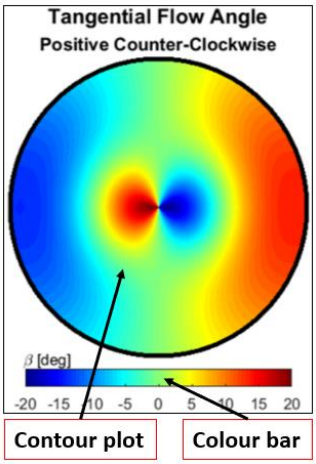

Figure 14: Image of a contour plot taken from Smith[13].

Plots found in literature have accompanying labels as shown in Figure 14. Manually preparing an image to show only the contour plot is tedious and unnecessary. In this work, it was decided that the data extraction process would begin with segmenting the input image. That is to say, the pixels comprising the regions of interest need to be identified. In this case, the actual contour plot containing the data needs to be segmented, as well as the colour bar containing the key relating the pixel colours to the values. This leaves only the pixels of interest for subsequent operations as will be seen in the following sections.

The process of rule-based segmentation is not novel and is covered in detail by the computer vision field[42]. In short, it relies on an initial edge detection followed by an application specific algorithm to attempt to define the boundaries of the region of interest. In both of the following applications, OpenCV's implementation of the Canny edge detector[43] was used, utilising an upper threshold of 200 and a lower threshold of 100. These decide which connected line of pixels qualify as an edge - as described in the documentation[44]. The values reported here were found to work well across test cases from literature which will be discussed further in section 4.1.1.

For plot segmentation, the Hough Circle Transform[45] implemented in OpenCV[46] is used. This involves finding the possible circle centres using the detected edges and then finding the best radius for each possible circle. From this set of possible circles, the largest one is assumed to be the contour plot. Once the position of the plot within the image has been obtained, the colours of the pixels comprising the plot can be extracted with their corresponding coordinates relative to the plot centre.

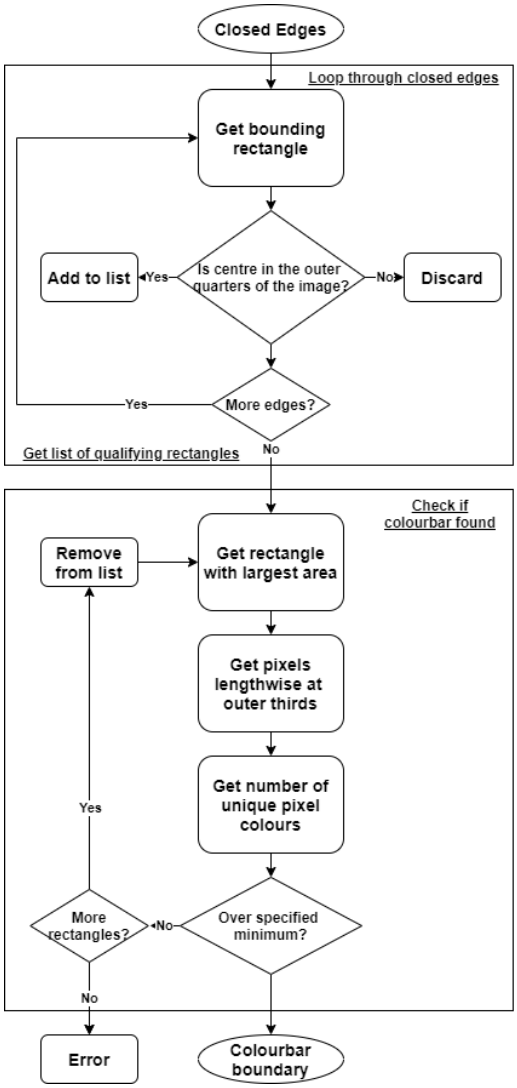

Figure 15: Colourbar segmentation algorithm.

While a colour map can be given to perform inverse colour mapping (for example 'jet' is commonly used in literature), having the capability to extract this from the image provides a useful systematic error correction tool as will be seen in section 4.2.3. The method used is illustrated in Figure 15 and was largely developed empirically - with checks to try and ensure accuracy. The process starts with getting the bounding rectangles of the closed edges (referred to as 'contours' by OpenCV[47]); the colourbar candidates are selected as those that have centres in the outer quarters of the image. These candidate boxes are sequentially investigated, from largest to smallest. A lengthwise line of pixels is extracted to see if they have a large enough variety of colours – this is attempted twice at different points along the rectangle width. If a range of colours is found, this is assumed to be the colour bar; if not, the next largest rectangle is attempted. In this work, a minimum of one hundred unique RGB values was used to qualify as the colour bar.

3.3.3 Plot sampling

It was decided that the RGB colour values of the extracted plot pixels would be sampled rather than using every pixel for inverse colour mapping. With the reasoning that this increase in computation would not come with a proportional increase in accuracy as the contour plot itself was created by interpolating discrete values. Sampling also decouples the runtime and computational cost from the input image's resolution and will potentially be more robust to noise within the image.

The translation accuracy and time requirements of different sampling approaches are discussed and compared in section 4.1.2. Three sampling distributions were considered, illustrated in Figure 16:

1. Equidistant sampling – the number of desired sampling rings is given; the sampling points are distributed such that the distance between all nodes is (almost) constant[48].
2. Constant angular resolution – the number of desired sampling rings and angular distance between points is given; taken from how the swirl descriptors are defined in AIR5686[2], [39].
3. Same points extracted from the inlet of the user's mesh.

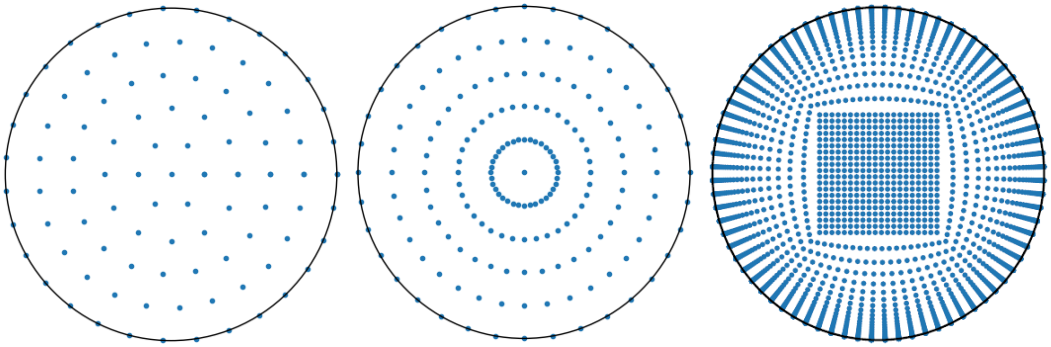

Figure 16: From left to right; equidistant sampling, constant angular resolution, from user mesh.

3.3.4 Inverse colour mapping

A process referred to as inverse colour mapping[41] can be performed on the sampled RGB values to get the numerical data represented by the contour plot. Using an ordered list of the values of the colour map (either given or extracted from the colourbar), each sampled point can be assigned a value between 0 and 1 depending on the index of the closest colour in the colour map list (treating the RGB values as 3D vectors). These normalised values are mapped to the range of values (given by the user) being represented on the contour plot, giving a representation of the numerical data used to create the plot originally. This algorithm is more tangibly illustrated as pseudocode in Figure 17. Note, no interpolation is done when mapping the pixel colour to the colour map levels, i.e., it is assumed that all colours found on the plot are in the colour map.

Inverse colour mapping (pseudocode)

```
for i = 1 to number of sample points:
        # Sample the colour of the plot at each point by using the pixel at that point
        pixel_index     ←     argmin( |pixel_coords − sample_points[ i ]| )
        colour          ←     pixel_RGB[ pixel_index ]

        # Get index of value in colour map which results in the smallest Euclidean distance
        level_index     ←     argmin( |colour_levels - colour| )
        values[ i ]     ←     level_index / size of colour_levels array


values  ←     values * (max_value - min_value) + min_value
```

Figure 17: Inverse colour mapping implementation.

3.3.5 Integration with boundary generation

As discussed previously, swirling flow is most often described by its tangential flow angle distribution usually presented as a contour plot. However, to reconstruct a 2D velocity profile, more information is needed to close the transformation equations. In this work, the radial flow angle was decided as the orthogonal variable to be used; it is often reported alongside the tangential flow angle.

With the tangential and radial flow angle fields extracted from their contour plots using the method above and given some uniform axial velocity V_z, the 2D velocity components in polar coordinates can be calculated with Equation 5 and Equation 6 below. Cartesian velocity components can then be obtained with Equation 7.

$$\dot{\boldsymbol{r}} = \boldsymbol{V_r} = \tan(\boldsymbol{\alpha}) * V_z$$

Equation 5: Radial velocity component and rate of change of radius.

$$\boldsymbol{V_\theta} = \tan(\boldsymbol{\beta}) * V_z \quad , \quad \dot{\boldsymbol{\theta}} = \frac{\boldsymbol{V_\theta}}{\boldsymbol{r}}$$

Equation 6: Tangential velocity component and rate of change of azimuthal angle.

$$\dot{\boldsymbol{x}} = \dot{\boldsymbol{r}}\cos(\boldsymbol{\theta}) - \boldsymbol{r}\dot{\boldsymbol{\theta}}\sin(\boldsymbol{\theta}) \quad , \quad \dot{\boldsymbol{y}} = \dot{\boldsymbol{r}}\sin(\boldsymbol{\theta}) + \boldsymbol{r}\dot{\boldsymbol{\theta}}\cos(\boldsymbol{\theta})$$

Equation 7: Coordinate system conversion from polar to cartesian for velocity components.

$\boldsymbol{\beta}, \boldsymbol{\alpha}, \boldsymbol{V_\theta}$, and $\boldsymbol{V_r}$ are arrays containing the polar variables described in Figure 1 for each point. $\boldsymbol{r}$ and $\boldsymbol{\theta}$ are arrays containing the polar coordinates of each point and their time derivatives $\dot{\boldsymbol{r}}$ and $\dot{\boldsymbol{\theta}}$. $\dot{\boldsymbol{x}}$ and $\dot{\boldsymbol{y}}$ are arrays containing the 2D cartesian velocity components of each point, equivalent to u and v from Equation 2 and Equation 3.

Note that if sampling methods 1 or 2 from section 3.3.3 are used, interpolation is necessary to define the flow field variables with respect to the inlet nodes of the input mesh. Within the accompanying Python tool, interpolation is done to the flow angles before calculating the polar velocity profile with Equation 5 and Equation 6. In this work, simple linear interpolation was used; specifically, the *griddata* function from the Scipy library. Cubic interpolation was attempted but did not give a significant increase in accuracy. It was decided that because of the computationally efficient methods used, it was reasonable to instead increase the number of sampling points if more accuracy is needed.

3.4 Summary of process

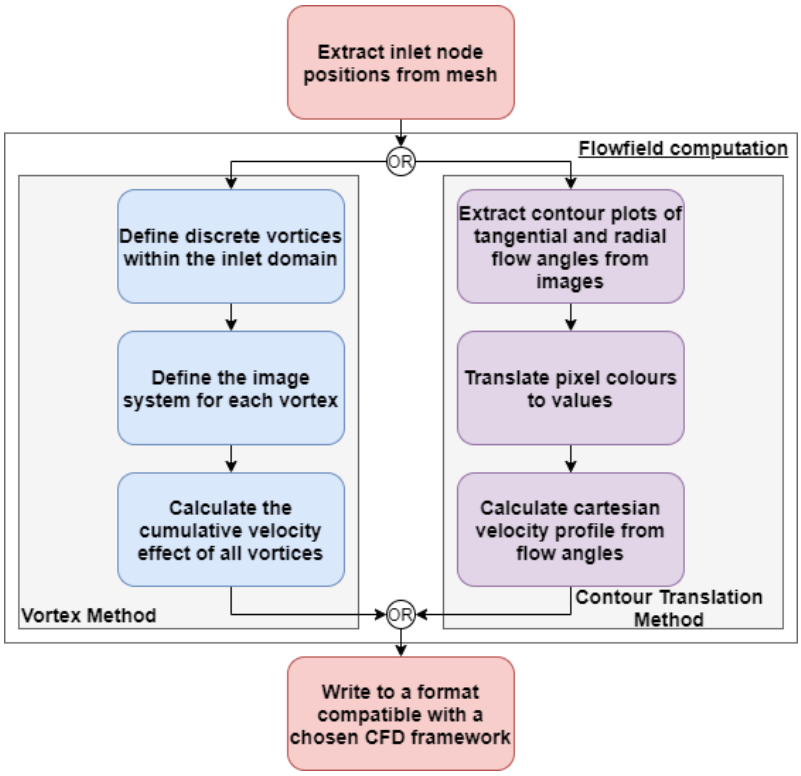

Figure 18: Flowchart of the overall boundary condition generation workflow.

