ABSTRACT

Title of Document: EFFECT OF KINEMATIC VARIATIONS ON VISCOUS PUMPING BY A ROBOTIC GILL PLATE ARRAY

Mary Alexandra Larson,
Master of Science, 2011

Directed By: Associate Professor, Kenneth Kiger , Mechanical Engineering

As the Reynolds number of a system decreases, traditional pumping techniques become less effective. In nature, oscillating appendage systems exhibit distinct patterns of movement based on their Reynolds number. Studies of pumping by mayfly nymph gill arrays have shown different kinematics over Reynolds numbers from 2 to 22. To understand why and how this pumping mechanism might be optimized, a robotic oscillating plate array was constructed allowing stroke and pitch variation as well as phase lag variation between adjacent gills. Stereoscopic PIV was used to obtain three dimensional velocity data, allowing computation of the net pumping rate and flow induced dissipation for five cases, focusing on the role of the gill plate interactions and their dependence on the phase lag. The results indicate that mayfly gills most likely use a phase lag of 90° because it produces the highest net mass flow rate and has the highest specific flux efficiency.

EFFECT OF KINEMATIC VARIATIONS ON VISCOUS PUMPING BY A ROBOTIC GILL PLATE ARRAY

By

Mary Alexandra Larson

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2011

Advisory Committee:
Associate Professor Kenneth Kiger, Chair
Associate Professor Elias Balaras
Professor James H. Duncan

© Copyright by
Mary Alexandra Larson
2011

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

As devices continue to decrease in size, traditional pumping techniques become less effective. A variety of micropumps, which take advantage of viscous effects rather than inertial, have been studied for applications such as microelectronic cooling, environmental monitoring, medicinal delivery, micro total analysis systems (μTAS) and individual biological and chemical assays. Many of these pumps exhibit effective fluid transport for particular conditions, but are limited to fluids with specific chemical properties, finite volumes of fluid or small flow rate ranges. For a micropump to be effective for an application such as an environmental monitoring chemical sensor, it has to be able to function with a variety of fluids at flow rates that are effective for sampling the properties of the fluid in question. A micropump that is effective with this level of versatility has not yet emerged.

For a pump to operate efficiently with a variety of flow rates and fluids with different viscosities, it has to be effective for different ranges of Reynolds numbers. For smaller length scales, this may require that the pump function in the regime of Reynolds numbers from about 1-100. This regime is less well understood and the interaction of the viscous and inertial mechanisms changes the behavior of the fluid. A pump that is effective in flows completely dominated by either inertial or viscous effects may no longer be effective in this transitional regime.

One approach to addressing this transitional Reynolds number regime is to observe how animals in this realm accommodate for this change in fluid interactions.

The current study focuses on a specific animal that functions in this range of Reynolds numbers, the mayfly nymph *Centroptilum triangulife.* This animal uses seven pairs of external gill plates to pump water around its body in order to acquire an acceptable concentration of oxygen necessary for breathing. Previous studies on live mayflies have shown that this animal undergoes kinematic and geometric changes as its Reynolds number increases from 2 to 22 (Sensenig et al, 2009). Although the flow generated by the animal could be documented in these studies, the limited range of behavior of the animal prevented a detailed study of why and how such a pumping mechanism might be optimized. In order to determine the influence of individual kinematic and geometric characteristics on the pumping efficiency, further experiments that allow for isolated variation of different properties are necessary.

The study described in this thesis focuses on the effects of two of these characteristics: the phase lag in between the gills and the amplitudes of the stroke and pitch angles of the gills. In order to study the individual influence of these characteristics, it was necessary to develop a device that could replicate the nominal kinematics of typical mayfly gills as well as extend the range beyond what was exhibited in nature. The full details of this robotic device and its capabilities are detailed in Chapter 3 of this document.

Fluid measurements using stereoscopic PIV were taken for five different test cases at multiple phase angles throughout an oscillation cycle. This allowed for the three components of the unsteady velocity to be reconstructed for a three dimensional volume surrounding the gills. This data allows direct computation of the flow induced dissipation and the net flux in and out of the volume over a cycle. These

parameters provide insight on the effect of the phase lag and stroke and pitch amplitudes on pumping efficiency and why certain kinematics might be used by a live mayfly. These parameters as well as qualitative observations about the unsteady flow fields and the mean flow fields for each test case also provide insight on the hydrodynamic mechanisms that generate this net flow. The full details of the results and conclusions drawn from this data can be found in Chapters 4 and 5.

Chapter 2

Background and Literature Review

2.1 Micropumps

An increasing number of engineering devices are being designed to incorporate the pumping of fluids in millimeter and sub-millimeter scales. This has motivated continued development of micropumps for applications including microelectronic cooling, environmental monitoring, medicinal dispersal, micro total analysis systems (μTAS) and individual biological and chemical assays. While numerous pumps have been developed for specific applications, the emergence of a pump that is effective with a wide variety of fluids, fluid volumes and flow rates has not emerged.

Current types of micropumps can be divided into two main categories, displacement pumps and dynamic pumps. Displacement pumps use one or more moving boundaries and the confining geometry of the device to displace a fixed quantity of fluid. Dynamic pumps continuously add energy to the working fluid in order to increase its pressure and to induce flow without a positive seal that prevents backflow (Laser & Santiago, 2004). For most applications, either type of pump could be used, but one type of pump may prove to be more effective than the other. Dynamic pumps are advantageous for both electronic cooling and environmental monitoring chemical sensors, as they typically provide a high flowrate but only a relatively small pressure increase. Electronic devices require a continuous source of cool fluid at high flow rates. Micropumps are required for this application because

the area to perform convective cooling continues to be reduced for new devices (Jiang et al., 2002). For environmental monitoring, there is a desire to develop autonomous chemical sensors with fast reaction times that may be used in the detection of potential biological and chemical attacks (Vitko, et al., 2004). For these to be effective, they need to be exposed to a large volume of fluid, and therefore require a high flow rate, continuous pump. For medical devices and μTAS, success has been found with both displacement and dynamic pumps. In the medical field, efforts to develop small pumping mechanisms for drug delivery, such as insulin delivery for diabetic patients, provided early motivation for micro pumps. This type of application generally requires a lower flow rate and needs a carefully metered amount of fluid. Success has been found in this area with both displacement and dynamic pumps, although displacement pumps are the most prevalent. For medical cases, patient pain is a strong consideration, providing motivation to use smaller devices for implants, and smaller samples of fluid, such as blood, for testing (LaVan et al., 2003). The development of μTAS is motivated by the ability to provide a larger number of measurements from a single sample, such that small-volume, high-value samples can be effectively analyzed. Because these systems generally use a finite amount of fluid, success has been shown using displacement pumps, but currently the most popular devices use dynamic micropumps (Dittrich, Tachikawa, & Manz, 2006). All of these applications are in quickly growing fields, and the emergence of a highly versatile, time efficient pumping mechanism is required for further advancement.

As the size of pumps decreases, the same pumping dynamics cannot be used because of the change in the dominance of inertial and viscous forces of the fluid

being pumped. A traditional centrifugal pump shows a rapid decrease in efficiency as it approaches smaller Reynolds number regimes where viscous forces become more influential (Laser & Santiago, 2004). Because of this, devices have been specifically developed to take advantage of viscous effects, rather than rely on inertia. A variety of devices that are currently in use or being developed are based on concepts ranging from reciprocating surfaces to electromagnetic fields. Each of these concepts has its advantages and disadvantages and has been met with different levels of success.

One of the most popular types of micropumps is a displacement pump described as a reciprocating diaphragm or membrane pump. The basic design of these pumps consists of a two-step cycle including an intake step, where the fluid is drawn into the chamber and an output step, where increased pressure in the chamber forces the fluid through an outlet. Valves at the inlet and outlet are used to control which direction the fluid flows. The change in pressure is provided by a flexible membrane on one side of the chamber that controls the volume of the fluid chamber. The nature of this pump causes the fluid to be delivered in discrete volumes, the size of which can be controlled by the volume of the chamber and the stroke length (van Lintel, van de Pol, & Bouwstra, 1988).

Different versions of these devices use a variety of actuation methods, the most prevalent including piezoelectric, thermopnuematic and electrostatic actuation. Piezoelectric actuation relies on an electric field being applied to a piezoelectric material, causing it to push against the diaphragm of the fluid chamber as it expands and contracts. This was one of the earliest actuation methods used for micropumps and remains a popular method because of its quick mechanical response and

relatively high stroke volume (Thomas & Bessman, 1975). This method originally required a comparably high actuation voltage, but after optimization many now function in the range of 100 V. The mounting procedure for a piezoelectric transducer is also considered a drawback of this method, and in some cases limits miniaturization (Woias, 2005). Thermopneumatic actuation, first demonstrated by Van de Pol et al., uses an air-filled chamber with a resistive heater. When the air is heated, the chamber expands and pushes the diaphragm (Van de Pol et al., 1990). This is a low voltage method, but requires a comparably long time during the cooling stage, which limits the frequency of the pumping. It also may have the side effect of heating the fluid being transported. For electrostatic actuation, voltage is applied to electrodes that cause an electrostatic attraction of the pump diaphragm. When this attraction discharges, the diaphragm returns to its original position (Bourouina, Bosseboeuf, & Grandchamp, 1997). This method provides high actuation frequencies, on the order of kHz, and has relatively low power consumption. A disadvantage is the method only provides a small actuation stroke, which limits the size of the discrete volumes being pumped (Woias, 2005). Each of these methods currently has its own limitations, including high voltage requirements, constraints on further miniaturization, and constraints on flow rate control. Solutions and minimization of these limitations are currently being researched.

A second type of displacement micropump has a moving boundary exert pressure on the fluid, but with an aperiodic motion. These micro pumps have shown commercial success, but are usually only effective with finite volumes of liquid. An example of such a device is an insulin delivery system marketed by Medtronic, which

is controlled by a reservoir of compressed gas. The device is implanted into the body and the valves control the release of insulin into the diabetic's intraperitoneal cavity. This device is effective for this application, but depends on the pressure reservoir being recharged when the device is refilled with insulin (Medtronic MiniMed Inc.) Pneumatic aperiodic displacement pumps are generally low power and can work with a variety of fluids, but they require close looped systems and therefore a finite volume of liquid (Laser & Santiago, 2004). Another genre of pumps are dynamic pumps in which an electromagnetic field interacts with the working fluid, including electrohydrodynamic (EHD) pumps, electroosmotic (EO) pumps and magnetohydrodynamic (MHD) pumps. These pumps all have the ability to deliver a continuous flow and have the advantage of no moving parts. Electrohydrodynamic pumps are based on the ions in dielectric fluids interactions with electrostatic forces. Operation of these micropumps relies on the electric properties of the working fluid, specifically the permittivity and the conductivity. It requires the existence of space charge, and the pumping scheme associated with this is based on the type of generation used to create the space charge. An example of this is the induction method used in pumps developed by Bart et al. and Richter et al. which involves generation and movement of induced charge at fluid-fluid or fluid-solid interfaces (Bart et al., 1990; Richter et al., 1990). This requirement limits the type of fluid that can be pumped to non-conducting and non-ionic fluids (Woias, 2005).

There has been a large focus on the development of electroosmotic pumps because of their compatibility with the μTAS or "lab on a chip" concept (Woias, 2005). These pumps take advantage of the electric double layer (EDL) which

consists of the surface charge that develops when a liquid contacts a solid, and the bulk liquid counter-ions that shield this surface charge. It is possible to use this EDL by applying an electric field parallel to the wall that causes some of the counter-ions to be set in motion (Laser & Santiago, 2004). These devices have been designed to produce high pressure and can be scaled to different sizes. They are also bidirectional, because when the polarity of the electric field is reversed, the flow is also reversed (Wang et al, 2009). A disadvantage is their dependence on the ion density and pH of the working fluid. The zeta potential, which affects the pressure and flow rate performance of the pump, is dependent on the pH of the fluid (Laser & Santiago, 2004). This limits the device to the fluids that it was optimized for, or a compromised efficiency for alternate fluids.

For a magnetohydrodynamic pump, a magnetic field is applied to a solution that contains current carrying ions. This causes a Lorentz force, which induces a flow. This flow is dependent on the direction of the magnetic field vector, which can be reversed to change the direction of the flow. In a typical MHD pump, the flow rate is related to the fourth power of the hydraulic diameter of the channel. This relationship makes miniaturizing this type of pump challenging without significantly limiting the flow rate. The performance of these pumps is also limited by the magnetic flux density of the fluid (Jang & Lee, 2000). An important consideration to note for EHD, EOP and Magnetohydrodynamics pumps is that because they induce an electric field, they all have the potential of heating the fluid (Laser & Santiago, 2004).

Additional concepts for micropumps that are being explored are ones that replicate nature. These use ideas observed from animals that pump fluid for the purpose of feeding, locomotion or ventilation in a low Reynolds number spectrum. An example of this is the idea of using cilia, which are hair like protrusions from cells, to pump fluid through microchannels. In nature, cilia cover protozoans and epithelial cells which allow them to move or transport loads on their surfaces. Cilia are also located in biological windpipes where they serve the purpose of transporting material (Timonen, 2010). Recent studies have shown that cilia are effective at pumping fluid through confined spaces and that the magnitude of the flow rate as well as the direction can be controlled. The magnitude of the flow can be altered by changing the amplitude of the bending stroke of the cilia as well as the speed of the cilia movement. The direction can be changed by altering the sperm number, which is a dimensionless number that relates the importance of the bending rigidity of the cilium and the viscous force (Alexeev et al, 2008). Other micropumping methods are also being explored including the use of rotating gears and acoustic driving mechanisms (Laser & Santiago, 2004; Woias, 2005). These are currently in the experimental phase and each has shown obstacles in their development that are yet to be overcome.

A variety of micropumps have been developed that are effective for specific uses, but the emergence of a completely versatile pump has not occurred. In order for a micropump to be versatile it must be able to adapt to different fluids, including variations in pH, viscosity, viscoelesticity and temperature, be effective with a wide range of volumes and be able to produce a variety of flow rates, while requiring a low

amount of power (Laser & Santiago, 2004). An example of this necessity is a chemical sensor that must be able to work with fluids that have different properties. The volume in question depends on the availability and dispensability of the fluid being tested and the concentration of target molecules in the fluid. The flow rate that should be used is dependent on multiple factors, and must account for the balance between convection and diffusion of the molecules in the fluid. As a sensor collects target molecules, a depletion zone forms around the sensing surface. This gradient causes more molecules to diffuse towards the sensor, allowing the sensor to eventually collect enough molecules for an effective sample. Since diffusion alone can take a large amount of time, convection can be introduced to force more molecules to approach the detecting surface. For the greatest efficiency, convection must be limited such that the depletion zone still spans the width of the channel to ensure that molecules from the entire sample still move towards the sensor. Finding this balance, accounting for the rate that the target molecules actually attach to the sensor, called the reaction limit, and still getting the sensor to take an accurate sample in a reasonable amount of time has proven to be a great challenge (Squires, 2008). For sensors to continue to be effective as they decrease in size, the development of a micropump that can be controlled to accommodate for these different factors is crucial.

2.2 Pumping in Nature

One challenge that emerges for a pump that must function with different flow rates and types of fluids is the ability for the pump to be effective in different Reynolds number regimes. Micropumps are generally designed to function in the

Stokesian (low Reynolds number) regime where viscous forces are completely dominant. Most macrosized pumps are designed to function in the Eulerian or high Reynolds number regime, where the flow can be approximated as inviscid and inertia is the dominant mechanism controlling the flow. In order for a pump to function efficiently with a variety of volumes of liquid, different flow rates, and different fluids viscosities, the pump must be able to deal with various Reynolds numbers and the different fluid behaviors associated with them. This may require that the pump be able to function in the regime of Reynolds numbers from about 1-100. This regime is less well documented and the interaction of the viscous and inertial mechanisms changes the behavior of the fluid so that a pump that may be efficient in the Stokesian regime is no longer ideal in this transitional regime.

One approach to addressing this transitional Reynolds number regime is to observe nature to see how animals that function in this realm accommodate for this change in fluid interactions. By observing the characteristic methods of motion used in the different regimes of Reynolds numbers and how animals transition from one regime to another, it is possible to determine when different methods of mass transport should be used and what the most effective means of pumping is in these different regimes.

A prevalent method of moving fluid among animals is the use of appendages. Many studies have been done on the use of appendages on live animals, and the typical motion used in different Reynolds number regimes has been classified into two major categories: flapping and rowing. Flapping motion is defined as motion where thrust is generated perpendicular to the stroke motion, such as the reciprocating

movement of a wing which generates lift. Motion is considered rowing when the thrust generated is in the same direction as the stroke motion. This thrust is created by having asymmetric stroke kinematics and geometry, generally with distinct power and recovery strokes. In a comparative study, animals were classified by their Reynolds number and by whether they use rowing or flapping movement. It was observed that animals that are nearly neutrally buoyant that only have to use their thrust generation for locomotion only use flapping motion at Re>100. Below this point, rowing motion is used. Animals that are negatively buoyant continue to use flapping motion below a Re of 100, because while the efficiency for the locomotion thrust might be lower, this movement has a better combination of thrust and upward forces (Walker, 2002). The necessity of this distinction in movement from rowing to flapping motion in this realm of Reynolds numbers has been reinforced by computational and analytical studies, the results of which show a viscous limitation on thrust for flapping motion and changes in propulsive efficiency based on movement type and Reynolds number (Walker, 2002; Childress & Dudley, 2004).

The use of appendages in oscillating arrays is an additional technique used for fluid pumping by animals in both the low and high Reynolds number ranges. For low Reynolds numbers, cilia have been observed beating slightly out of phase when positioned in large groups. The phase lag between adjacent cilia creates a motion referred to as a metachronal wave. The cilia discussed previously were able to induce flow using an asymmetric movement and geometry (Alexeev et al, 2008). A computational study by Hussong et al. (2010) shows that the use of a metachronal wave is an additional way to induce flow. This study used an array of symmetric

beating cilia that only exhibited asymmetry through the use of a metachronal wave. The results show that this motion is an effective transport method for flow in a channel for Reynolds numbers ranging from 1 to 2000. The size of the phase lag in between adjacent appendages is also a factor that can change the effectiveness of the coordinated motion. A study on the locomotion of krill, which function at a Re> 100, compared how using synchronized pleopod motion or metachronal pleopod motion affected the efficiency of propulsion. It was found that the metachronal wave is the most efficient and also produces the highest body velocities for the krill (Alben et al., 2010). The metachronal wave is exhibited on animals that function both in low and high Reynolds number ranges and may be an important contributor to effective pumping in both cases.

2.3 Previous Work on Mayfly Nymphs

The rowing and flapping classifications described above provide good general guidelines for what type of motion would be the most efficient in these Reynolds number ranges, but more details are necessary when designing and optimizing an engineering device. While the general motion of the animals can be classified, the details of each animal are different. The animals documented have different appendage shapes, stroke amplitudes, asymmetric rowing patterns, and use different angles of attack. As seen with the animals that also use their thrust for the purpose of lift, these traits may also perform multiple roles (Walker, 2002). This means that the trait might be the most efficient for the animal and its purpose, but not necessarily for the engineering device replicating its motion.

When modeling an engineering device after something observed in nature, general trends and common factors provide a good starting point for initial design guidance. In order to develop the specifics of the device, such as the shape or inclination angle of an oscillating plate, a more comprehensive understanding of the effects of the different geometric and kinematic features is necessary. An effective way to obtain a better understanding of these details is to select an animal that performs a similar function to the device being designed, such as pumping, and determine the influence that the different kinematic and geometric properties have on its efficiency.

In this study, the mayfly nymph was identified as an animal that transitions in the comparatively understudied Reynolds number range from 1 to100. A mayfly nymph has seven pairs of external gill plates located on the lateral dorsal region of its abdomen. For many species, these plates are actively controlled to circulate water when the oxygen concentration surrounding the animal is too low. This is done when the mayfly nymph is not moving, so the gill movement is only associated with ventilation. This animal is also significant because it uses different movement at different Reynolds numbers, transitioning from rowing at its lowest Reynolds number to flapping at its highest Reynolds number. Unlike other animals observed in this regime, it uses the same set of appendages as its Reynolds number increases (Sensenig et al., 2009). The animal's goal of fluid movement for circulation is analogous to the goals of a micropump, specifically for an application such as a chemical sensor where un-sampled fluid needs to be moved closer to the sensor. The kinematics, gill interaction and physical features of the mayfly gill movement may be

able to provide a base model and further insight into the optimal method of circulating flow in this intermediate Reynolds number range.

In previous studies, the kinematics of the gills and the flow field around them were quantified while the animal was operating in a frequency Reynolds number, $Re_f = L_g^2 f/\nu$ range from 2 to 22. This Reynolds number uses the gill length (L_g), the oscillation frequency (f), and the viscosity of the working fluid (ν). The key parameters of the gills that are described in this study are the individual movement of a single gill relative to the stroke plane, seen in , described by the stroke (Φ), pitch (α) and stroke plane deviation angles (θ), the phase lag ($\Delta\Phi$) in between the movement of each gill, the location of the gills on the body, and the physical development of the insect as it grows and increases its Reynolds number.

The study by Sensenig et al. (2009) provides a comparison of the kinematics for mayflies functioning at Reynolds numbers from 2.3 to 21.6. At a Reynolds number of 2.3, the gills have large stroke and pitch ranges and minor stroke plane deviation. The motion is also observed to be highly asymmetric. The gills spend about 1/3 of the cycle retracting and 2/3 of the cycle protracting, with a much higher maximum velocity on the retraction portion of the stroke. The pitching motion also creates an asymmetry, as the gill pitches on the recovery portion of the stroke, and remains approximately perpendicular to the stroke plane on the power portion of the stroke (Sensenig et al., 2009). This movement is considered rowing because the flow associated with this motion is directed ventrally, approximately parallel to the stroke plane (Sensenig et al., 2010). For the higher Reynolds number case, the pitch range is much smaller and the stroke plane deviation is close to zero. The stroke range is also