The opportunity is taken in this section to restate how the components of the proposed method meet the requirements discussed in section 2.4.2. The overall workflow begins with the extraction of the inlet nodes from the user's mesh; aiding the process of ensuring that the output is useable within the chosen CFD framework. To define the velocity profile for the inlet condition, either the vortex method or the contour translation method can be used. The former utilises mathematical modelling while the latter utilises empirical data, with the convergent purpose of defining a realistic swirling flow field. The process ends by writing the flow field data to a format which is useable by a CFD package – enforcing a 3D flow and facilitating the use of the swirling flow as a boundary condition. This workflow and the integration of the components are illustrated in Figure 18.

3.5 Python implementation

With the previous sections, methods have been described to allow the reader to implement a code for generating swirling inlet boundary conditions for SU2. However, the overall goal of this project is to develop a tool to aid flow analysis; therefore, an accompanying Python code with implementations of the discussed methods has been provided on GitHub[1].

The code was developed with the design philosophy of full but optional control; this was achieved by facilitating two user interface paradigms. Firstly, to simply generate a swirling boundary condition 'out-of-the-box', a command line interface is available. Together with a 'config' file, similar to how SU2 is controlled, users can use the tool without exposure to the source code.

For increased control of the process, a modular object-based approach was used in the code. Figure 19 illustrates the minimum number of calls which need to be made to the toolkit to create an inlet condition file (i.e., not including imports and variable declarations). It should be apparent that this allows for easier integration with existing Python workflows, access to intermediate data, and parametric variations (especially important for the contour translation method). The author feels this is important for an engineering user base. The full code architecture is illustrated in Appendix B.

```
# Extract nodes from mesh
nodes = pre.Input.extractMesh(meshfilename)

# Initialise FlowField class instance with nodes
flowfield = core.FlowField(nodes)

# If using vortex method
vortexDefs = core.Vortices(model, centres, strengths, core_radii)
flowfield.computeDomain(vortexDefs, axialVel)

# If using contour translation method - with default parameters
tangential = contour_translation.Contour(tangentialImage, tanColourbarRange)
radial = contour_translation.Contour(radialImage, radColourbarRange)

flowfield.reconstructDomain(tangential.getValuesAtNodes(flowfield.coords), \
                            radial.getValuesAtNodes(flowfield.coords))

# Write file
writeBC.writeSU2(flowfield, filenameBC)
```

Figure 19: Minimum code to create a boundary condition.

4 Development

4.1 Image translation

Instilling algorithms with the ability to interpret visual input just as humans do is an active area of research. In this work traditional rule-based image processing algorithms were used which require ad-hoc parameter choices to suite particular applications. This section provides reasoning for the default parameters used within this work and the accompanying tool. These were deemed acceptable for a proof-of-concept and are not claimed to be optimal. Generalisation is beyond the scope of this project, but speculations/suggestions are given in section 6.1.

4.1.1 Segmentation performance

The performance of the segmentation method with the given parameter values described in section 3.3.2 were tested across a range of contour plots of various formats taken from different papers in literature, shown in Figure 20. Contour plots created with a greyscale colour map taken from Zachos[39] were also used.

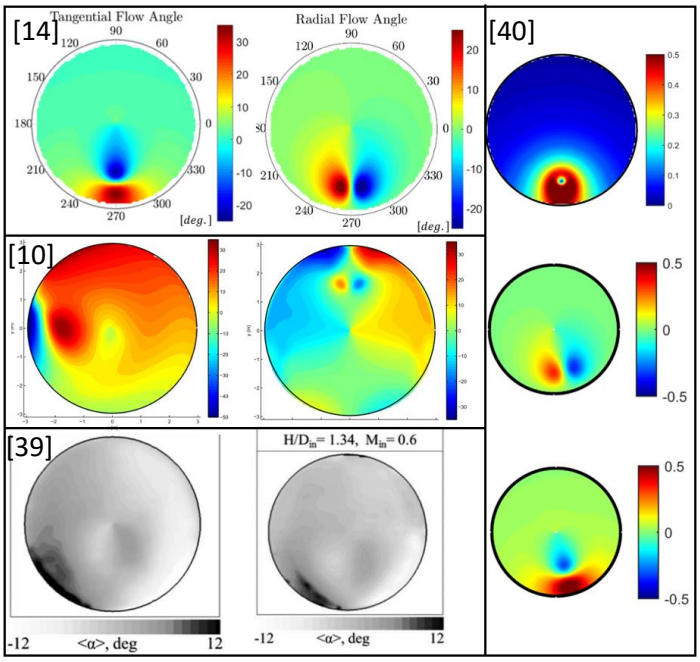

Figure 20: Collection of contour plot test cases from literature successfully segmented. Taken from: Guimaraes[14]*, Hoopes*[10]*, Zachos*[39]*, Schneckiii*[40].

The plot segmentation method uses the largest circle found by the Hough Circle Transform and tends to include the black border often added to contour plots. Therefore, the points at the edge sample colour values which are not in the colourmap thus affecting the resulting data as seen in Figure 21.

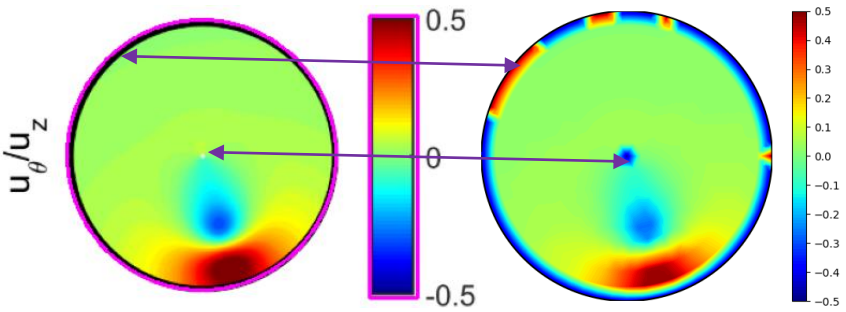

Figure 21: (Left) Initial segmentation of the plot and colourbar – input image taken from Schneckiii[13]. *Segmented boundaries shown in pink. (Right) Extracted values replotted (using sampling mode 1 with 10 radial rings).*

To correct this, a boundary shrinking method was used. Essentially, the radius of the boundary circle was reduced by one pixel if more than 90% of the outermost sample points had colour values which were out of range of the colour map. This works reasonably well as shown in Figure 22. It is noted however that a small portion of the contour plot is now missed since the bounding circle obtained from the Hough Circle transform (with the chosen parameters) is not perfectly concentric with the plot; this is made obvious when the radius is changed. It is left to the discretion of the user if this is an appropriate correction. Further, this correction naturally is not able to remove the border when the colourmap is greyscale as in the plots from Zachos[39] shown in Figure 20. Also note the blue artifact in the centre of the replotted values within Figure 21 and Figure 22; this is due to the white dot in the middle of the source image which is not included in the colour map. Corrections to the source image has not been addressed in this work.

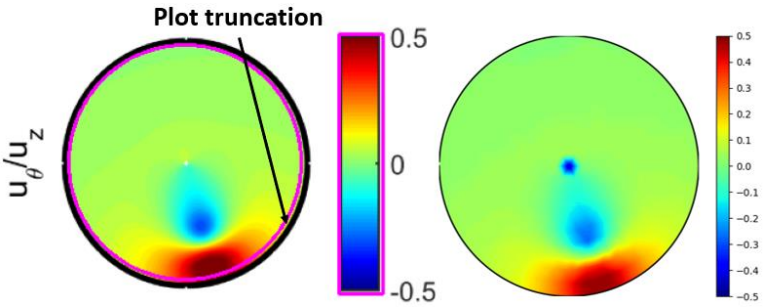

Figure 22: (Left) Corrected segmentation after plot boundary circle shrinking occurs, segmented boundaries shown in pink. (Right) Replotted values.

4.1.2 Sampling parameters & accuracy

A study was done to investigate the relative accuracy and computational cost of the different sampling distributions considered – equidistant sampling, constant angular resolution, and equivalent to the input mesh. Five hundred test cases were created with random vortex configurations using the vortex method from section 3.2. A representative selection is shown in Figure 23.

The data from the contour plots for these test cases was extracted by the contour translation method, with different sampling distributions. The extracted flow angles for each case were compared with the stored actual angles using a root mean square error normalised with the range of true values, shown in Equation 8. The time taken to translate both contour plots and reconstruct the velocity profile for each case was also measured. Results are shown in Table 1. In the second set, the cylindrical mesh described in section 3.2.2 was used - containing 3744 nodes at the inlet. Parameters were chosen for the other two sampling modes which resulted in approximately the same number of nodes. Lower resolution distributions were also tested with sampling modes 1 and 2, as well as a higher resolution set to show sampling independence.

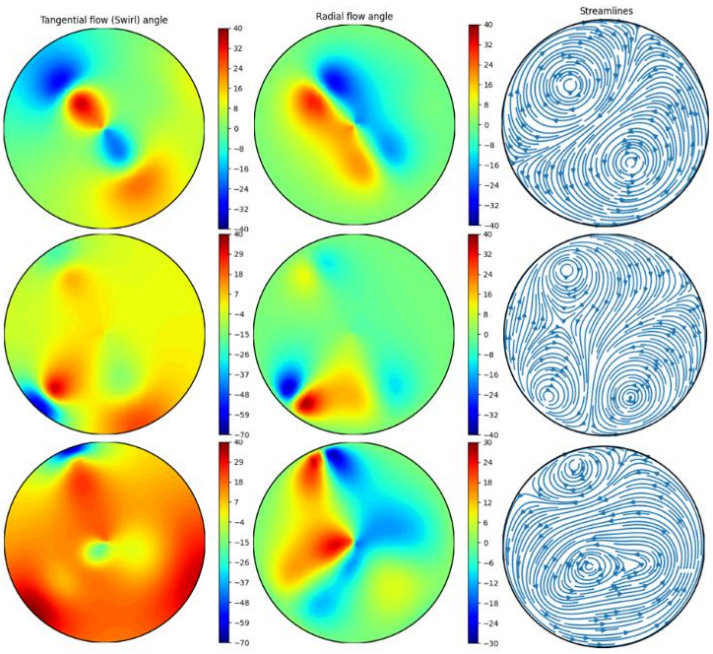

Figure 23: Randomly initialised swirl distortions. (Left) Swirl angle. (Middle) Radial flow angle. (Right) Streamlines.

$$NRMSE = \frac{1}{2}\left(\frac{\sqrt{\frac{1}{n}\sum_i^n\left(\beta_{actual,i} - \beta_{extracted,i}\right)^2}}{\beta_{actual,max} - \beta_{actual,min}} + \frac{\sqrt{\frac{1}{n}\sum_i^n\left(\alpha_{actual,i} - \alpha_{extracted,i}\right)^2}}{\alpha_{actual,max} - \alpha_{actual,min}}\right)$$

Equation 8: Normalised root mean square error of reconstructed data, averaged across tangential and radial flow angles.