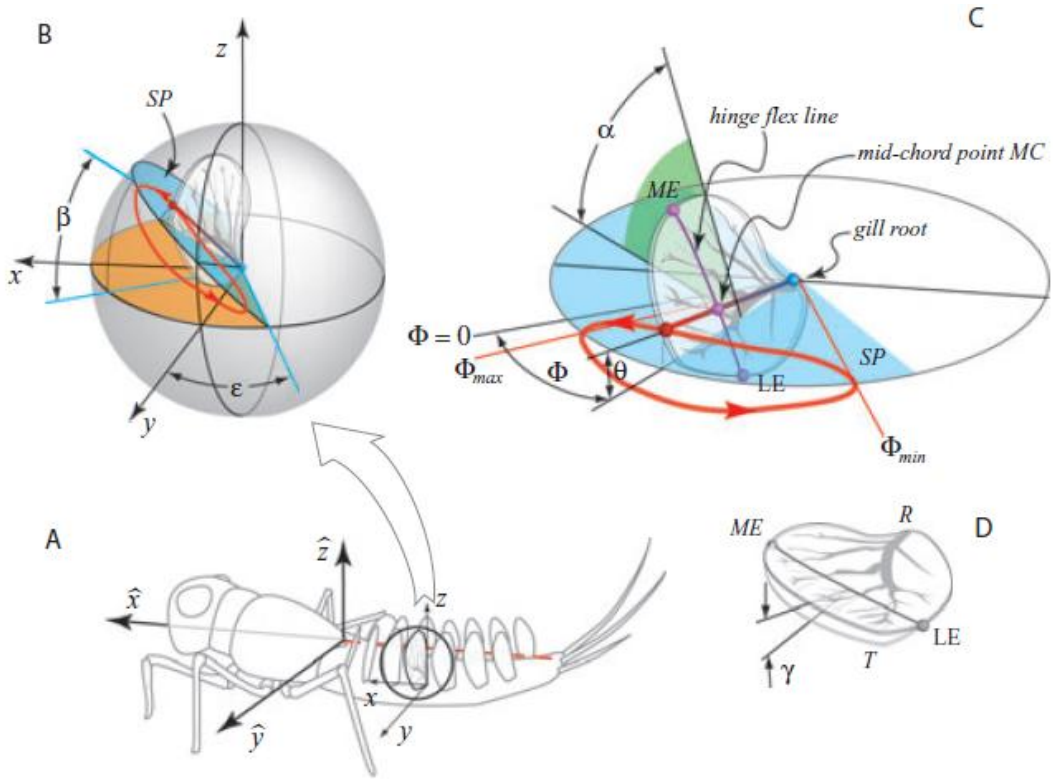

Figure 2.1: Spatial axes and angles used in characterizing gill-plate kinematics in the nymphal mayfly. A, diagrammatic mayfly nymph from an oblique lateral perspective showing spatial coordinates relative to the body ($\hat{x}$, anterior–posterior axis; $\hat{y}$, transverse axis; $\hat{z}$, dorsal–ventral axis) and parallel coordinates originating at the gill root (x, y, z). B, C, key kinematic parameters. The stroke plane (SP) for each cycle (indicated in blue) is defined by three points: the gill root and the anterior and posterior extrema of the gill mid-hinge point. The orientation of the SP is defined by the stroke-plane inclination angle (β), measured between the SP and the horizontal (x–y) plane within a mutually orthogonal plane, and the stroke-plane lateral offset angle (ε), which indicates where the SP crosses the horizontal (x–y) plane. The instantaneous position of the mid-chord point (path indicated by the red curve) is then given by the combination of the stroke angle, Φ, and stroke-plane deviation, θ, which measures the angular displacement along and normal to the SP, respectively. The stroke angle origin (Φ = 0) is referenced to the position where the SP intersects the transverse-vertical (y–z) plane, and the respective posterior/anterior excursion limits are given by Φmin/Φmax. The orientation of the gill plate is referred to as the pitch, α, which is defined as the angle between the gill plate and the SP, as measured in a mutually orthogonal plane. D, gills in larger nymphs have a distinct transverse hinge flex line. From Sensenig et al. (2009, Fig. 2).

smaller, but is still large compared to the pitch and stroke plane deviation. The individual kinematic parameters do not show the same asymmetries as in the lower Reynolds number case. The gill spends approximately the same amount of time moving forward as it does backwards, and the pitch and stroke plane deviation are relatively small (Sensenig et al., 2009). This is considered flapping motion because the net flow is directed dorsally, essentially transverse to the stroke plane (Sensenig et al., 2010).

Each of the gills moves with similar motion, but there is a phase lag of approximately 90º between each adjacent gill. This means that gill 2 and gill 6 move approximately in phase with each other. It was documented that this phasing produces a time-dependent array of vortices which are associated with the current produced by the gills. As the animal size increases, the oscillation frequency of the gills and radii of these vortices do not change. Therefore the vortex radius relative to the inter-gill distance decreases as the animal grows. (Sensenig et al, 2010). It was observed in a previous study that the mayfly nymph makes the rowing to flapping transition at a Reynolds number of about 5, which corresponds to the case where the mean inter-gill distance is equal to the vortex radius (Sensenig et al, 2009). This is a critical observation, because it may indicate a point in the relationship between the circulation mechanism and its self-generated vortices where a shift should be made from rowing to flapping kinematics for an array of appendages (Sensenig et al, 2010).

Other characteristics of nymph gills that may have an effect on the efficiency of the movement and fluid pumping is the orientation of the gills and the physical shape and structure of the gills themselves. The stroke axes of the gills are inclined from

the body by an angle β, which is referred to as the stroke plane inclination angle. For the fourth gill this inclination angle is about 52º for the lower Reynolds number case and 61º for the higher Reynolds number case. This angle varies some for each of the seven gills. The stroke ranges of the gills are also centered at angles that are directed slightly towards the posterior of the animal. As the animal grows the shapes of the gills change and they develop a passive hinge that allows the gill to flex in a preferred direction (Sensenig, et al., 2009).

Since mayfly nymphs transit through a Reynolds number range where the most efficient method for pumping is less well understood, a study on the influence of the traits described above is necessary to provide details on how an efficient pump could be designed for this regime. All of these traits have potential influence on the pumping efficiency of the gill arrays used by mayfly nymphs. Since it is unknown for what purpose the mayfly exemplifies each of these parameters, it is necessary to perform an experiment where individual parameters can be varied so that their influence on the pumping can be determined. The current experiment focuses on the influence of the stroke and pitch amplitudes of mayfly nymph gills and the phase lag in between these gills. While it has been shown that a wave can be used as an additional method of producing fluid transport for both low and high Reynolds numbers, the influence of the size of the phase lag that creates this wave has not been explored in detail for Reynolds numbers in the transitional regime between 1 and 100. In order to provide insight on why the mayfly uses a specific phage lag and whether this phase lag would be preferable for technology modeled after such an array, a robotic array of gills was constructed allowing kinematic parameters to be modified

individually for different test cases. For this experiment, five different test cases were examined: four with different phase lags and one with smaller stroke and pitch amplitudes. In order to compare the net pumping and pumping efficiencies of the gill arrays for these different test cases, it is necessary to take velocity data for a complete volume surrounding the gill array. This experiment uses stereoscopic particle image velocimetry (PIV), which provides all three components of the unsteady velocity. This allows direct computation of the net flux and flow-induced dissipation for each test case. These parameters as well as qualitative observations about the flow fields for each case provide details on why the mayfly might use one phase lag over any other and the advantages of using this phase lag for a pumping array at low Reynolds numbers.

Chapter 3

Experimental Design, Setup, and Analysis Techniques

3.1 Introduction

The purpose of this experiment is to study how changing select kinematic features of an array of oscillating plates influences net pumped flux through the array and the hydrodynamic mechanisms that generate the net flow. The geometry and kinematics of the specific plates used in this study are based on the gill plates of the mayfly nymph *Centroptilum triangulifer*. In order to perform this experiment, it was necessary to develop a device that could both replicate the nominal kinematics of typical mayfly gills as well as extend the range beyond what was exhibited in nature, so that the influence of the kinematic parameters on the fluid flow could be better understood. The array of gills used for this experiment was designed using a simplified geometry compared to the live mayfly and was controlled using programmed micro-servomotors. In order for the device to function at a comparable Reynolds number to a live mayfly nymph, the device was scaled to a larger size, but the frequency of the oscillations was decreased and a fluid with a higher viscosity was used. The fluid flow was measured using stereo Particle Image Velocimetry (PIV) over a three dimensional volume made up of thirty two planes of data. This technique measures all three components of the unsteady velocity field, allowing the flow-induced dissipation and the net pumping rate to be directly computed. Images were recorded at seventeen phase angles throughout the oscillation cycle, providing enough data to observe the flow patterns throughout an entire cycle as well as calculate the

cycle-averaged velocities, dissipation and pumping rate. For this experiment, measurements were taken for five different test cases. The kinematic parameters that were varied were the phase lag ($\Delta\Phi$) between each gill plate and the amplitudes of the stroke and pitch angles of the gill plates. Data was taken for four different phase lags: 0°, 90°, 180° and 270°, all with the same programmed stroke and pitch amplitude. For a phase lag of 90°, data was taken for two different stroke and pitch amplitudes. In order to take this data, the experimental device had to be able to control the phase lag between the motion of the gills in the array as well as the amplitudes of their stroke and pitch. The details of this device and the test set-up that allowed three dimensional data to be taken are described in the sections below.

3.2 Experimental Design

3.2.1 Kinematic Design and Actuation Method

In order to study the effect of select kinematic parameters of an array of oscillating plates, it was necessary to design a device that could mimic the motion of the gills of a mayfly nymph as well as perform controlled kinematic variations of this movement. The kinematics and physical design of the gill plate array was taken from Sensenig *et al* (2009). Kinematic definitions from this previous study can be seen in Figure 2.1. For simplicity and cost considerations, the physical structure was simplified and only the most significant kinematic parameters were considered. In order to understand the reasoning behind the simplified kinematics, a brief review of the animal model is necessary.

A mayfly nymph has seven pairs of gill plates on the lateral dorsal side of its body. The generalized motion of a ball-joint appendage is characterized by 3 degrees of freedom: two angles to describe the spherical angles to position the axis of the appendage, and a third angle to specify the orientation of the appendage about its axis. In most flapping appendage systems, the motion is referenced to a coordinate system aligned with the plane representing the "average" motion of the appendage (referred to as the stoke plane, see Figure 2.1) The position of the appendage axis relative to this plane is then specified by the stroke angle and the stroke plane deviation angle, and the orientation of the appendage about its axis is specified by the pitch (see Figure 2.1). For the model animal, Sensenig *et al* (2009) quantified the stroke angle, pitch angle and stroke plane deviation of each gill. A summary of the results for gills 4 and 5 can be seen in Figure 3.1. This previous study found that for gill 4 for a Reynolds number of 2.3, the stroke angle (Φ) has a range of 49º, the pitch angle (α) has a range of 36º and the stroke plane deviation (θ) has a range of 11.55º. The stroke angle for this case ranges from about -58º to -9 º, and the pitch angle ranges from 54º to 90º. For the case where the Reynolds number is 21.6, the stroke angle has a range of 24º, the pitch angle has a range of 8º and the stroke plane deviation has a range of 1.82º. The stroke angle ranges from about -38º to -18º and the pitch angle stays close to 90º. In both cases, the largest motion is the stroke. For the lower Reynolds number case, the pitch appears to have comparable significance to that of the stroke angle, while for the higher Reynolds number case, the pitch is comparably small. The stroke plane deviation is the least significant factor in both

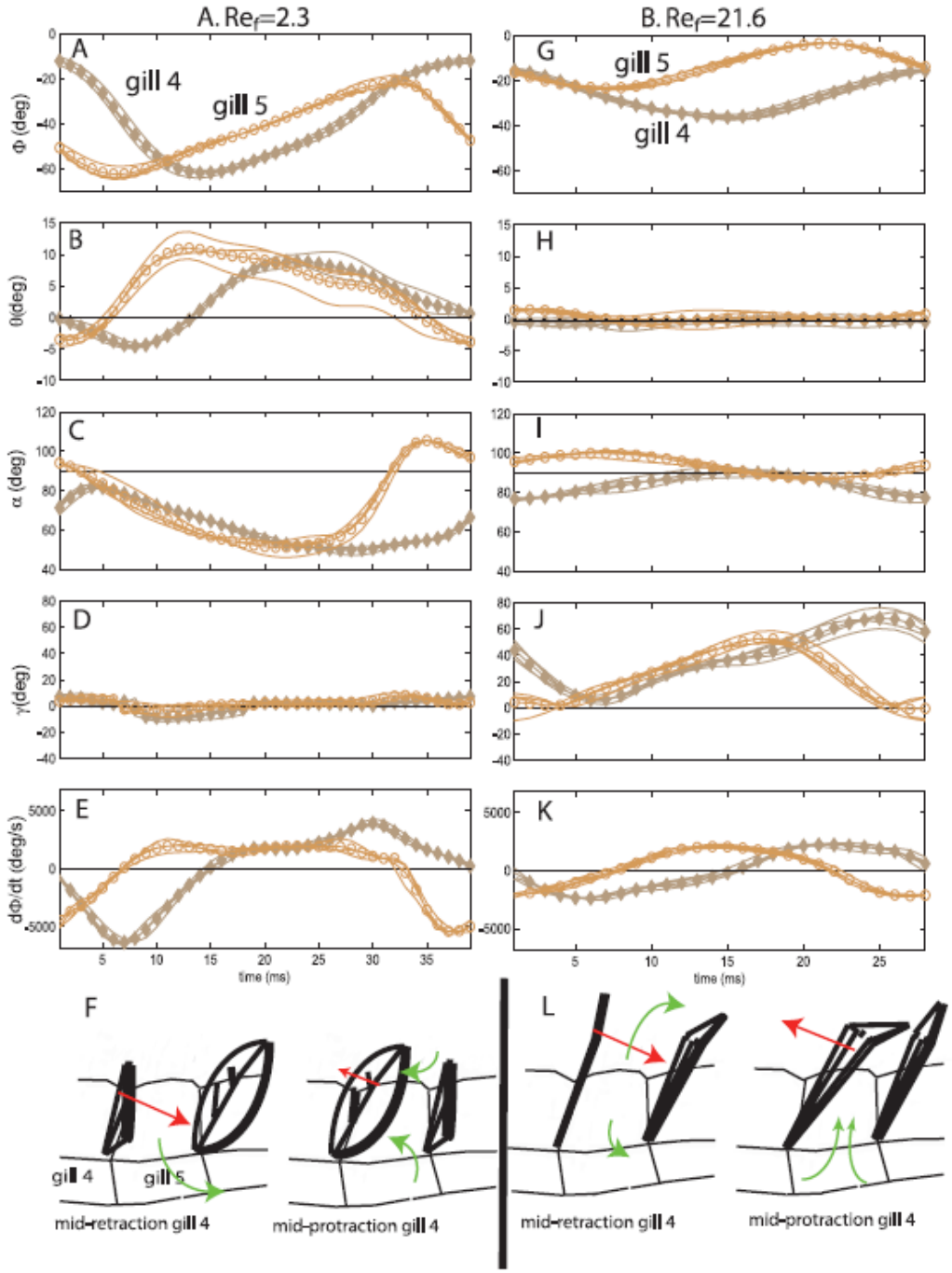

Figure 3.1: Kinematic parameters for gill 4 (solid diamonds) and gill 5 (open circles) over a stroke cycle. Initiation of retraction of gill 4 marks time = 0 ms. Stroke angle angular velocity is negative during retraction. Re_c number is that associated with the space between the gill indicated and its posterior partner. Thin lines represent individual strokes, whereas thick or dotted line represent mean of individual strokes (*N* = 4). Curved arrows represent the mean flow at that phase, whereas straight arrows indicate gill velocity at that phase. A, B, C, D, E, F, *Re* = 2.3. G, H, I, J, K L, *Re* = 21.6. From Sensenig et al (2009, Fig. 5).

cases. For the lower Reynolds number case, it is small compared to the stroke and pitch angles, and for the higher Reynolds number case, it is close to 0º.

Only the most significant kinematic parameters are replicated in the designed device. Based on the analysis above, this would require the stroke and hinge angle motion for the high Reynolds number case, and the stroke and pitch for the low Reynolds number case. In both cases the stroke plane deviation is the least significant and is not varied in the designed device. By retaining stroke and pitch functionality, the device has the ability to replicate simplified kinematics for both cases, as the hinge motion is not actuated and depends on the construction of the gill plate.

To simplify the kinematics of this project, identical gill planform shape and kinematics were used for all gills. These were based on the kinematics of gill 4 examined by Sensenig et al (2009), as this was from the mid-point of the array. For this specific study, the stroke was altered with respect to the real animal kinematics such that the gill stroke was centered around 0º (the axis normal to the axis of the animal) instead of being centered around a more posterior facing angle. Figure 3.1 also shows a phase lag that can be seen between the patterns of the two gills. The phase lag is approximately 90º between all of the adjacent gills in the array. This means that gills 2 and 6 are approximately in-phase with each other. The current study focuses on different phase lags between the gills, which imposes limitations on the kinematics. Due to the use of identical kinematics for all gill plates over a wide range of phase lag angles, it was necessary to limit the range of the stroke and the pitch to prevent collision between adjacent gills and control linkages. The phase lag

of 270º required that the stroke and pitch be reduced to 64% of the values originally reported by Sensenig et al (2009). The same pattern for the stroke and pitch are followed, but this gives a reduced design stroke range of (29.9°) and pitch range of (22.1°). The same stroke and pitch ranges were used for all four of the phase lags that are tested. For a phase lag of 90°, a second test case was performed with even further reduced stroke and pitch amplitudes of 15° and 11°, 32% of the original values reported. This case was studied to determine the role of stroke amplitude on the flow efficiency and pump rate. The reduced amplitude case was conducted with a phase lag of 90°, as this is the phase lag closest to that of the actual mayfly gills.

The robotic gill array designed to achieve the movement described above does so using two micro servomotors, crank arms and sliding linkages to control each gill (See Figure 3.2). The linkage is designed such that correlated motion of both servos changes the stroke of the gill, while a differential movement provides the pitch variation. The drive motors are inexpensive hobby servomotors that are relatively easy to control (Hitec HS-81). The angular position of the motor's shaft is controlled by the duty-cycle of a standardized 50 Hz TTL square wave signal. For this project, these signals are sent using an SSC-32 Lynxmotion servo controller, which can control up to 32 servomotors. The motion is specified by sending the controller the specified target locations and times to be achieved, and the controller then sends the appropriate pulse signals to match the target motion. The code used for this experiment was written in C# (Microsoft) and can be found in Appendix A. Servomotors expect their position to be updated every 20 ms. The SSC-32 will

continue to send the servos the same position signals every 20 ms until they have reached that position or until it receives an updated position signal from the computer.

To achieve the correct stroke and pitch angles for the mayfly gills, the position waveforms were divided into discrete points and signals were sent to the SSC-32 every 30 ms, which is the recommended rate. It is important to note that DC hobby servomotors, such as the HS-81, attempt to run at a single angular velocity determined by the voltage applied to the motor. This poses difficulty in generating a smooth, continuous motion. If the angle that the servomotor is commanded to reach is close to its current position, such that it will reach the destination before it receives its next command, it will stop at this location and wait for the subsequent motion command. It is possible to create a continuous motion if the servomotors velocity is operated within a reasonable range below the maximum velocity that the motors are trying to achieve. This maximum speed can be varied by changing the amount of voltage supplied to the motors. By matching a maximum speed possible with a cycle frequency that requires this maximum speed, it is possible to create smooth motion for the mayfly gill plates. To do so, the servomotors were supplied with a voltage of 3V. This is the lowest voltage that can be supplied to the motors and have them still be operational. This low voltage is required because of the low speeds necessary for the desired frequency for the Reynolds numbers used in this experiment. A separate power supply was used to power the SSC-32, which required an input of 8.5 V.

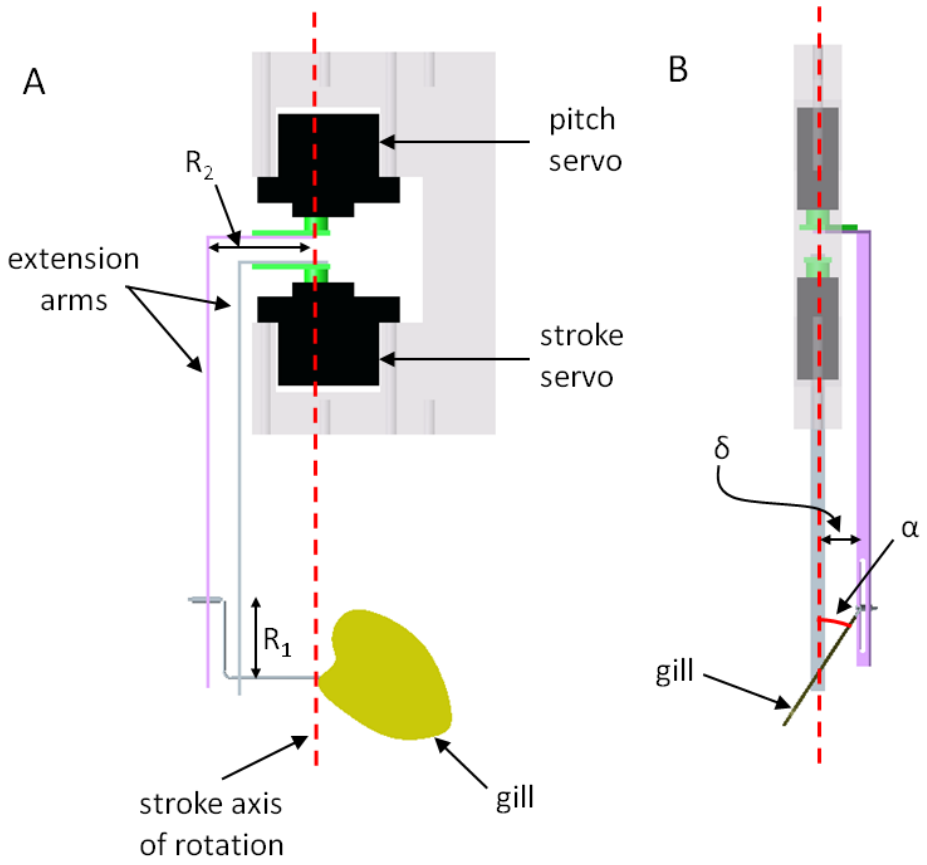

Figure 3.2: Two hobby servomotors are used to control the kinematics of a single gill plate. The stroke angle of the gill is directly coordinated with the rotating motion of the stroke servo, while a differential movement between the stroke and pitch servos causes the gill to pitch (α). The lengths R1, R2 and δ are used in the calculation of this differential angle. Figure A shows the side view of this device. Figure B shows the end view of this device.

The geometry of the gill actuators is shown in Figure 3.2. To control the stroke of the gills, the servomotors are arranged such that the motor rotation axis is perpendicular to the stroke plane of the gill. A steel extension arm is attached to each servomotor and connected to a cylindrical metal rod that is embedded in the gill. When the servo arm moves, the extension arm rotates, which then moves the gill rod the same number of degrees as the servo arm. The pitch was measured so that $\alpha = 90^{o}$ indicates that the gill is vertical. The pitch of the gills is controlled using a second servomotor, aligned coaxially with the stroke servo. The metal rod that is embedded

in the gill passes through the stroke positioning arm, and then is connected to the pitch extension arm using a sharp s-bend terminated in a slotted connection. In order to prevent binding, the s-bend uses a rod-in-sleeve configuration to allow the pin connection to remain perpendicular to the slot during differential motion of the servos. The pitch is controlled by varying the difference in the angle between the two servo arms. When the two servos move together so that the extension arms connected to them stay parallel, the pitch remains fixed at α=90º. When there is a differential movement of the two arms, the metal rod in the gill is forced to rotate to compensate for the difference, which causes the pitch of the gill to change. Therefore the angles of the servo arms used to control the pitch are calculated relative to the positions of the stroke servo. The differential position between the two servo arms (γ) was calculated using the equation below, where R_1, R_2, and δ are labeled in Figure 3.2.

$$\delta = R_1 \sin \alpha$$

$$\gamma = \sin^{-1}\left(\delta / R_2\right)$$

A detail of the motion for three different positions is shown in Figure 3.3.