No. Nodes	Sampling Parameters			NRMSE (%)		Runtime (s)	
	Mode	No. Rings	Angular Res. (°)	Mean	σ	Mean	σ
5192	1	40	-	5.02	4.32	17.387	5.483
5161	2	40	2.8	5.03	4.29	16.999	5.126
3744	3	-	-	5.19	4.47	12.192	2.805
3773	1	34	-	5.09	4.35	12.705	3.379
3741	2	34	3.3	5.09	4.34	12.547	2.516
1340	1	20	-	5.12	4.25	6	1.485
1341	2	20	5.4	5.16	4.21	5.98	1.446
356	1	10	-	5.46	4.13	3.33	0.965
361	2	10	10	5.72	4.18	3.279	0.977
99	1	5	-	5.96	3.7	2.696	0.927
101	2	5	18.4	6.33	3.92	2.64	0.922

Table 1: Results of contour translation parametric study – accuracy and runtime statistics taken over 500 cases.

Looking at the higher resolution tests in Table 1, the error is beginning to converge, and sampling independence is suggested. In the second set, sampling mode 3 is outperformed by the more uniform sampling distributions. This is because the CFD mesh being used has a high concentration of nodes near the walls to capture the boundary layer during the simulation; with regards to sampling contour plots, it is not guaranteed that this region is where large contour gradients will be present, as in the first case shown in Figure 23. For this reason, and since it is not trivial to change the 3D input mesh due to other constraints, this sampling mode is not present for the remaining sets.

As the resolution is decreased to achieve more reasonable run times, the difference in accuracy between sampling mode 1 and 2 becomes increasingly apparent. The equidistant sampling mode always outperforms the constant angular resolution mode. This is due to the increasing interpolation error with distance from the centre incurred by constant angular resolution sampling, since the arc length between points increases closer to the edge as seen in Figure 16. Therefore, when swirl is concentrated at the edges of the inlet, as in the last 2 cases shown in Figure 23, this method of sampling performs poorly. It is also computationally inefficient to increase accuracy at the edges by using a lower angular resolution value as this results in unnecessarily dense sampling rings towards the centre.

Consequently, for the remainder of this work equidistant sampling will be nominally used (mode 1), with 10 radial rings. This gives reasonably accurate results without too much computational cost, as seen in Table 1.

4.2 Fit-for-purpose testing

4.2.1 Idealised swirl cases

To show that the methods described in this work can successfully generate swirling flow, each method was used to recreate the four idealised swirl cases reported by K. Smith[13]. For the vortex method (Method 1), the ‘bulk swirl’ case was created with a ‘solid’ vortex model (i.e., one who’s vorticity goes to zero outside of its radius) with a linearly increasing swirl magnitude from centre to edge up to a maximum of 15°. The other three cases used standard Lamb-Oseen vortices. Identical parameters were used as in Smith’s paper and are reproduced in Table 2, with a uniform unitary axial velocity. Parameters are as defined in section 3.2.1 and units have been neglected – any coherent unit system can be used for the purposes of this comparison. For the contour translation method (Method 2), the images of the contour plots representing the swirl cases taken from Smith’s paper were used as input to recreate the flow fields.

	Vortex	Γ	a	x_c	y_c
Twin swirl	1	1.243	0.250	0.083	0.000
	2	-1.243	0.250	-0.083	0.000
Offset swirl 1	1	1.304	0.250	0.083	0.000
	2	-1.174	0.250	-0.083	0.000
Offset swirl 2	1	1.178	0.250	0.083	0.000
	2	-1.061	0.225	-0.083	0.000

Table 2: Parameters for each vortex in each swirl case, values taken from Smith[13]

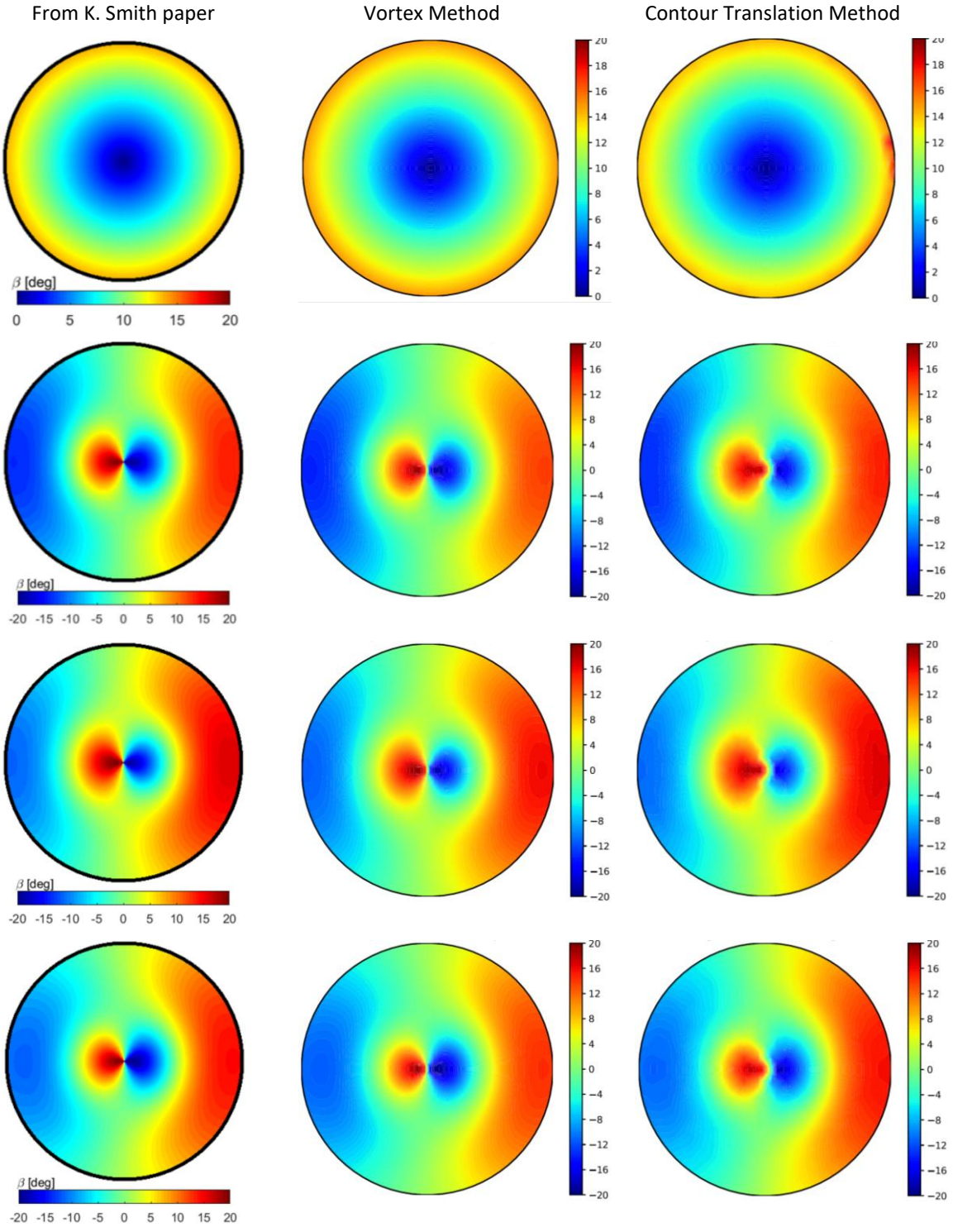

Figure 24: Idealised swirl test cases from K. Smith[13] *recreated with vortex method and contour translation method, represented by their tangential flow angle plots. From top to bottom: bulk swirl, twin swirl, offset swirl 1, offset swirl 2.*

Figure 24 reports the recreated swirl cases and shows good agreement with the literature - only the tangential flow angle plots have been reported for clarity. This shows that both methods can generate specific swirl distributions; their realism is explored in section 4.2.3. Note that the same extremes in swirl angle are not reached at the edges of the plot compared to the literature cases, especially evident in the dual vortex cases from the vortex method. The cause of this was not found and could benefit from further investigation. However, this was deemed a minor issue as it does not detract from the main aim of this work – generating swirling inlet conditions for CFD.

In addition, the dual vortex cases created by the contour translation method exhibit an artifact in the centre where one vortex seems to encroach the region of the other. This can be explained by the slight inaccuracy of the plot segmentation. Figure 25 shows that the centre of the bounding circles found by the Hough Circle method are not perfectly coincident with the actual plot. Therefore, the central sampling point will not have the correct value resulting in a shift in the interpolated values. This is confirmed by examining the radial plot segmentation and reconstruction; here the blue region appears to 'spill over' since the sampling centre is shifted towards the blue region.

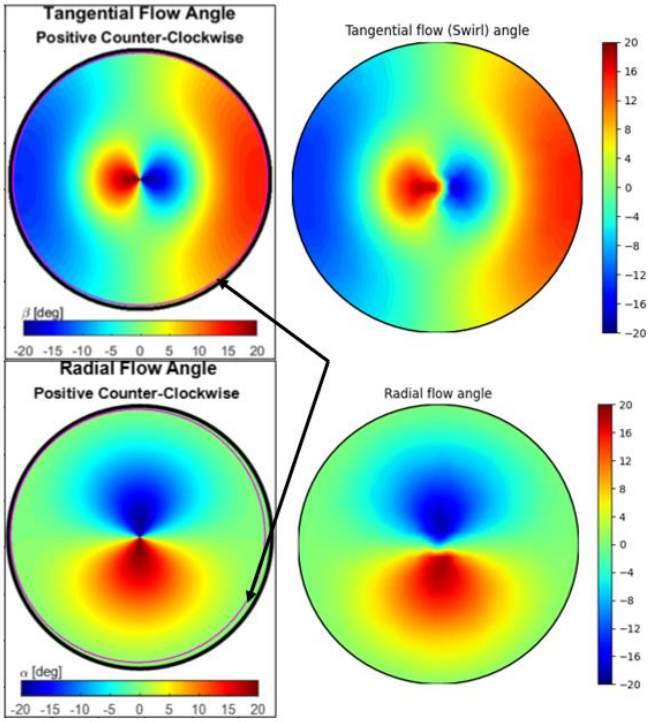

Figure 25: Non-coincident bounding circles for the plots result in inaccuracies during sampling and inverse colour mapping.

The precision of the circle segmentation for these cases can be increased by changing the threshold values reported in section 3.3.2, however this can cause the algorithm to fail entirely at finding the plot from other images within literature. A trade-off was made for generality over complete accuracy. It is also shown in section 4.2.3 that this error is sufficiently corrected in the CFD simulation.

4.2.2 Boundary flux

The effectiveness of the solid boundary modelled via the method of images can be tested by calculating the flux across the boundary. This can be done numerically for any arbitrary boundary shape as shown in Figure 26 (provided the boundary nodes are known).

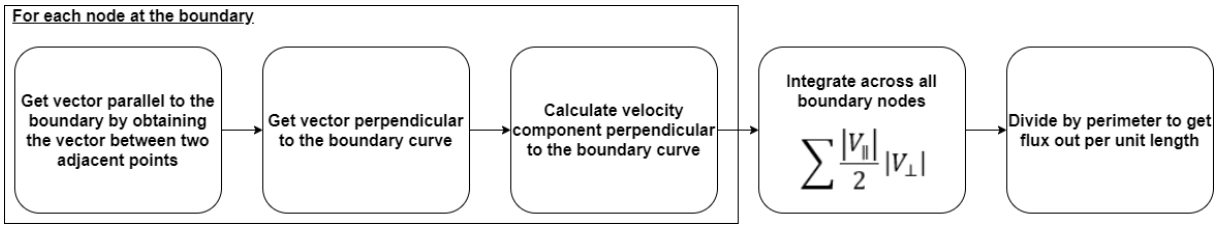

Figure 26: General method for numerically calculating flux out per unit length along a boundary curve.

One thousand test cases of swirling flows were generated with the vortex method, each created with a random number of vortices with random parameters (subject to constraints: maximum of 10 vortices per domain; maximum strength of 2 units^2/s; maximum core size of 0.5 units). A representative case is shown in Figure 27. The specific flux per unit perimeter was calculated for each case obtaining a mean of 3.61e-4 units/s with a variance of 1.57e-7. The solid boundary approximation is sufficiently accurate and is effective at allowing reasonable results to emerge from the ultimate CFD simulations as will be shown in the following section.

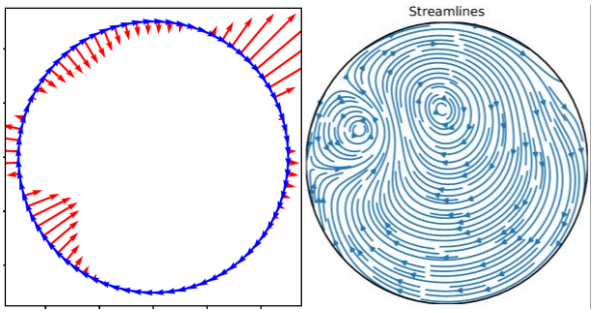

Figure 27: A randomly initialised swirl distribution. (Left) Flux across the boundary, red arrows show the velocity component perpendicular to the boundary at each node. (Right) Plot of streamlines.

4.2.3 CFD simulations in SU2

Inviscid simulations were initially done with the recreated swirl cases from section 4.2.1 to confirm that the proposed method interfaces correctly with SU2 and produces numerically stable solutions. These results have not been reported as they provide no insight which cannot also be seen from the following viscous simulation results.

The aim of these more realistic viscous tests was to investigate the development of the swirl, confirm if the boundary layer correctly develops given some buffer region, and confirm that the generated boundary conditions also work for other solver types (RANS in this case). To reflect realistic conditions, the wind tunnel experiments from Smith[13] and Hoopes[10] were emulated in the simulations with an axial velocity of 50m/s (~M 0.145) and using the 3D mesh from section 3.2.2 with a diameter of

0.1524m (~6 inches). The bias on the mesh's radial cell distribution was tuned until a maximum y^+ at the wall of 1.7 was achieved – aligned with the guideline for turbulent simulations. The parameters in Table 2 to generate the idealised swirl cases are relative to an axial velocity of 1 unit and a duct diameter of 1 unit – since the StreamVane™ part can simply be scaled up after geometry specification. To generate the same form of inlet swirl computationally for the actual duct dimensions and flow speed, these parameters need to be rescaled – vortex strength is rescaled by both the speed and diameter ratios while the remaining length variables are rescaled by the diameter ratio. The scaled parameters are shown in Table 3; the bulk swirl case's parameters do not require scaling as the maximum swirl angle is specified.

	Vortex	Γ	a	x_c	y_c
Twin swirl	1	9.472	0.0381	0.0126	0.000
	2	-9.472	0.0381	-0.0126	0.000
Offset swirl 1	1	9.936	0.0381	0.0126	0.000
	2	-8.946	0.0381	-0.0126	0.000
Offset swirl 2	1	8.976	0.0381	0.0126	0.000
	2	-8.085	0.0343	-0.0126	0.000

Table 3: Vortex parameters scaled to a 0.1524m diameter duct and 50m/s axial velocity

Results are shown in Figure 28 and it is clear that the generated boundary condition files are correctly read by SU2. With regards to the inlet profile, the given swirl cases are kept intact; this compatibility with the RANS equations speaks to the realism of the specified boundary conditions. Looking at the state of the swirl at 10D downstream, the vortex systems correctly dissipate and induce velocities on each other. In addition, Figure 28 shows that the boundary conditions generated by contour translation develop similarly to those which were mathematically defined (vortex method).

From Figure 28, the twin swirl case created by the contour translation method can be seen to rotate slightly like the offset swirl cases, whereas the same twin swirl case created by the vortex method only translates. This suggests that an error has been introduced during contour translation. What should be equal strength vortices are shown to be asymmetric; one vortex induces larger velocities on the other causing the system to rotate as it develops along the duct.

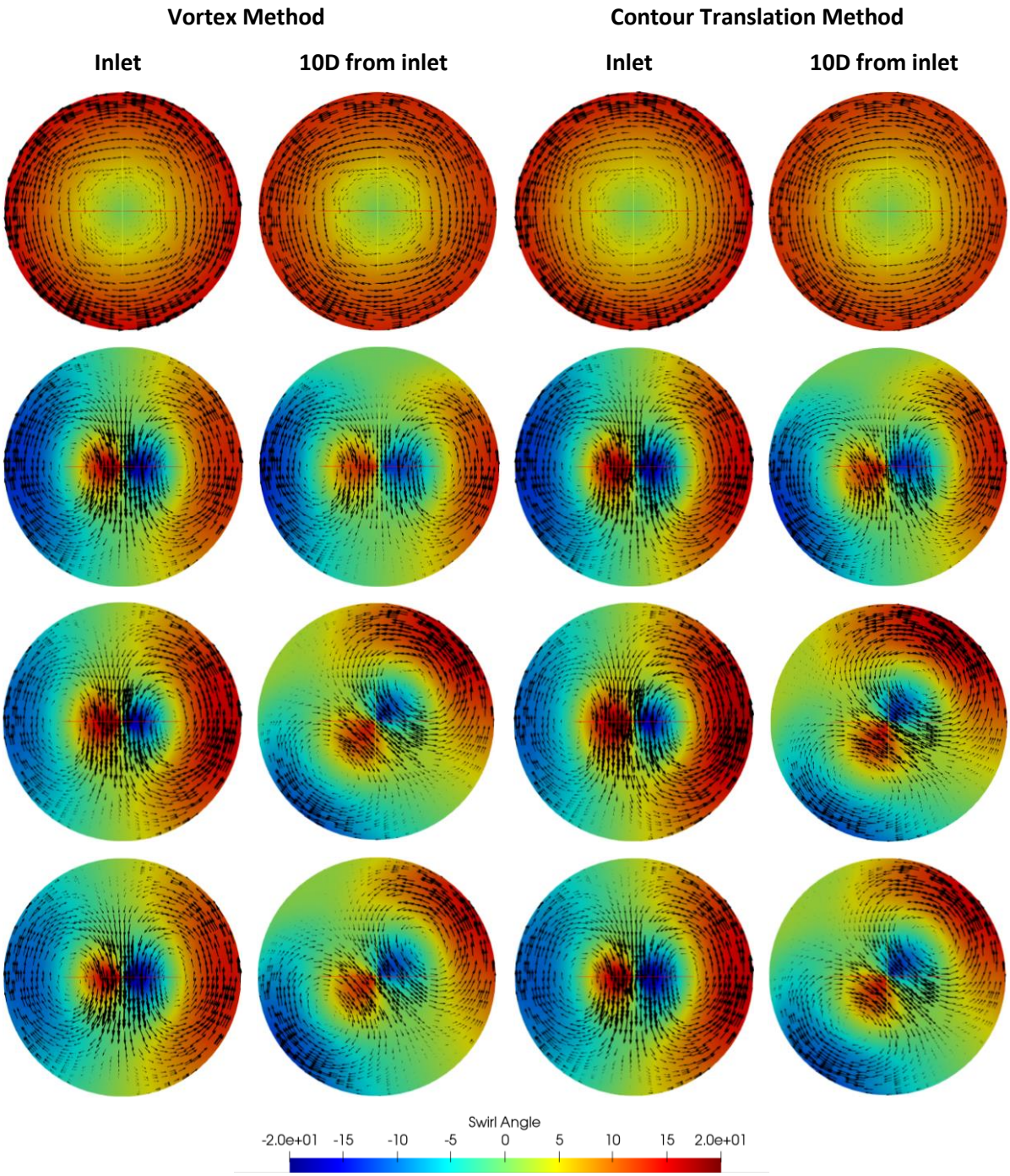

Figure 28: Slices parallel to the x-y plane (forward-looking-aft) showing the development of the swirl along the duct. Inlet conditions generated from both methods. Measurements taken at inlet and 10D downstream. From top to bottom: bulk swirl, twin swirl, offset swirl 1, offset swirl 2. In-plane velocity vectors have been overlayed on contour plots of tangential flow angle. The central axis lines have been shown to illustrate the movement of the vortex systems more clearly.

The twin swirl case was recreated again with contour translation, but this time the colourmap to be used was not given and was instead extracted from the colourbar within the input images. Results are shown in Figure 29, showing that the symmetrical vortex system now correctly maintains its orientation and simply translates downwards and dissipates downstream. This implies that a systematic error across the input images was causing the asymmetry. The extracted colour values no

longer matched those of the 'jet' colourmap as the mean was shifted – in this case towards negative infinity since the right vortex became stronger. This can also be seen (more faintly) in the other dual vortex cases. Allowing the extraction of the colourmap directly from the image offsets this error allowing the process to produce the desired symmetrical vortices. This systematic error was likely introduced because of image compression at some point between the original contour plot creation and the image being taken from a PDF for this work (using Window's Snipping Tool). Therefore, this error is likely unavoidable in practice; however, it has been shown here that it can be mitigated.

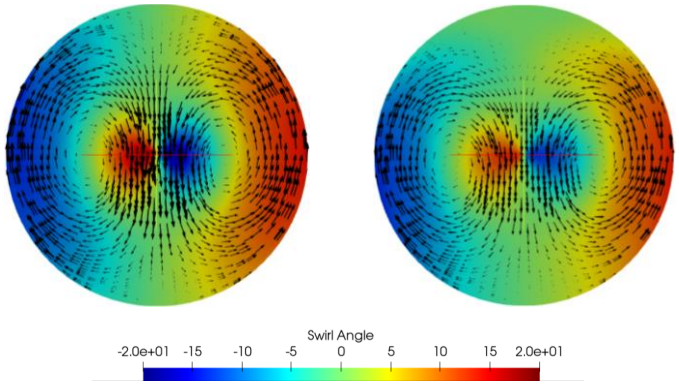

Figure 29: Slices parallel to the x-y plane (forward-looking-aft) showing the development of the twin swirl case when generated by the contour translation method while allowing the extraction of the colourmap from the input image. Measurements taken at inlet (Left) and 10D (Right).

The development of the boundary layer can be clearly observed from Figure 30 – the bulk swirl case exhibits the least interaction between the boundary layer and the vortex system. A boundary layer was not modelled within the inlet condition, opting instead to let it develop along an initial buffer region which can be seen in Figure 30.

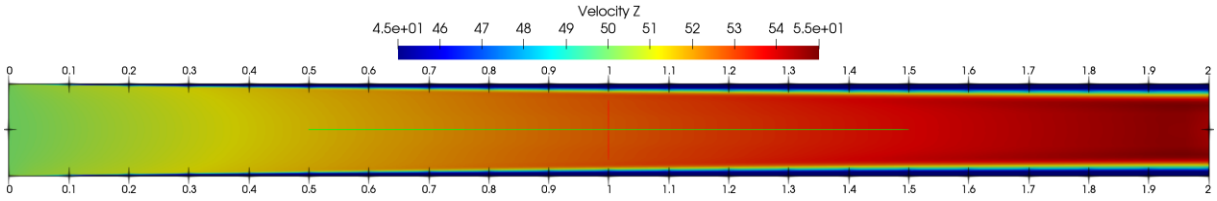

Figure 30: Slice along the duct showing axial velocity, slice plane coincident with the x-z axis. Bulk swirl inlet condition generated by the vortex method.

However, the specified swirl distribution has also itself developed and dissipated in conjunction, apparent in the results shown in Figure 28. This is in fact analogous to the problem tackled by K. Smith[13]; by the time the flow is 'useable', the specified profile has changed. Smith solves this problem by providing a method to generate an intermediate swirl profile which will develop into the desired profile at the given point downstream - since a buffer region is always necessary immediately

downstream of the StreamVane™ geometry. Here, in the numerical domain, a developed boundary layer could instead be modelled within the inlet condition to remove the need for a buffer region entirely and allow the flow to be immediately useable. This is explored further in the next section.