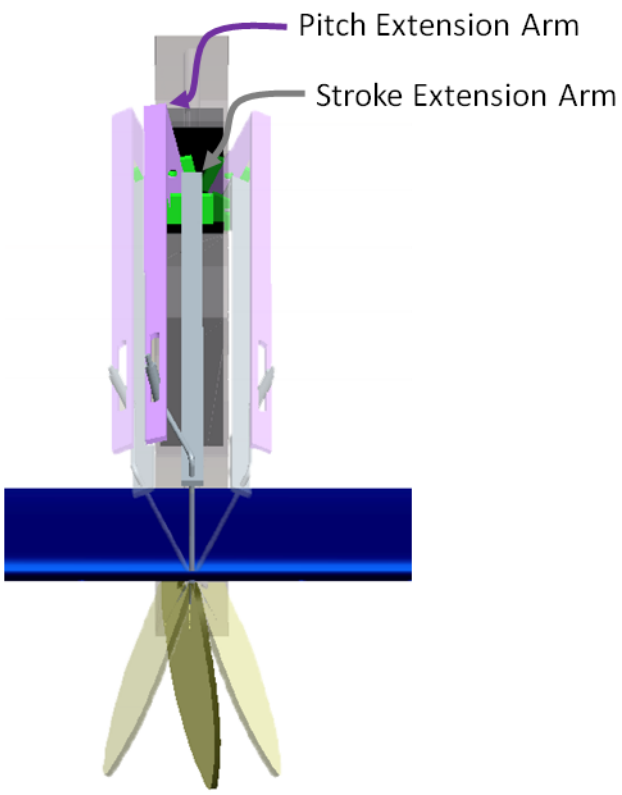

Figure 3.3: Detail showing actuation of gill stroke and pitch for three different positions within the cycle.

3.2.2 Physical Experiment

An important aspect of the pumping done by mayfly gills is the effect of combing the gills into an array. For this reason, five gills were used for the experiment to provide three "internal" gills free of end effects. On a live mayfly, there is some slight variation for the distance between the roots of the gills, but for this simplified model the gills are equally spaced with a root separation to gill length ratio of 0.6. This ratio is about 0.06 higher than the ratio for the root separation measured on the higher Reynolds number nymph (Sensenig et al., 2009).

The flow generated by each lateral set of gills on the mayfly nymph is symmetrical, so it was only necessary to construct a single side of the mayfly. To recreate the line of symmetry, the array was rotated so that the surface of the liquid

could be used as the centerline, as seen in Figure 3.4. A simplified body shell for the mayfly was constructed using stainless steel, which omits details such as a head or tail. It only models the abdomen of the mayfly nymph that the gills protrude from. The shape of the body was simplified so that it is uniform across all five gills, and the shape of the underside of the mayfly body was also altered so that the metal extension arms could be contained within the body and not disrupt the flow with their movements. The body is approximately 200 mm long with the third gill attached to its center.

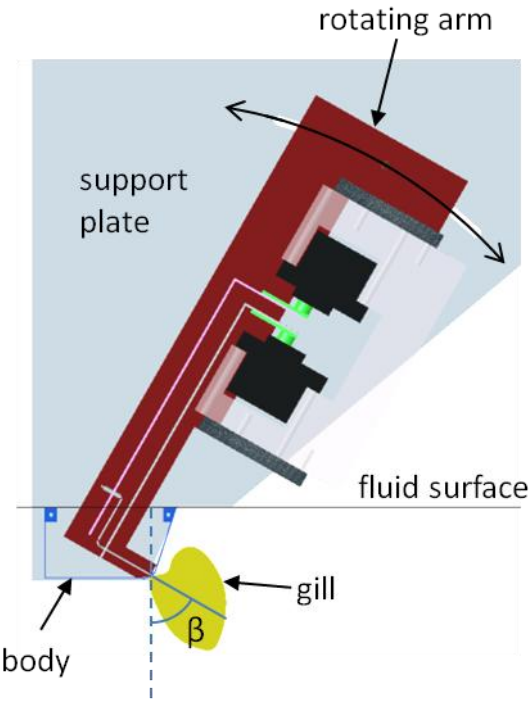

Figure 3.4: A side view of the support system for the servo controlled gill array. The modified shape of the body is shown in blue. The body remains fixed so its centerline is flush with the surface of the fluid. The rotating arms can be moved to determine the stroke plane inclination angle (β).

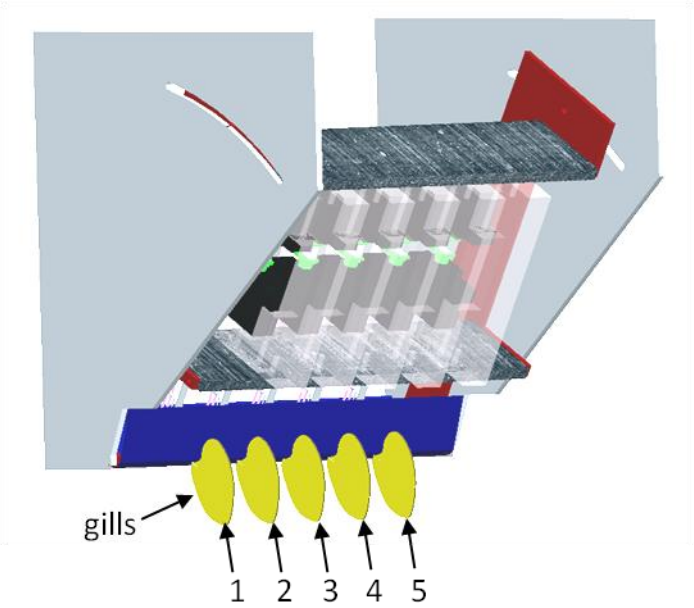

Figure 3.5: An array of five gills is supported using the system above. The rotating arms, marked in red, can be moved to determine the inclination angle of the gills relative to the body, marked in blue. The gills are labeled as they are referred to for the rest of the data.

As shown in Figure 3.4 and Figure 3.5 the body is supported by two aluminum plates, one on each side. These two plates have circular slots in them and a pinhole at the center of rotation, which is located in-line with the roots of the gills. Two aluminum arms are connected to the support plates with a pin at the center hole and with a bolt and nut at the circular slot. These arms are connected to aluminum plates that support the ten servomotors used for the actuation. The arms can be secured at various angles, which can be changed by varying where in the circular slot the bolt is secured. This geometry allows for a gimbaled motion about the gill root, which provides for variation of the stoke plane inclination angle (β) relative to the fixed body. This is a parameter that changes as the mayfly nymph grows. In the current work, the stroke plane inclination angle was fixed at 60^{o}.

The aluminum support plates for the robot are attached to a frame made out of 80-20. This frame for the mayfly robot is attached to linear bearings which are

connected to a larger 80-20 frame that also holds the tank for the experiment. The sliders allow the robot to be moved forward or backward in the tank at precise intervals. The position is controlled using a ¼"-80 thumb screw. The thumb screw pushes against a piece of the aluminum connected to the mayfly frame to move the robot forward and a spring is used to keep the aluminum in contact with the thumb screw at all times. A metric scale is attached to this device so that the location of the robot in the tank can be recorded.

A ten gallon fish tank is used for the experiment which is l_x = 260 mm wide, l_y = 310 mm tall and l_z = 510 mm long. The tank is filled with seven gallons of fluid, which gives a surface height of 200 mm. The height of the frame controls the height to which the robot is immersed so that the surface of the liquid can be matched with the centerline of the mayfly. During the experiments, the root of the gills are moved from a position of z = 155 mm to z = 217 mm. Although this variation will alter the details slightly in the outer flow, the end walls of the tank are sufficiently far from the measurement area to prevent alteration of the local flow.

It was necessary to construct the gills for the robot out of a transparent material that was stiff enough to mimic a flat plate. It was also advantageous to be able to embed the metal rod that controls the gill so its protrusion would not disrupt the flow. In this experiment, the gills were fabricated from a UV cured acrylic (LOCTITE 3525), that were cast into a model gill shape 1 mm thick. The liquid was first placed in a syringe and a vacuum pump was used to remove the bubbles. The syringe was then used to apply a thin layer of acrylic to a transparency. The area of the liquid was contained on the transparency using 1 mm thick glass slides on each

side. A second transparency was then placed on top of the liquid. This transparency had an image in the shape of a gill printed on it, with the gill being transparent and the rectangular area around it being black, as seen Figure 3.6. The image was printed three times using an ink printer so that the black rectangle was opaque. A UV light was then placed on top for six minutes to cure the material into the shape of the gill. The gill, transparencies and glass plates were then flipped over. A second mask was then taped onto this side of the transparency, which was aligned in the same position as the first one. The UV light was then placed on top for six more minutes. It is necessary to cure the acrylic from both sides because otherwise the cured gill will curl in the direction that the UV light was applied. Using this method allowed the metal rod that controls the position of the gill to be embedded into the material by placing it in the material before the acrylic is cured. An image of the final gill is shown in Figure 3.6.

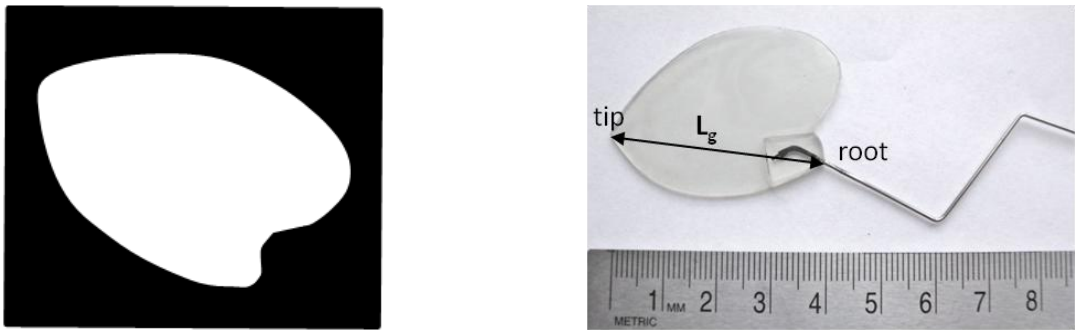

Figure 3.6: The left image shows the mask used when curing the acrylic with UV light. The right image shows an actual gill with a cylindrical arm embedded in it. The length of the gill, Lg, is measured from the root to the tip of the gill.

3.2.3 Scaling

The model was scaled using the oscillating Reynolds number The model was scaled using the oscillating Reynolds number $Re_f = L_g^2 f/ \nu$, where L_g is the length of the gill from the root to the tip in mm, as seen in Figure 3.6, f is the frequency that the

gills oscillate at in Hz and ν is the viscosity in cSt. In order to use the actuation method described, the gills had to be spaced far enough apart so that the servomotor arms did not collide with each other. This put a minimum length requirement on the device when scaling it. The servomotors also had a limited range of velocities over which they could operate, placing a limit on the frequency that could be used in the robotic model. Considering these factors, the robotic model dimensions are approximately 54 times those of the original mayfly. The motors are set to operate at a frequency of 1.9 Hz. If the motors are programmed to go slower than this, they reach their destination positions too quickly and the gills pause before the next position command reaches them. Due to this limitation, different working fluids are required to run a different Reynolds numbers. The fluid used for this experiment to achieve a higher Reynolds number is mineral oil, which has a viscosity of 175 cSt. To reach a lower Reynolds number a second fluid could be used, such as silicone fluid, with a viscosity of about 1000 cSt. An example of the scaling variables and calculations can be seen in Table 3.1 below.

Table 3.1: Experimental scaling based on oscillating Reynolds number $Re_f = L_g^2 f/ \nu$

	Live Mayfly	Robotic Model
v (cSt)	1	170
l_g (mm)	0.77	40
f (Hz)	37	1.85
Re_f	21.9	17.4

3.3 Experimental Setup and Acquisition Method

3.3.1 Particle Image Velocimetry Setup

A Litron nanoPIV Nd:YAG laser provided illumination for the measurement volume, which consisted of an approximately 2 mm thick vertical slice located at multiple planes parallel to the mayfly nymph body. The light sheet was formed using a 500 mm focal length cylindrical lens located 140 mm from the robot and a 60 mm focal length cylindrical lens located 130 mm from the robot. As illustrated in Figure 3.7, the light sheet was brought into the tank from the bottom, using a mirror to project the sheet from the below the tank.

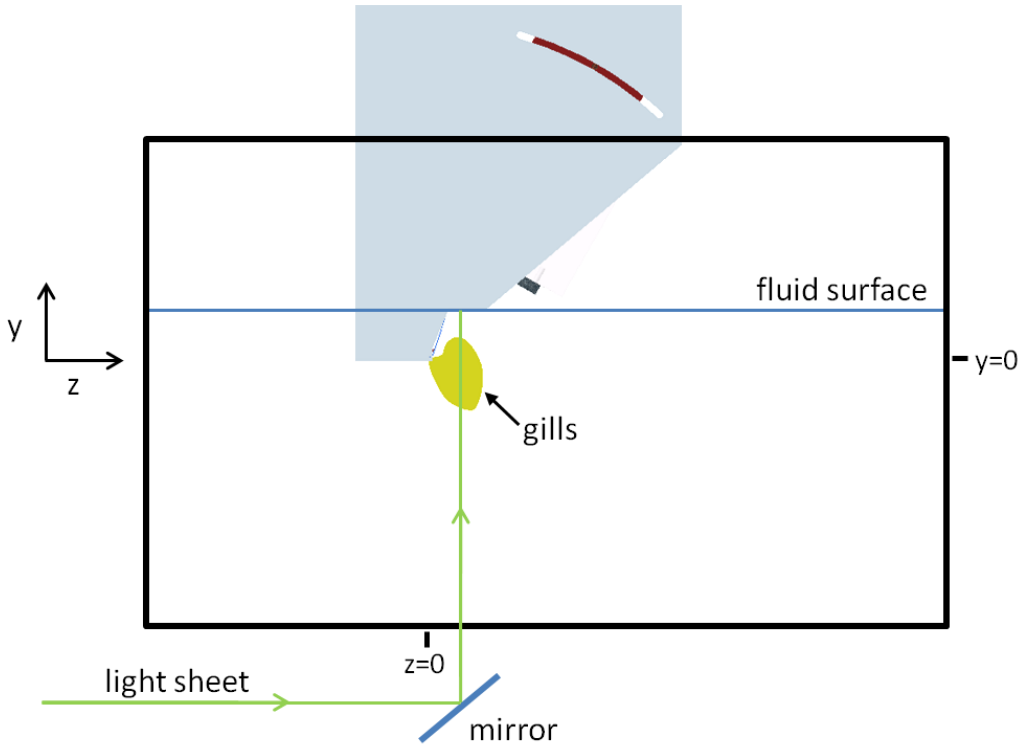

Figure 3.7: Sketch of the experimental set up in the y-z. The origin is located at the root of the gills.

The fluid motion around mayfly gills is a highly three dimensional flow. For this reason, stereo Particle Image Velocimetry (PIV) was chosen for this experiment. Stereo PIV uses two cameras with two different views to provide enough information to reconstruct all three velocity components. In this experiment, two Imager ProX

4M PIV cameras were placed on the same side of the laser sheet, as seen in Figure 3.8, with a 66º angle between them. In this configuration, the light scattering angle is 90º for both cameras, making the observed brightness of the tracer particles the same in both images. Because of the shape of the tank and the index of refraction between mineral oil and air, it was necessary to construct prisms on the tank in order to avoid distorting the camera views. As proposed by Prasad and Jenson (1995), the prisms were constructed so that the camera lens would be normal to the window, and they were filled with mineral oil to match the experimental fluid. The viewing angle, θ, affects the ratio of the random error amplitude in the out-of-plane displacement to the random error amplitude in the in-plane displacement. It has been shown that these amplitudes become equal at a viewing angle of $2\theta=90º$ (Lawson & Wu, 1997). It is generally accepted that using viewing angles between 60º and 90º provides sufficient accuracy for the out-of-plane displacement component.

For this experiment, two 50 mm lenses were used, and the cameras were placed approximately 600 mm away from the laser sheet. This gave a 154 mm by 90 mm field of view, which was wide enough to capture all of the gills throughout their stroke range. Scheimpflug mounts were used on each camera in order to satisfy the Scheimpflug condition required to focus the plane of particles illuminated by the laser sheet. The particles used for this experiment were 12μm hollow glass spheres. Because of the high viscosity of the fluid, the settling velocity of these particles is negligible compared to the velocities of the flow field being tested.

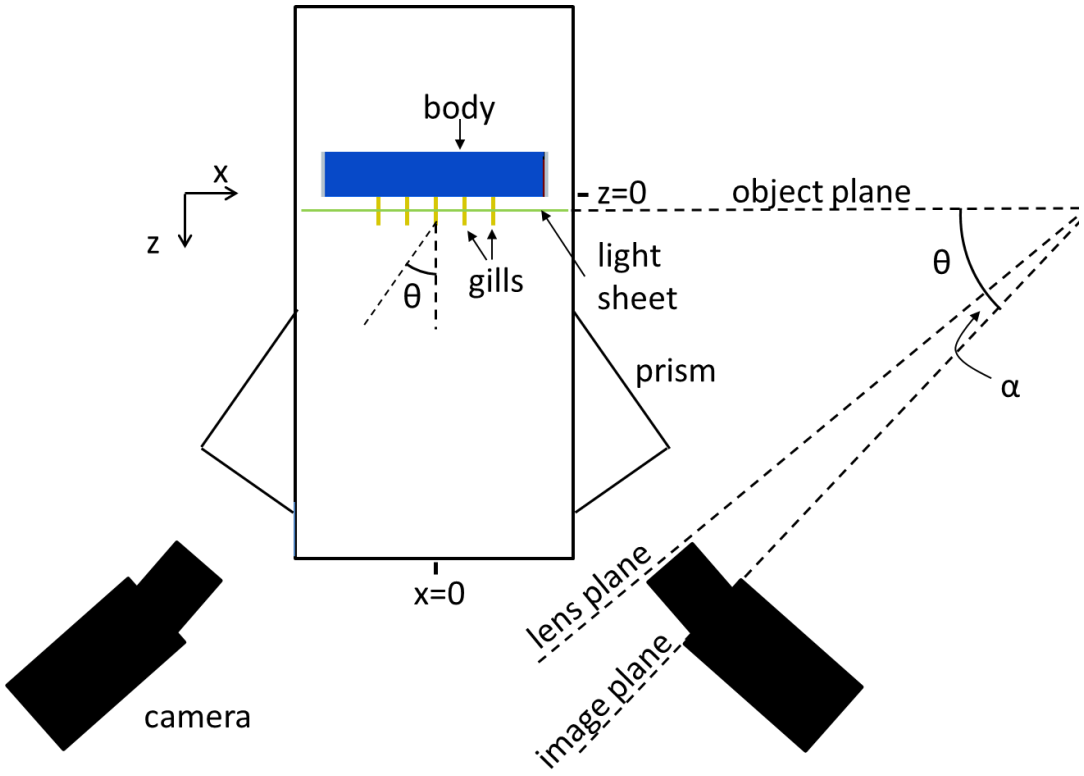

Figure 3.8: Sketch of experimental test setup from above, in the x-z plane. The origin is located at the root of the center gill. α is given by the Scheimpflug condition.

3.3.2 Acquisition Method

In order to reconstruct a three dimensional flow field around the gills, it is necessary to have three dimensional vector fields at multiple planes along the gill plates. Due to the low Reynolds number associated with the flow and the cycling kinematics of the mayfly gills, it was determined that the flow would be repeatable. Because of this, data could be recorded at different planes at different times, but at the same point in each cycle, and be reconstructed into an entire flow field. To take data at different planes, the robot was moved using the linear bearings and a thumbscrew mechanism with an attached scale so that the location of the planes could be

controlled and recorded. Thirty two planes of data were taken with 2 mm in between each plane, providing a similar spatial resolution to the in-plane interrogation (19 planes in total across a single gill).

The laser and the cameras were both controlled using an external trigger, which allowed the images to be taken at the same phase for each cycle. This was done using the SSC-32 servo controller. Once per cycle, when it sent signals to the servos, it would also send a signal to the programmable timing unit (PTU), which would cause the laser and cameras to operate. The point in the cycle that this signal was sent could be controlled with the C# program also used to control the servomotors. Images were taken at seventeen different phases within the cycle, each evenly spaced in time. This number corresponds to the number of commands sent to the motors during each cycle. Forty image pairs were taken at each phase in order to check for consistency and aid in processing and developing statistics such as the average or RMS of the vector fields associated with a phase. This was done to account for slight timing variations in the stroke kinematics, as described in the next section.

3.4 Data Processing Techniques

To obtain vector fields from the stereo images obtained during the experiment, the commercial software DaVis 7 was used. The processing takes a total of six steps to obtain a final average and RMS vector field for each phase angle for each plane in the experiment. The servomotors are programmed to operate at a frequency of 1.9 Hz. Due to the variations in tolerance in the individual linkage mechanisms, there is some variation in the exact locations of each gill from one cycle to the next. This

variation is generally within 1 mm, which is approximately the thickness of the gill. In order to avoid influencing the data with possibly singular outlier variations in the stroke position, the images are sorted and reduced so that only thirty of the images are included in the final vector field calculation. The sort criterion is based on minimum variance of the flow field with respect to a median composite flow field. To process the images, the ensemble mean image from each set is first subtracted from each image to limit the reflections on the gills and on the surface of the mayfly body. Once pre-processed, the images are then interrogated to get a first estimate of the velocity field. The vector fields are calculated using stereo-cross correlation, with two passes. The first uses window sizes of 64x64 pixels with 50% overlap, and the second uses window sizes of 32x32 pixels with 0% overlap. A median filter is used on the results and interpolation fills in any drop outs, so that the sorting process is not skewed by the presence of drop out vectors. A median vector field is then constructed by sampling the median vector at each location, and the thirty vector fields that have a global minimum deviation from this field are selected for reprocessing, and the ten remaining ones are discarded. The deviation is calculated by computing the residual of the vector field with respect to the median, and then normalizing each local residual by the median of the residuals in the set at that location (local residual). The magnitude of the normalized residuals at each vector location for a single vector field are then summed and divided by the total number of vector locations. The thirty vector fields with the lowest error are then selected to keep. The original images associated with the selected vector fields are then selected for final processing.