4.3 Boundary layer modelling

4.3.1 Method

Requiring a buffer region to allow the boundary layer to develop is a constraint; in this section, the capability of modelling the boundary layer within the inlet condition is explored. Turbulent boundary layer modelling is a large and active research area which seeks to find general methods of approximating the flow close to a wall. Here we follow the curve-fitting approach described in Rona et al.[49] to essentially obtain a function of the non-dimensional velocity u^+ in terms of the non-dimensional distance from the wall y^+. The Reichardt profile[50] has been used for its simplicity to blend the linear profile of the laminar sub-layer with the turbulent log law of the wall to model the velocity profile across the inner region of the boundary layer. Coles' approach[51] is used to complete the full boundary layer profile, including the velocity defect region.

The implementation of this boundary layer profile for a discrete mesh is shown in Figure 31. The parameter values $\kappa = 0.384, B = 4.17$ from Rona et al.[49] are used here. Note that because of the circular duct, the y for each node is calculated as its radial distance from the wall. Also, the mean velocity magnitude across the inlet has been used for u_∞.

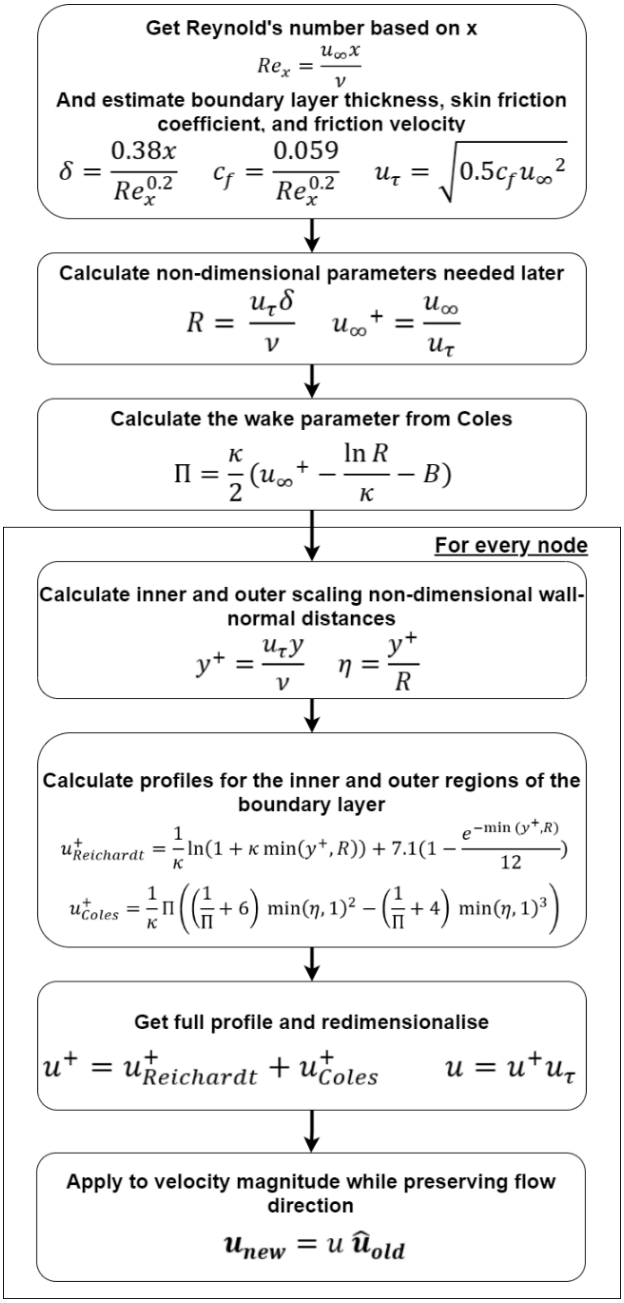

Figure 31: Method for modelling a turbulent boundary layer in a circular duct with swirling flow.

The u obtained from this process is used as the new velocity magnitude and its components are distributed into the orthogonal directions at the same proportions as the uncorrected velocity vector, so that the given form of the swirl is preserved. This is a simplification; in reality, there would be complex interactions between the swirling flow and the developing boundary layer. However, attempting to model this is out of the scope of this project; the adequacy of this approximation is suggested in the following and its usefulness is discussed.

4.3.2 Validation

Boundary conditions were generated using the bulk swirl test case – this case showed the least interaction between the boundary layer and the swirl. An uncorrected bulk swirl case was simulated (one without a boundary layer modelled) and the velocity profile was queried at three points along the duct. The equivalent boundary layer size was modelled with the method shown in Figure 31 using the appropriate *x* values and applied to a bulk swirl inlet profile (without being simulated).

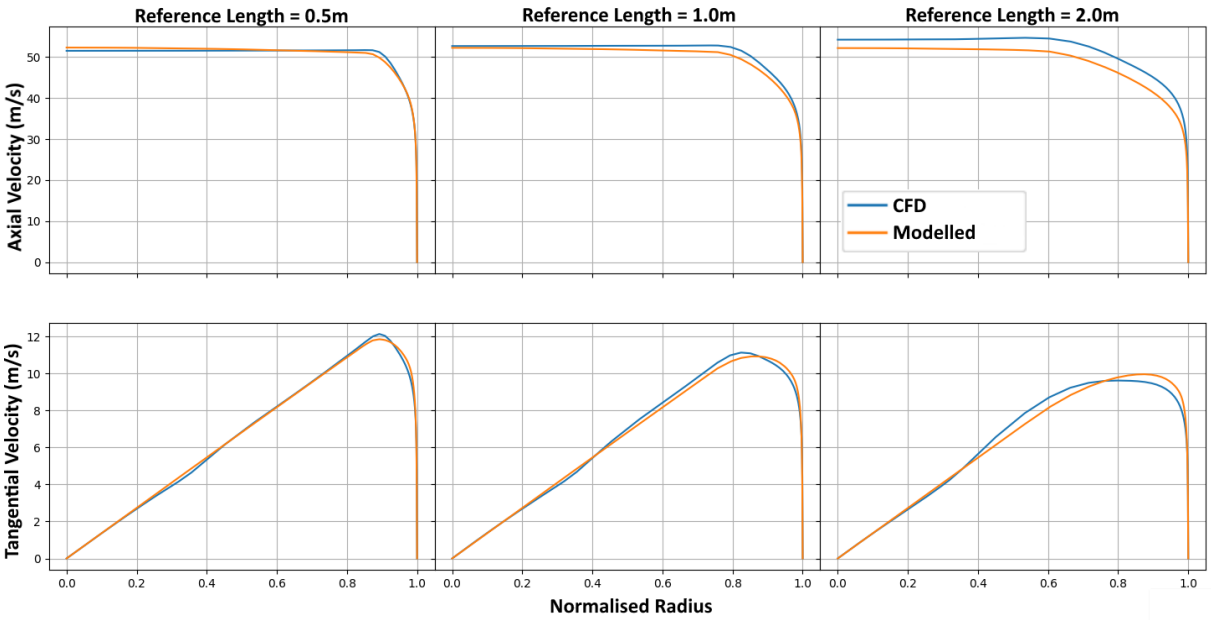

Figure 32: Tangential and axial velocity profiles along a constant angle radial line. Comparing the boundary layer profile developed within CFD measured at planes downstream with the equivalent profile modelled at the inlet.

Figure 32 shows the equivalence between the simulated and modelled velocity profiles. The disparity in the axial velocity magnitudes can be explained by the simulated flow being accelerated because of the conservation of mass constraint - as it flows through the duct the boundary layer displacement increases. This is not modelled within the equivalent inlet condition. The disparity in the tangential velocity is likely from the interaction between the swirl and the boundary layer during simulation.

Instances of inlet conditions for the four idealised swirl cases were generated with different boundary layer thicknesses (controlled by the reference length used for Re_x). Figure 33 shows that these boundary conditions develop similarly and converge along the same approximate curve at 5D downstream of the inlet. The disparities in swirl magnitude within the centre of the duct can be explained by the fact that the velocity profile correction function is being applied to all nodes across the inlet. This is a first approximation and further work is needed to define a more appropriate stopping condition or smoothing for the correction function. More comprehensive results are reported in Appendix C.

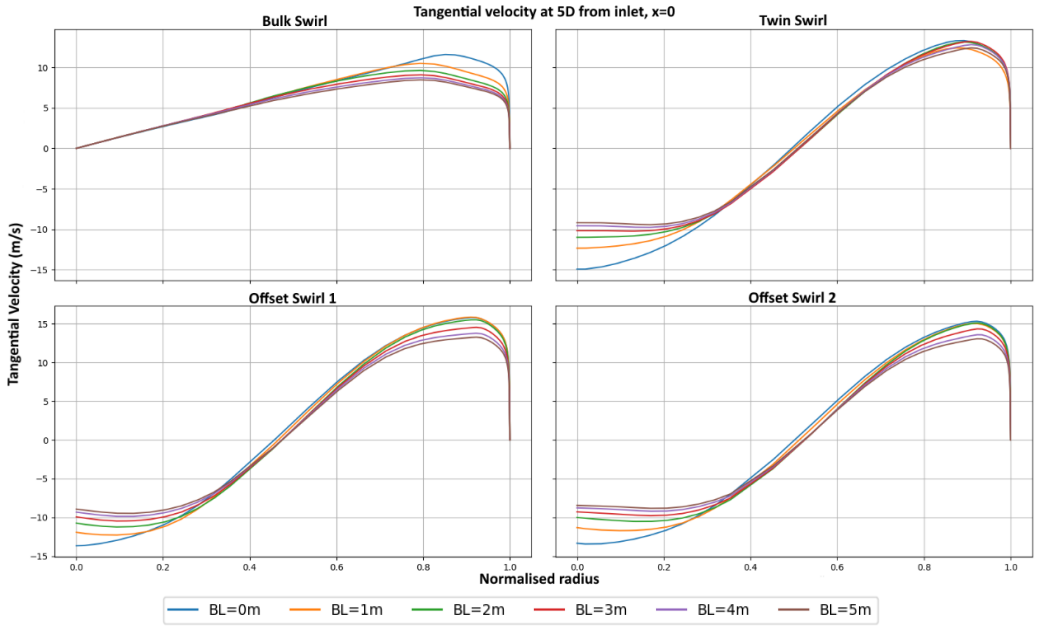

Figure 33: Tangential velocity profile of simulated swirl cases with varying degrees of boundary layer modelled. 5D from inlet

With the accuracy of the modelled boundary layer suggested above, it is reasonable to assume that these generated inlet conditions can be used almost immediately within the simulation without a buffer region to allow the inflow to stabilise. To quantify the computational savings this could potentially cause, a study was done with increasingly longer domains. The same cylindrical mesh from the previous tests was used; simply increasing the number of lengthwise cells as the duct was elongated to keep a constant mesh density. Each simulation was performed up to convergence – utilising the RMS residual of the axial velocity as the convergence criteria (up to a value of 1e-8) and a CFL number of 16.