Once the final image set is determined, the revised image set is again pre-processed by subtracting the average of these images to reduce the reflections from the gills and from the laser sheet hitting the model mayfly body. These images are further preprocessed by applying particle intensity normalization with a scale of 5 pixels. A mask is applied to the area above the surface, and vectors are calculated using stereo-cross correlation with one pass that uses 64x64 pixel windows with 50% overlap and two passes that use 32x32 pixel windows with 0% overlap. On the first two passes a standard correlation function is used: $C = I_1 I_2$, where I_1 and I_2 are the image intensities of the first and second interrogation windows. For the final pass, a normalized correlation function is used: $C = (I_1 - I_{1avg}) * (I_2 - I_{2avg})/rms$. A median filter that removes vectors greater than two times the RMS (root mean square) value of its neighbors was used to post-process the data. Smoothing was applied in between passes, but was not used after the final pass. This means the velocity data used in flux calculations was not smoothed. Smoothing with a Gaussian kernel for 3x3x3 volumes was applied to the velocities before the vorticity was calculated. The average vector field and the RMS of the vector fields were then calculated from the thirty vector fields. When the individual planes of vectors are then constructed into a three dimensional data set, a second median filter is run for each plane in the newly constructed x-z direction. For the worst plane, approximately 4% of the vectors were filtered. The average number of vectors in a plane that were filtered was less than 1%.

3.5 Uncertainty

The greatest source of uncertainty in these velocity measurements was caused by the variation in the cycle length for each gill. The servomotors used to control the gills were given the same commands each cycle, but the rate that the motors responded to these commands varied slightly. This became noticeable when the forty PIV images were taken at a set phase within the cycle, because the location of each gill would fluctuate slightly from image to image. For this reason, only the thirty images whose vector fields best matched the median of the vector fields were used for the final average velocity field calculation. The uncertainty from this variation in the mean velocity field from the thirty sorted images was determined by calculating the RMS (root mean square) of the thirty vector fields, then dividing by the square root of the total number of vector fields. Dividing by the square root of the total number of vector fields gives the standard error of the mean (SEM) velocity.

This error was compared to values calculated for the sub-pixel interpolation error. To calculate this error, the thirty image sets taken at one phase in the cycle were extracted. Since this was a stereo PIV measurement, the second and forth frame of the image set were replaced by the first and third frame of the image set. This allowed two pairs of the same images to be correlated. Since the result of this correlation should be zero, because no displacement is occurring, the values that were produced represent the sub-pixel interpolation error. To get the uncertainty in the mean velocity field, the square root was taken of the sum of the squares of these error fields. These were then divided by thirty. These quantities, when compared for a plane of data, were found to be approximately an order of magnitude smaller than the

values found from the velocity field variation described earlier. For this reason, the sub-pixel interpolation error was deemed insignificant in comparison and was not included in the error propagation. Using the standard error from of the thirty images, the uncertainty in the mean is taken to be ±0.05 pixels or ±0.18mm/s. Note that the error in a single velocity realization greater by a factor of $\sqrt{30}$, or equal to 0.27 pixels. The uncertainty propagations for the uncertainty in the average flux and dissipation are included in Appendix C. This propagation yielded uncertainties of ±1% of the average of the total flow in and out of the control volume and uncertainties of approximately ±0.03% of the total, cycle-averaged dissipation.

Chapter 4

Results

4.1 Kinematics

The results of this experiment contain flow field data for five different test conditions. In the first four cases, the amplitude of the stroke and pitch are kept consistent for each case, but different phase lags are used in between each gill. The phase lags tested are approximately 0º, 90º, 180º and 270º. The fifth test case uses a 90º phase lag in between the gills, but it uses stroke and pitch amplitudes that are half the magnitude of the other test cases. The actual stroke and pitch of each gill plate achieved in the experiment will differ from gill to gill from the specified motion due to variations in the tolerance of the actuator construction.

In order to document the kinematics realized during the experiment, the gill motion was tracked using the images from the PIV measurements. Even though the index of refraction of the gill material (n_{gill} = 1.49) was similar to that of the oil used in the experiments (n_{oil} = 1.47), the edge of the gill still scattered sufficient light to be visible in the PIV image. For a single phase angle and imaging plane, the average corrected left and right PIV images were superimposed, revealing the location of the top and bottom edges of the gill intersecting the light sheet. These points were recorded for the five gills at every phase angle and every plane where the laser sheet intersected with the gill plates (19 planes total, providing 38 points on every gill). These points were then used in a least-squares fit to determine the ensemble-averaged planar orientation of each gill for each phase angle. By calculating a normal vector to

each plane, and knowing the stroke plane inclination angle, the stroke angle and pitch angle for each gill could be determined for each phase angle. The results of this kinematic tracking for each test case (distinguished by their phase lag, $\Delta\Phi$) can be seen in Figure 4.1. The black circles indicate the position commands that were sent to the servomotors. The same pattern was sent for each gill, just at different times, depending on the phase lag between the gills.

These graphs show the differences in the movement of the five gills. Gill 5 consistently has a lower stroke amplitude than the four other gills. Gills 1 and 5 have the highest pitch amplitudes out of the five gills in all of the cases except for $\Delta\Phi=270^{o}$, where the pitch amplitude of gill 5 is notably smaller than in the other case. Gill 4 consistently has the lowest pitch for each case. The pitch amplitude of gill 5 for the $\Delta\Phi =180^{o}$ test case is notably higher than it is for the other test cases. A variation in the central angle of the stroke and pitch between each of the gills can also be seen. The one with the largest difference is gill 1 for the fifth test case ($\Delta\Phi =90^{o}$, smaller amplitude).

The differences in the kinematics between each gill does cause some perceptible differences in the local velocity fields. Since the stroke and pitch amplitude of gill 5 is smaller for $\Delta\Phi =270^{o}$, the average velocity around this gill is smaller than the other gills and smaller than what was seen in other cases. Similar effects can be seen around gill 4.

Variation in the cycle timing can also be seen in these plots. This is most readily noticeable in the flow fields for the $\Delta\Phi =0^{o}$ case, when all the gills are supposed to be moving together (particularly for gills 1 and 5, which had the largest

variation from the programmed motion). It is evident from the plots that gills 1 and 5 are not always perfectly synchronized and may begin reversing their pitch direction or travel at different velocities at slightly different points in the cycle.

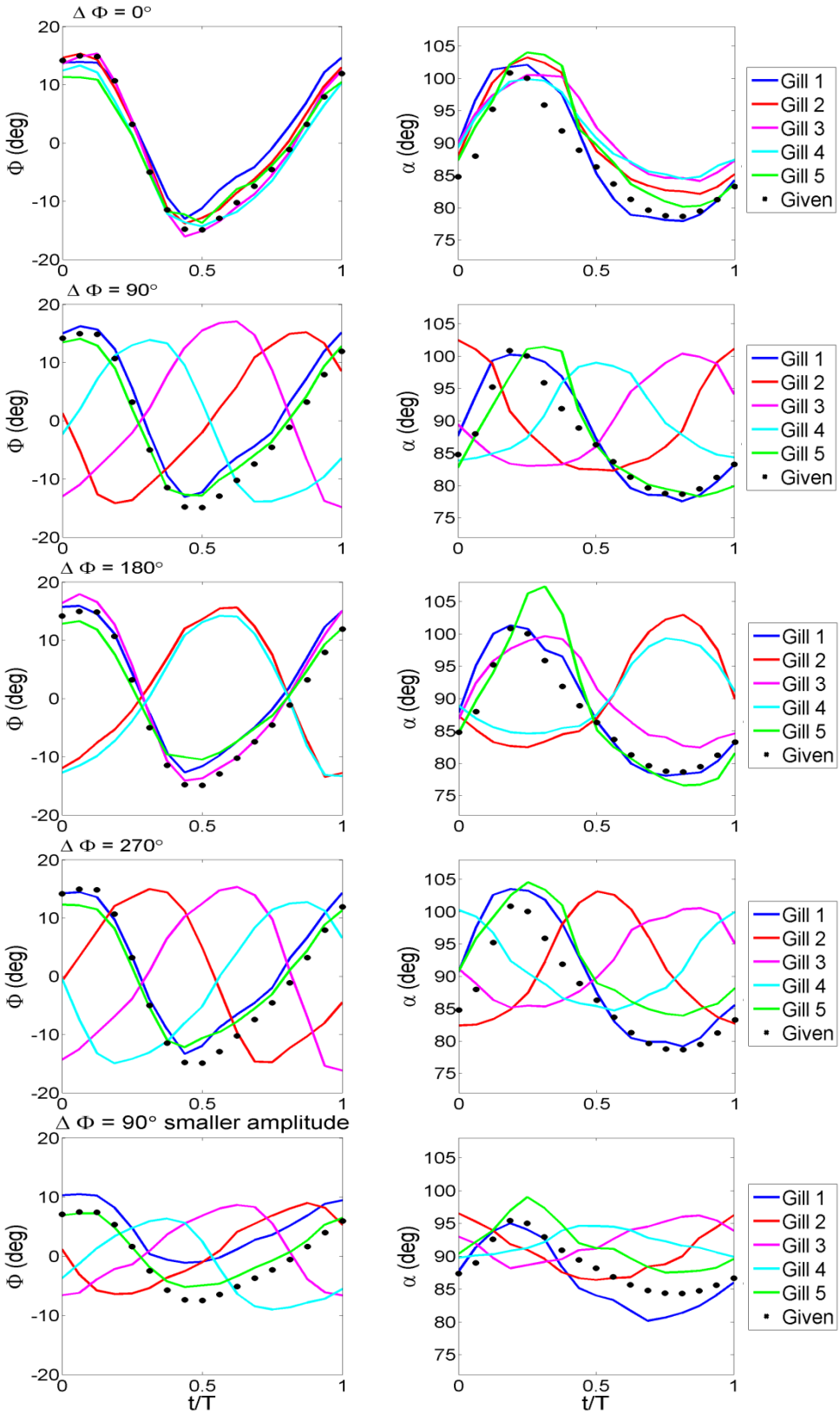

Figure 4.1: Measured experimental stroke (Φ) and pitch (α) for each gill for each of the five test cases.

4.2 Description of the Flow

4.2.1 Flow over a Cycle

The data was taken using stereoscopic PIV, allowing all three velocity components could be measured. Figure 4.3 through Figure 4.7 show the flow patterns (streamlines) with the normalized out-of-plane vorticity magnitude in the background for an *x-y* plane located across the center of the gills. Figure 4.8 through Figure 4.12 show the flow patterns in an *x-y* plane located across the center of the gills with the normalized velocity magnitude in the background. These figures show flow fields for eight different times throughout one cycle for each of the different test cases. The gray lines are streamlines, and the black lines represent the locations of the gills in the chosen plane at each time. The location of the *x-y* plane that the velocity and vorticity data is shown for is illustrated in Figure 4.2.

Differences in the flow fields of the actual mayfly gills and the robotic gills are exhibited due to using modified rowing (low Reynolds number) kinematics at a higher Reynolds number for the robotic experiment. To facilitate the reliable operation of the robot, the test case kinematics were modified from the original mayfly so that they function with: 1) a "centered" mean stroke and pitch (i.e. the average position is $\Phi = 0^o$ and $\alpha = 0^o$), 2) a reduced stroke and pitch amplitudes, and 3) nominally the same kinematics for the entire array. These differences influence the flow direction as well as the non-dimensional velocity magnitudes.