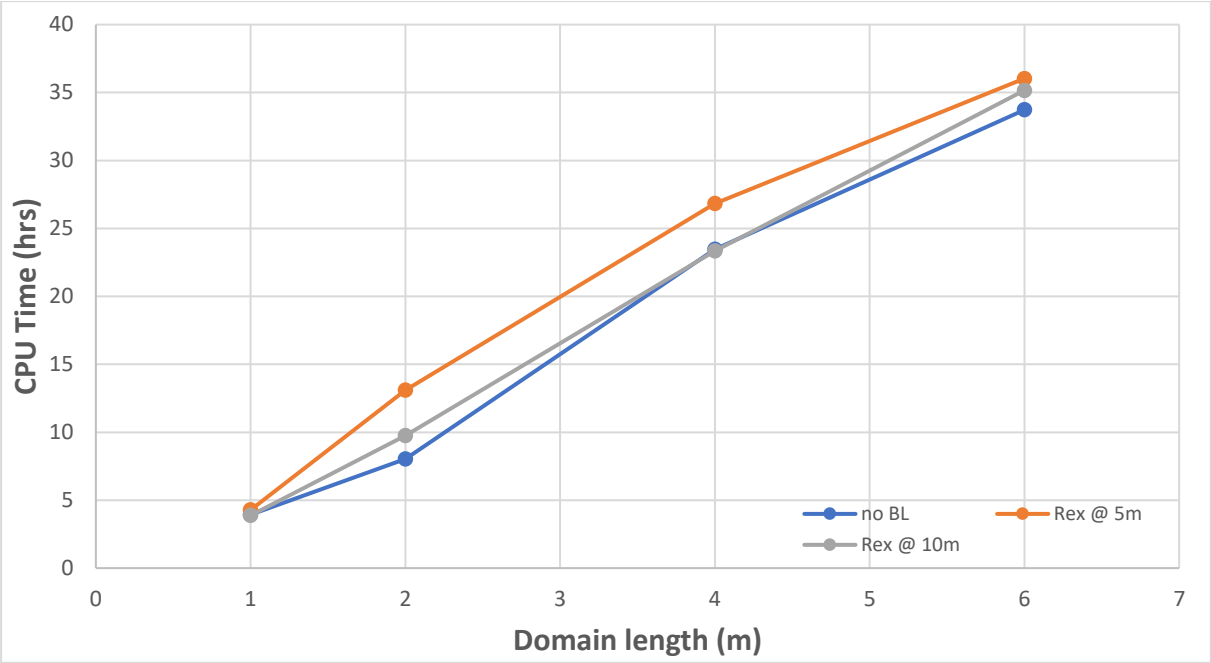

Figure 34: CPU time required to finish simulations with increasingly longer domains. The simulations were done on a cluster of Dell PowerEdge R6525s utilising 12 cores – therefore actual elapsed time is 1/12 of the CPU time shown.

Looking at the boundary layer in Figure 30, we can conservatively say that the flow is fully developed and 'useable' at around 1m. From showing the equivalence of a modelled boundary layer above, one could reasonably truncate the domain at this point, specifying an inlet condition at this plane instead of simulating the upstream flow. This would result in only 1m of duct needing to be simulated instead of 2m; Figure 34 suggests that the time required to simulate this truncated domain would be half of that required for the original domain. The approximately linear relationship shown suggests significant computational savings can be made especially for large, high fidelity meshes.

5 Sustainability

Work within the engineering field increasingly considers sustainability. As technical capability progresses, engineers are increasingly able to cause significant and lasting change. Considering this, it is important to ensure that work presented within STEM does not compromise the ability of future generations to meet their own needs[52]. The work presented in this report is relevant to sustainability in so far as it affects resource use. The methods explored has the potential to significantly decrease the computational cost of CFD simulations, potentially decreasing energy consumption and carbon footprint. In addition, the ultimate aim of this work is to aid the design of aeroengines. The potential fuel efficiencies from better designs could decrease the carbon footprint of the aerospace industry while it relies on fossil fuels for the foreseeable future.

6 Future Work

6.1 Limitations and suggestions

This work outlines the development of a tool within a finite period; naturally, there are further improvements to be made to the performance of the methods described. Apparent from section 4.1.1, improvements are needed for the accuracy of the segmentation algorithm. Here, a traditional rule-based workflow has been utilised to segment and translate contour plots. It is possible instead to utilise machine learning methods for performing segmentation; essentially optimising the parameters which were simply chosen manually. Alternatively, a GUI implementation which allows the user to manually segment the image could be more easily implemented and be more appropriate for the use cases involved (bulk/real time segmentation is not needed here).

The contour translation method proposed can only reconstruct flow fields if contour plots for both the tangential and radial flow angles are available; this is not always the case in literature. Where other variables have been provided to close the equations, the proposed contour translation method can be similarly used. However, in the case where the literature provides incomplete data, little can be done apart from trying to approximate the results. Both methods from sections 3.2 and 3.3 could be integrated within an approximation workflow whereby contour translation can be used to extract what data is available and an optimisation loop can be used with the vortex method to find the combination of vortex parameters which result in an equivalent flow field (with the loss function tied to the difference between the data extracted from the contour plot and that created by the vortex method).

With reference to the general design guidelines set out in section 2.4.2, the capability for editing boundary conditions generated by the contour translation method easily/intuitively should be present. For instance, contour plots may be scanned to reconstruct a flow field, but the user may want to change the swirl slightly for parametric studies. More work is needed to explore this capability, but a possible implementation is to again use an optimisation loop, similar to that described above, to find the vortex parameters which result in an equivalent flow field.

6.2 Going forward

It is necessary to prove the usefulness and test the performance of these methods within more representative use cases. The simulations presented in section 4.2.3 are within simplified domains only meant to showcase a proof-of-concept. The author acknowledges that more conclusive tests need to be done to confirm the realism of the generated boundary conditions. For instance, results from a pre-cursor simulation could be used to recreate the flow field with the proposed methods at some plane

downstream of vortex generating structures. The development of the flow further downstream of this plane across both cases can then be compared. Of course, an even more definitive test is to compare the development of boundary conditions recreated from empirical data with the actual results of an experiment. These tests would also further support the claim that the generated inlet conditions can be used to truncate a domain and save computation.

In addition, the methods presented here exclusively generate inlet conditions for incompressible flow. Further work is needed to generate boundary conditions for compressible flow since the thermodynamic properties of the flow also need to be defined. For this, a more analytical approach could be found; however, staying with the theme of this work, a computational sampling method like that utilised by Smith[13] can also be used. A compatible thermodynamic distribution could presumably be obtained by utilising Monte Carlo methods to repeatedly guess and correct towards some defined ideal which meets physical constraints. Alternatively, if comprehensive empirical data of compressible flow is available, intuition suggests that the contour translation method can be used to directly reconstruct the compressible flow field.

7 Conclusion

1. Method proposed for creating non-uniform inlet conditions by superimposing discrete vortices successfully recreates the idealised swirl cases described in AIR5686. Also shown to be able to create arbitrary continuous vortical flows.
2. Capability for translating contour plots from images into their underlying numerical data has been explored. A proof-of-concept implementation has been given which adequately segments the plot and colourbar from an image and accurately extracts the data via inverse colour mapping.
3. Both methods integrated into a workflow for generating inlet boundary condition files which are immediately useable within an SU2 3D simulation.
4. Capability of modelling a wall boundary layer within the inlet condition has been explored to potentially truncate a CFD domain further and save computation.
5. Realism and physical accuracy of generated boundary conditions have been suggested but not fully confirmed.
6. Python toolkit for generating swirling conditions and translating contour plots has also been developed and provided with this work.

8 References

[1] G. Vidanes, “GericoVi/virtual_swirlgenerator: Python tool for creating arbitrary swirling inlet boundary conditions useable within a CFD framework.,” 2021. https://github.com/GericoVi/virtual_swirlgenerator (accessed Jan. 18, 2021).

[2] SAE, “AIR5686: A Methodology for Assessing Inlet Swirl Distortion,” 2017. Accessed: Jan. 06, 2021. [Online]. Available: https://www.sae.org/standards/content/air5686/.

[3] F. Aulehla, “Intake Swirl - A Major Disturbance Parameter In Engine / Intake Compatibility,” 1982.

[4] K. A. Geiselhart, D. L. Daggett, R. Kawai, and D. Friedman, “Blended wing body systems studies: boundary layer ingestion inlets with active flow control,” *Nasa/Cr*, vol. 212670, no. December, 2003.

[5] S. M. Flesberg, “Mixing enhancement in a scramjet combustor using fuel jet injection swirl,” 2015.

[6] H. Ogawa, “Mixing characteristics of inclined fuel injection via various geometries for upstream-fuel-injected scramjets,” in *Journal of Propulsion and Power*, 2015, vol. 31, no. 6, pp. 1551–1566, doi: 10.2514/1.B35581.

[7] H. Ogawa, “Effects of injection angle and pressure on mixing performance of fuel injection via various geometries for upstream-fuel-injected scramjets,” *Acta Astronautica*, vol. 128, pp. 485–498, Nov. 2016, doi: 10.1016/j.actaastro.2016.08.008.

[8] Y. Jiang *et al.*, “Effect of free stream angle on mixing performance of hydrogen multi-jets in supersonic combustion chamber,” *International Journal of Hydrogen Energy*, vol. 45, no. 46, pp. 25426–25437, Sep. 2020, doi: 10.1016/j.ijhydene.2020.06.055.

[9] G. Saravanan and C. Suresh, “Numerical simulation of mixing enhancement in scramjet using micro vortex generator,” in *Procedia Engineering*, Jan. 2012, vol. 38, pp. 3969–3976, doi: 10.1016/j.proeng.2012.06.454.

[10] K. M. Hoopes, W. F. O’brien, K. T. Lowe, and C. B. Williams, “A New Method for Generating Swirl Inlet Distortion for Jet Engine Research,” Virginia Tech, Jun. 2013. Accessed: Oct. 02, 2020. [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/49545.

[11] G. R. Tabor and M. H. Baba-Ahmadi, “Inlet conditions for large eddy simulation: A review,” *Computers and Fluids*, vol. 39, no. 4. Pergamon, pp. 553–567, Apr. 01, 2010, doi: 10.1016/j.compfluid.2009.10.007.

[12] Y. L. Wu, E. Y. K. Ng, and K. Wong, “Numerical study of the swirl flow in F-5E intake with subsonic speeds,” *Mathematical and Computer Modelling*, vol. 48, no. 3–4, pp. 447–467, Aug. 2008, doi: 10.1016/j.mcm.2007.10.009.

[13] K. N. Smith, W. F. O'brien, C. K. T. Lowe, and A. L. Wicks, "New Methodology for the Estimation of StreamVane™ Design Flow Profiles," Virginia Tech, Feb. 2017. Accessed: Oct. 02, 2020. [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/82039.

[14] T. Guimaraes Bucalo, K. T. Lowe, and W. F. O'Brien, "An overview of recent results using the StreamVane method for generating tailored swirl distortion in jet engine research," in *54th AIAA Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, 2016.

[15] C. D. Pierce and P. Moin, "Method for generating equilibrium swirling inflow conditions," *AIAA Journal*, vol. 36, no. 7, pp. 1325–1327, May 1998, doi: 10.2514/2.518.

[16] X. Jiang, G. A. Siamas, and L. C. Wrobel, "Analytical equilibrium swirling inflow conditions for computational fluid dynamics," *AIAA Journal*, vol. 46, no. 4, pp. 1015–1018, Apr. 2008, doi: 10.2514/1.29439.

[17] V. Venkatakrishnan, "On the role and challenges of CFD in the aerospace industry Spalart Boeing Commercial Airplanes Seattle USA," vol. 120, pp. 1223–209, 2016, doi: 10.1017/aer.2015.10.

[18] T. D. Economon, F. Palacios, S. R. Copeland, T. W. Lukaczyk, and J. J. Alonso, "SU2: An open-source suite for multiphysics simulation and design," *AIAA Journal*, vol. 54, no. 3, pp. 828–846, Dec. 2016, doi: 10.2514/1.J053813.

[19] M.-F. Auclair-Fortier, D. Ziou, and M. Allili, "Global computational algebraic topology approach for diffusion," in *Computational Imaging II*, May 2004, vol. 5299, p. 357, doi: 10.1117/12.525975.

[20] "Quick overview of different 'Boundary Conditions' in CFD." https://www.manchestercfd.co.uk/post/quick-overview-of-different-boundary-conditions-in-cfd (accessed Mar. 04, 2021).

[21] K. J. Millman and M. Aivazis, "Python for scientists and engineers," *Computing in Science and Engineering*, vol. 13, no. 2. pp. 9–12, Mar. 2011, doi: 10.1109/MCSE.2011.36.

[22] T. Oliphant, "NumPy." https://numpy.org/ (accessed Jan. 18, 2021).

[23] "PythonSpeed - Python Wiki." https://wiki.python.org/moin/PythonSpeed (accessed Mar. 04, 2021).

[24] "SciPy.org — SciPy.org." https://scipy.org/index.html (accessed Mar. 04, 2021).

[25] "SciPy library — SciPy.org." https://scipy.org/scipylib/ (accessed Mar. 04, 2021).

[26] Intel, "OpenCV - OpenCV." https://opencv.org/ (accessed Mar. 04, 2021).

[27] M. Mohamad, "A Review on OpenCV," no. August, 2015, doi: 10.13140/RG.2.1.2269.8721.

[28] G. Bradski and A. Kaehler, "OpenCV," *Dr. Dobb's journal of software tools*, vol. 3, 2000.

[29] D. A. Forsyth and J. Ponce, *Computer vision: a modern approach*. Upper Saddle River, N.J: Prentice Hall, 2003.

[30] L. Maddalena, F. Vergine, and M. Crisanti, “Vortex dynamics studies in supersonic flow: Merging of co-rotating streamwise vortices,” *PHYSICS OF FLUIDS*, vol. 26, p. 46101, 2014, doi: 10.1063/1.4871022.

[31] “Laminar Backward-facing Step.” https://su2code.github.io/tutorials/Inc_Laminar_Step/ (accessed Mar. 04, 2021).

[32] D. J. Acheson, *Elementary fluid dynamics*. Oxford: Clarendon, 1990.

[33] L. K. Brandt, “Numerical and analytical studies of two-dimensional vortex pair dynamics in unstratified and stratified environments,” 2009.

[34] L. M. Milne-Thomson, *Theoretical Hydrodynamics*. Dover, 1996.

[35] N. Drakos, “Method of Images, Problem solution,” *MIT*. http://web.mit.edu/fluids-modules/www/potential_flows/LecturesHTML/lec1011/lec10home2/node4.html (accessed Jan. 11, 2021).

[36] C. Geuzaine and J.-F. Remacle, “Gmsh: a three-dimensional finite element mesh generator with built-in pre-and post-processing facilities,” 2009. Accessed: Mar. 05, 2021. [Online].

[37] “Mesh File.” https://su2code.github.io/docs_v7/Mesh-File/#su2-native-format-su2 (accessed Jan. 10, 2021).

[38] A. Sijal, “Understanding Y+ for CFD Simulations.” https://www.linkedin.com/pulse/understanding-y-cfd-simulation-sijal-ahmed/ (accessed Mar. 15, 2021).

[39] P. K. Zachos, D. G. MacManus, D. G. Prieto, and N. Chiereghin, “Flow distortion measurements in convoluted aeroengine intakes,” *AIAA Journal*, vol. 54, no. 9, pp. 2819–2832, 2016, doi: 10.2514/1.J054904.

[40] W. C. Schneckiii, T. Guimarães, D. J. Frohnapfel, K. T. Lowe, W. F. O’Brien, and W. W. Copenhaver, “Swirling flow evolution Part 2: Streamflow 2D+t model validated with stereo PIV measurements,” 2017, doi: 10.2514/6.2017-1622.

[41] G. Ward, *Graphics Gems II*. 1991.

[42] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. O’Reilly Media, Inc, 2008.

[43] J. Canny, “A Computational Approach to Edge Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986, doi: 10.1109/TPAMI.1986.4767851.

[44] “Canny Edge Detection — OpenCV-Python documentation.” https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html (accessed Mar. 16, 2021).

[45] R. O. Duda and P. E. Hart, “Use of the Hough Transformation to Detect Lines and Curves in Pictures,” *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972, doi: 10.1145/361237.361242.

[46] “OpenCV: Hough Circle Transform.” https://docs.opencv.org/3.4/d4/d70/tutorial_hough_circle.html (accessed Mar. 05, 2021).

[47] “OpenCV: Contour Features.” https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html (accessed Jan. 17, 2021).

[48] P. Holoborodko, “Generating Equidistant Points on Unit Disk,” 2015. http://www.holoborodko.com/pavel/2015/07/23/generating-equidistant-points-on-unit-disk/ (accessed Mar. 05, 2021).

[49] A. Rona, M. Monti, and C. Airiau, “On the generation of the mean velocity profile for turbulent boundary layers with pressure gradient under equilibrium conditions,” Accessed: Feb. 03, 2021. [Online]. Available: http://oatao.univ-toulouse.fr/.

[50] H. Reichardt, “Vollständige Darstellung der turbulenten Geschwindigkeitsverteilung in glatten Leitungen,” *ZAMM - Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 31, no. 7, pp. 208–219, Jan. 1951, doi: 10.1002/zamm.19510310704.

[51] D. Coles, “The law of the wake in the turbulent boundary layer,” *Journal of Fluid Mechanics*, vol. 1, no. 2, pp. 191–226, 1956, doi: 10.1017/S0022112056000135.

[52] W. Commission on Environment, “Report of the World Commission on Environment and Development: Our Common Future Towards Sustainable Development 2. Part II. Common Challenges Population and Human Resources 4.” Accessed: Mar. 08, 2021. [Online].

9 Appendix A – Solid boundary modelling need

SU2 simulation results showing the effect of adding solid boundary modelling to the given inlet velocity profile. The effect on the in-plane velocities is minor as seen in Figure 36 and the swirl angle plots in Figure 37 and Figure 38. However if the solid boundary's effect is not modelled, there is a significant effect on the resulting axial velocity after it is altered by the CFD solver, as seen in Figure 35 and the axial velocity plots in Figure 37 and Figure 38. The truncation of the infinite velocity field leaves a net downward flow direction which manifests itself clearly in Figure 37.

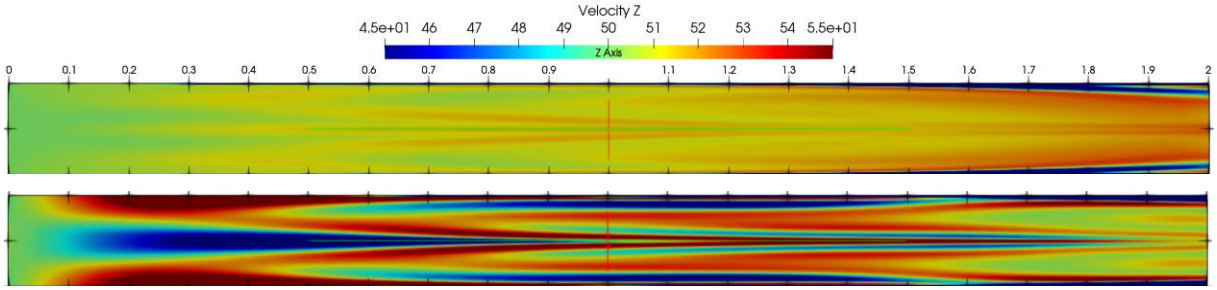

Figure 35: Slice along the duct, coincident with the x-z axis showing axial velocity. (Top) With a solid wall modelled. (Bottom) Without a solid wall modelled.

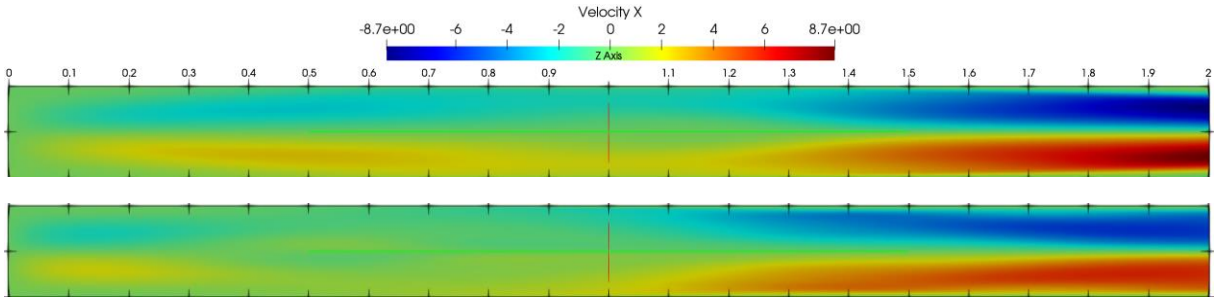

Figure 36: Slice along the duct, coincident with the x-z axis showing x-velocity. (Top) With a solid wall modelled. (Bottom) Without a solid wall modelled.

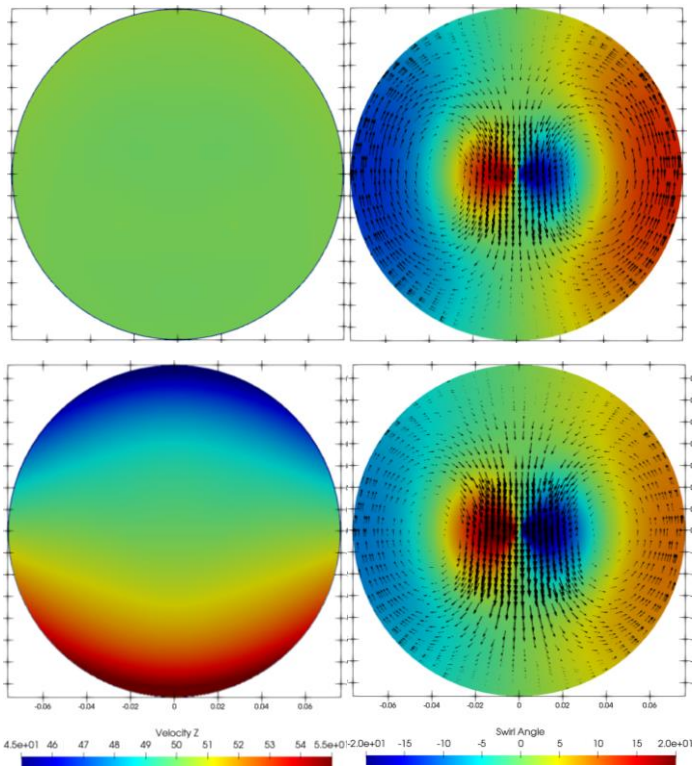

Figure 37: Duct slices parallel to the x-y axis at the inlet plane showing axial velocity, and swirl angle with in-plane velocity vectors. (Top) With a solid wall modelled. (Bottom) Without a solid wall modelled.

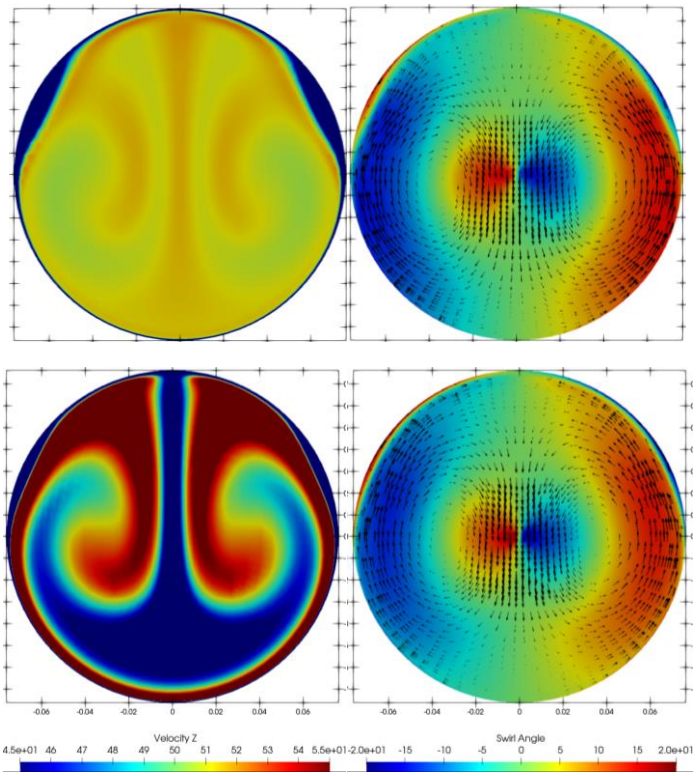

Figure 38: Duct slices parallel to the x-y axis at a plane 5D from the inlet showing axial velocity, and swirl angle with in-plane velocity vectors. (Top) With a solid wall modelled. (Bottom) Without a solid wall modelled

10 Appendix B – Code architecture

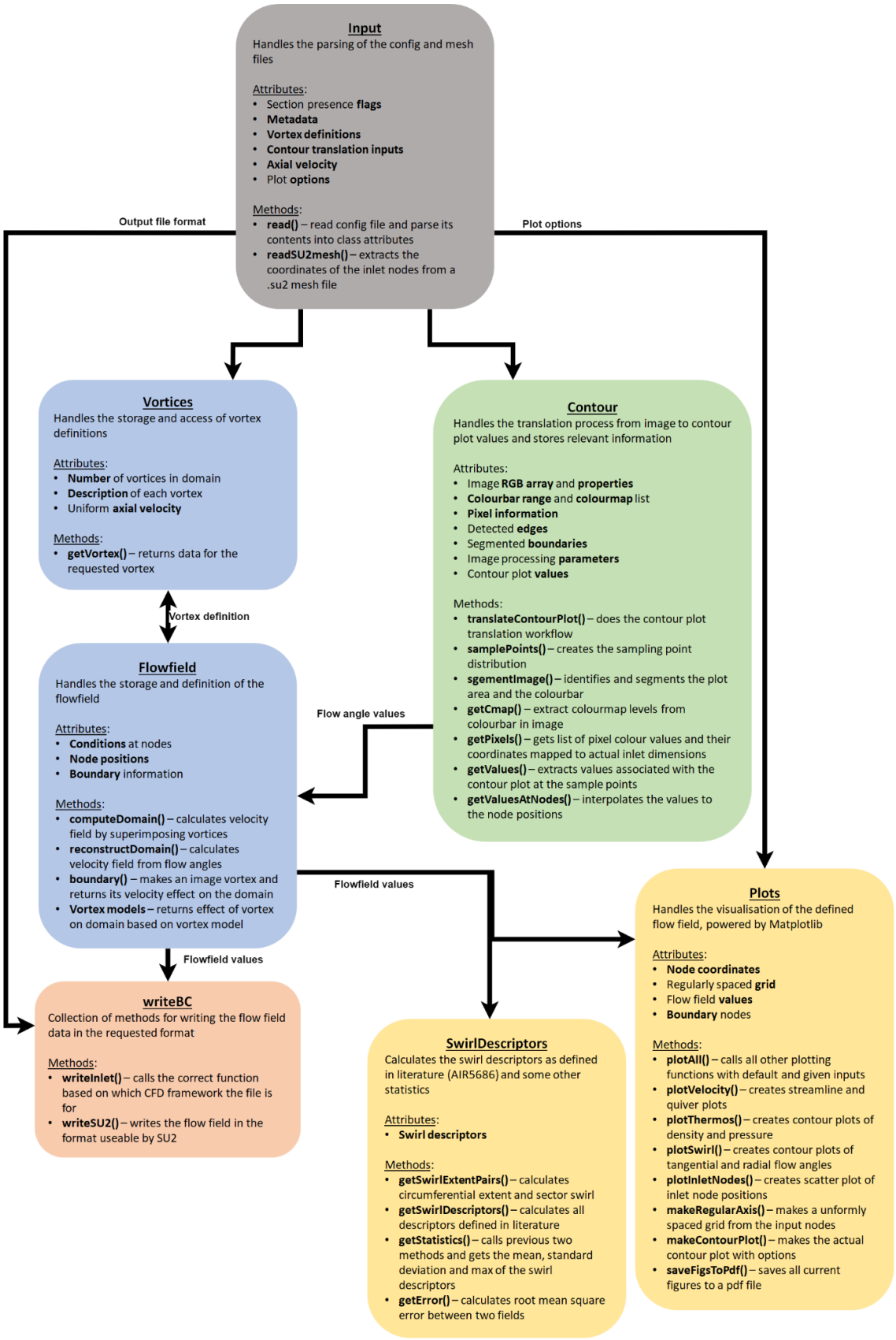

Figure 39: General class-based code architecture and interfaces

11 Appendix C – Modelled boundary layer development

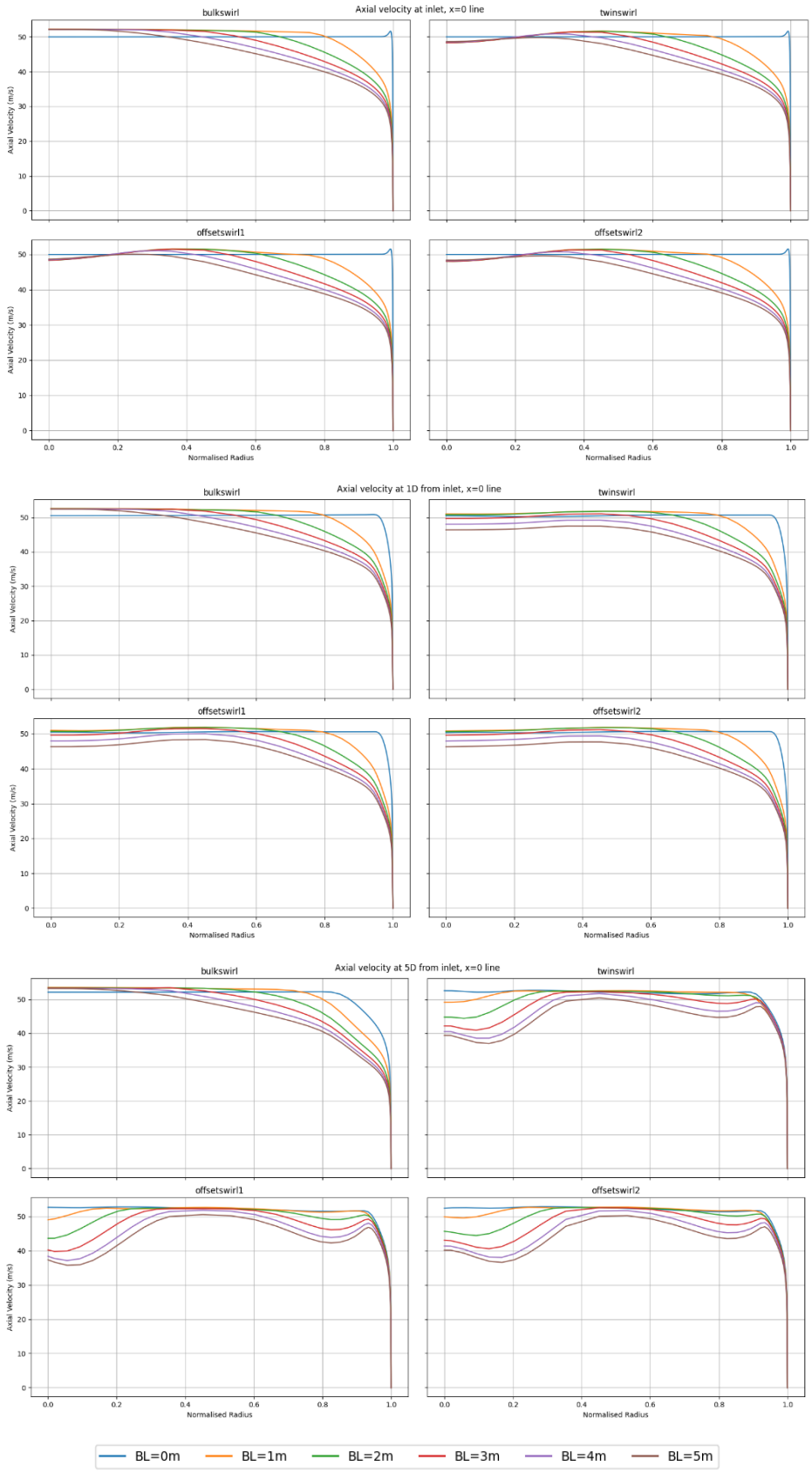

Figure 40: Axial velocity profiles along a radial line for the idealised swirl cases with different extents of boundary layer modelled within the inlet condition. Measurements taken at inlet, 1D and 5D.

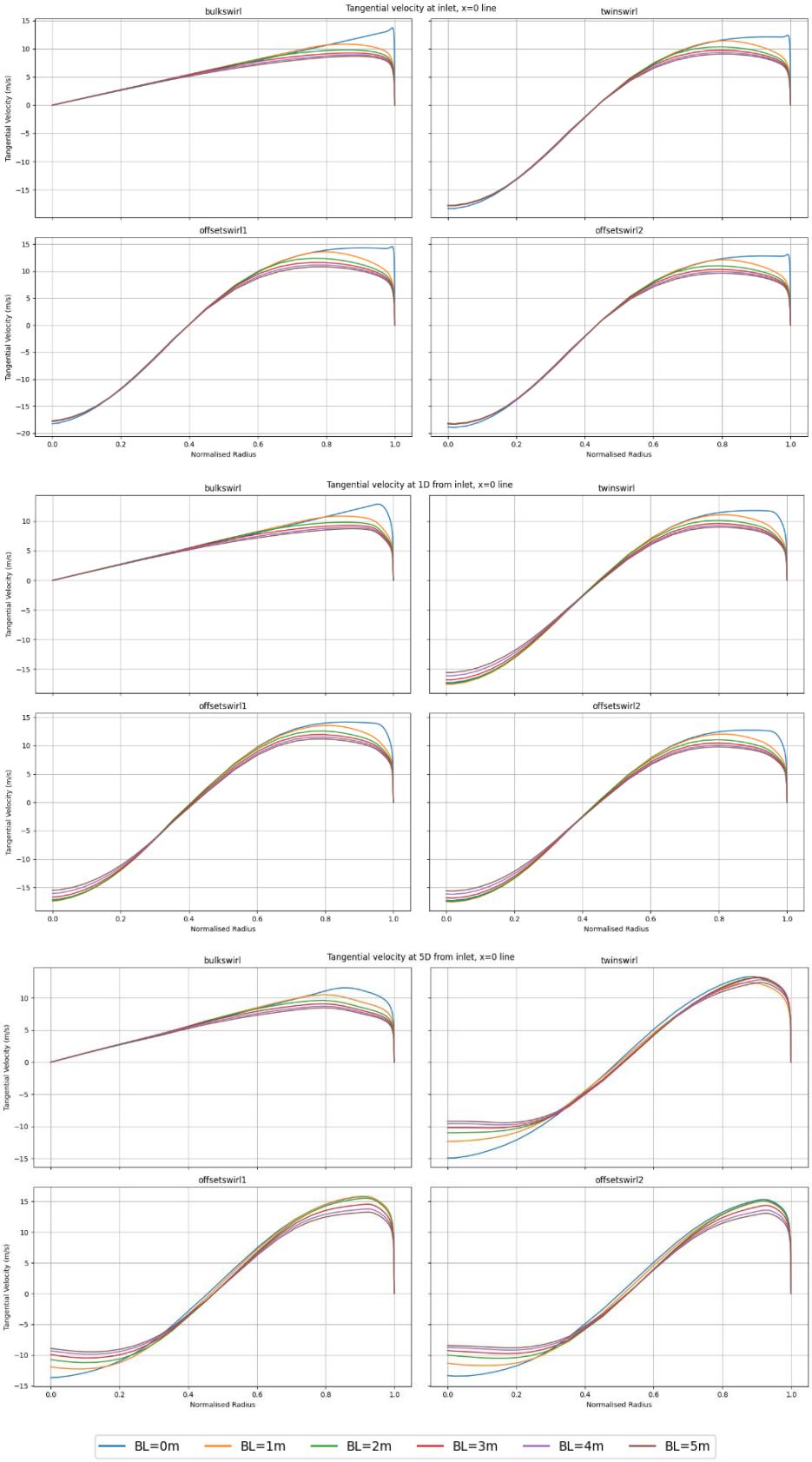

Figure 41: Tangential velocity profiles along a radial line for the idealised swirl cases with different extents of boundary layer modelled within the inlet condition. Measurements taken at inlet, 1D and