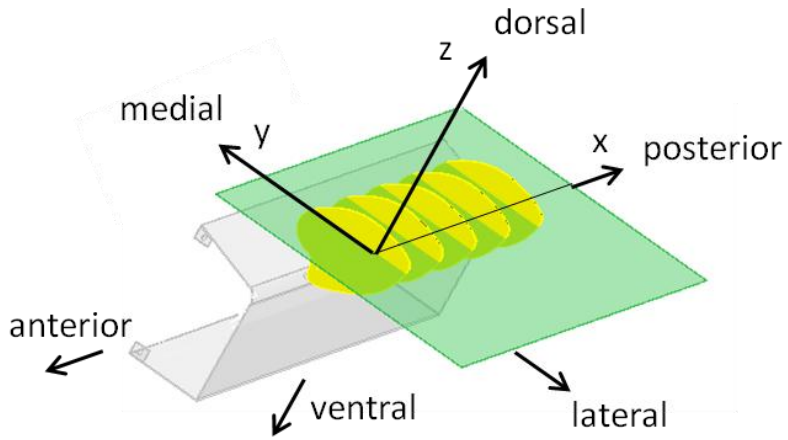

Figure 4.2: The location of the plane that the velocity fields are shown for. The gills are shown in yellow and the location of the modified body is shown in gray. The origin is located at the root of the central gill.

The first test case, which uses a phase lag of 0º, is shown in Figure 4.3 and Figure 4.8. For this test case the gills are approximately synchronized, and the flow patterns around each of the gills are similar to each other at each point in time. In Figure 4.3, it can be seen that as the gills retract (move in the $-x$ direction), positive vortices form around the edges of the gills closest to the body and negative vortices form around the tips of the gills furthest from the body. The magnitudes of the vortices further from the centerline of the body are higher than those closer to the centerline of body, because the gills are moving at a higher velocity at these locations, causing larger velocity gradients. The flow closest to the gills moves in the same direction as the gills, but the flow further from the gills begins flowing in the same direction as the gills, then reverses as the speed of the gills increase, and they move further towards the posterior. When the gills reach their most posterior ($-x$) position and begin to change direction, the direction of the flow directly around the gills is reversed, causing the signs of the vortices to be reversed. The vortices closest to the gill root become negative, and the vortices furthest from the gill root become positive.

The velocity of the gills is lower during the protraction (movement in the $+x$ direction) portion of the stroke, causing the peak vorticity magnitudes to be smaller during protraction than during retraction. Because the vortices on the ends of the gills are all rotating in the same direction, an area of circulation in the same direction is formed spanning the entire width of the array of gills. This causes the flow closest to the gills once again to move in the direction of the gills, but the flow further from the gills to reverse direction as the area of the circulation grows. As shown in section 4.1, the gills are only approximately doing the same motion. There are slight differences in the timing and amplitudes of the stroke and pitch, which causes the magnitudes of the vortices on each gill and the time that they occur to be slightly different. This can be seen in all of the test cases.

The second test case uses a phase lag of 90º between the gills, causing an antiplectic, metachronal wave to propagate from the posterior ($-x$) of the robot to the anterior ($+x$) as seen in Figure 4.4 and Figure 4.9. This case has kinematics most similar to those used by the live mayfly nymph. Differences in the flow fields of the actual mayfly gills and the robotic gills are exhibited due to the kinematic differences between the live mayfly and the robot described earlier. These differences influence the overall direction of the flow as well as the nondimensional velocity magnitude of the flow. It also changes the vortex interactions, because the spacing between adjacent vortices is different between the two cases.

While there are numerous differences in the kinematics used in this experiment and those of the actual mayfly, there are still qualitatively similar characteristics that can be seen when comparing the flow from both cases. Similar

vortex patterns can be seen around the individual gills. When a gill retracts, a positive vortex forms around the edge of the gill closest to the body centerline, and a negative vortex forms around the edge of the gill furthest from the centerline. When the gill protracts, the signs of the vortices are reversed, and the magnitude is lower due to the lower velocity. The size of the vortices relative to the size of the body for the robotic experiment are most similar to the size of the vortices compared to the body for the higher Reynolds number case for the live mayfly (Sensenig et al., 2010). This is expected, since the robotic experiment is also operating at this Reynolds number so the ratio of the frequency, size and viscosity is the same for both cases.

For both the live mayfly gills and the robotic gills, the 90º phase lag between the gills prevents the formation of unidirectional flow as observed for the $\Delta\Phi = 0^{o}$ case. This causes the vortices on adjacent gills to not always be rotating in the same direction. Because the gills spend more time protracting than retracting, on the edge furthest from the body there are always at least two positive vortices adjacent to each other. For this phase lag, the two or three gills with these vortices on them are relatively close together. This leaves the retracting gill, which has the highest velocity, relatively isolated, allowing it to force a larger volume of high velocity fluid in the posterior direction. As seen for the robotic test case in Figure 4.4 and Figure 4.9, when this fluid reaches the adjacent, protracting gill, it is redirected at an angle away from the centerline of the body. This pattern is repeated throughout the cycle, causing the flow further from the gills to move consistently away from the body ($-y$) and to the right.

The third test case, seen in Figure 4.5 and Figure 4.10, uses a phase lag of 180º. For this condition, every other gill is performing the same motion. Similar vortex structures are seen on each individual gill as in the previous cases, but the interactions have changed because of the difference in phase lag. Because adjacent gills are always moving in opposite directions, there are never two vortices of the same sign directly next to each other, but rather vortex dipole pairs that can act effectively to induce a local jetting motion outward from in between the two gills as they are squeezed together. As seen in Figure 4.5, this causes the flow to consistently move in a direction away from the centerline of the mayfly. Larger areas of re-circulating fluid can be seen on the right and left sides of the gill array. Finally, due to the fact that the gills retract more quickly than they protract, the flow moving away from the centerline of the body also consistently moves slightly towards the right.

The flow field of the fourth case ($\Delta\Phi = 270^{o}$) can be seen in Figure 4.6 and Figure 4.11. The gills in this case move in a symplectic, metachronal wave (wave motion in the same direction as the effective stroke). The vortex patterns for each individual gill are the same as for the previous cases, but the fluid interactions are different because of the change in the phase lag. As in the case of the 90º phase lag, there are always at least two gills protracting with positive vortices adjacent to each other on the distal gill edge, because the gills spend more time protracting than retracting. Due to the difference in phase lag for this case, the gills that are protracting are relatively far apart while, and the retracting gill is relatively close to the adjacent gills. This leaves less space for the gill to draw fluid from and less space before the flow it forces in the posterior direction is redirected by the adjacent, almost

stagnant gill. Figure 4.6 shows that the overall flow further away from the gills moves away from the centerline of the body and in the anterior (-*x*) direction. Like in the 90º case, the flow is moving in the opposite direction of the propagating wave.

The fifth test case, seen in Figure 4.7 and Figure 4.12, uses $\Delta\Phi = 90^{o}$, but the stroke and pitch amplitudes are half of what they were in the previous test cases. The vortex pattern for each of the individual gills is similar to those in the previous cases, but the magnitude of the vorticity is lower due to the lower velocity of the gills. The overall flow pattern is similar to the one seen in the previous 90º phase lag, but there are some notable differences. It can be seen in Figure 4.7 that similar to before, where there are adjacent protracting gills, the flow moves in the anterior (-*x*) direction close to the gills and away from the center line, in the posterior direction further away from the gills. Unlike in the previous case though, the fluid moves in from the anterior and outer directions towards the centerline of the body until it is redirected by the negative vortex adjacent to this positive pair. After moving around this vortex, the fluid moves back away from the body in the posterior direction. This makes the flow in the outer-posterior direction less continuous than what was seen in the earlier test case. Because the amplitude of the gill motion is smaller than it was in the previous case, the protracting gills are not as close together, and the negative vortex on the retracting gill is not as far from adjacent gills. This change in amplitudes results in an overall flow field similar to the previous 90º phase lag case, with the fluid moving away from the centerline and in the posterior direction, but there is greater fluctuation in the pattern over the cycle.

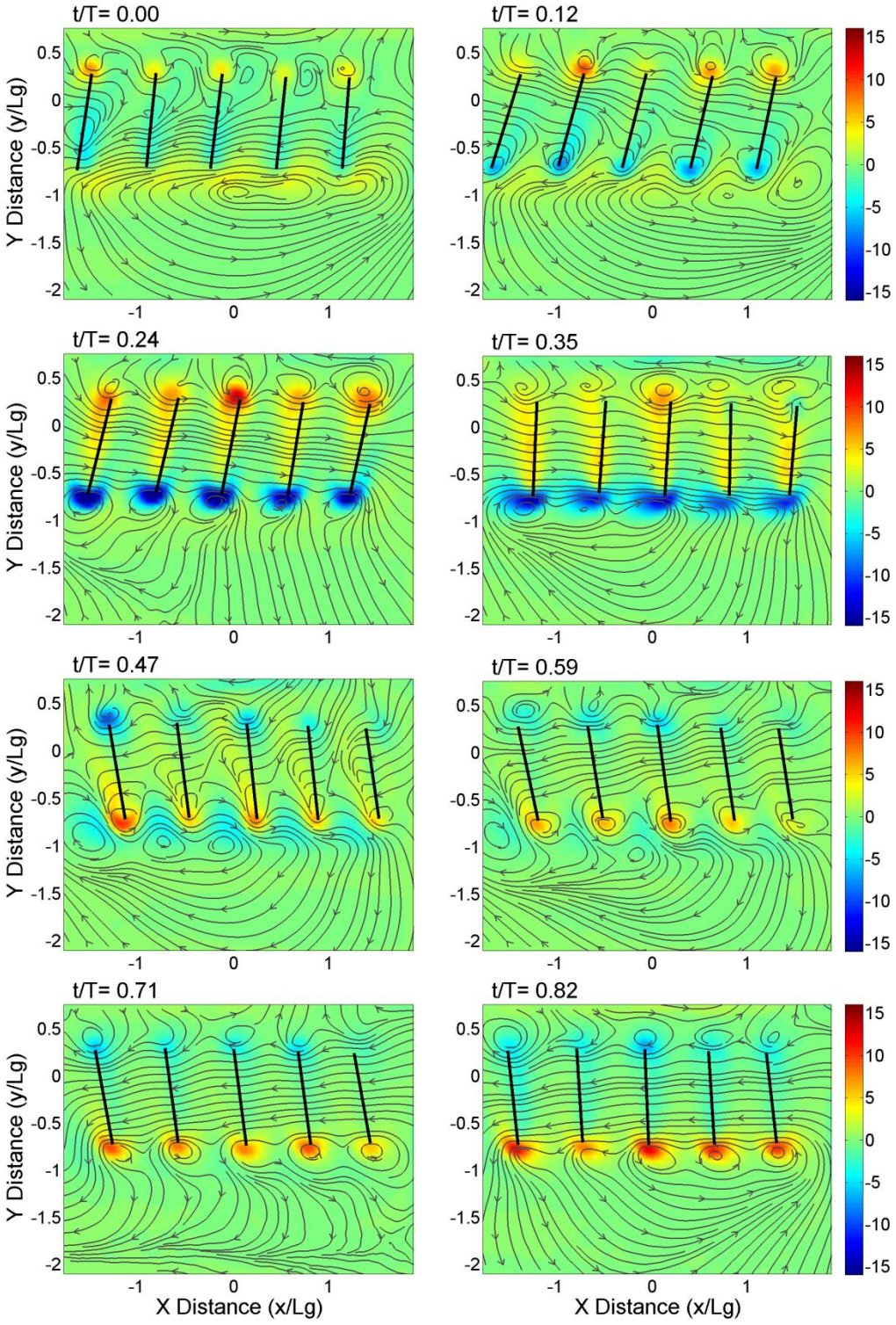

Figure 4.3: For ΔΦ =0º, the plots above are colored with the normalized z vorticity component and show streamlines of the x and y velocity components for eight different times throughout the cycle. The origin is at the root of the third gill. The black lines represent the gill locations at each phase.

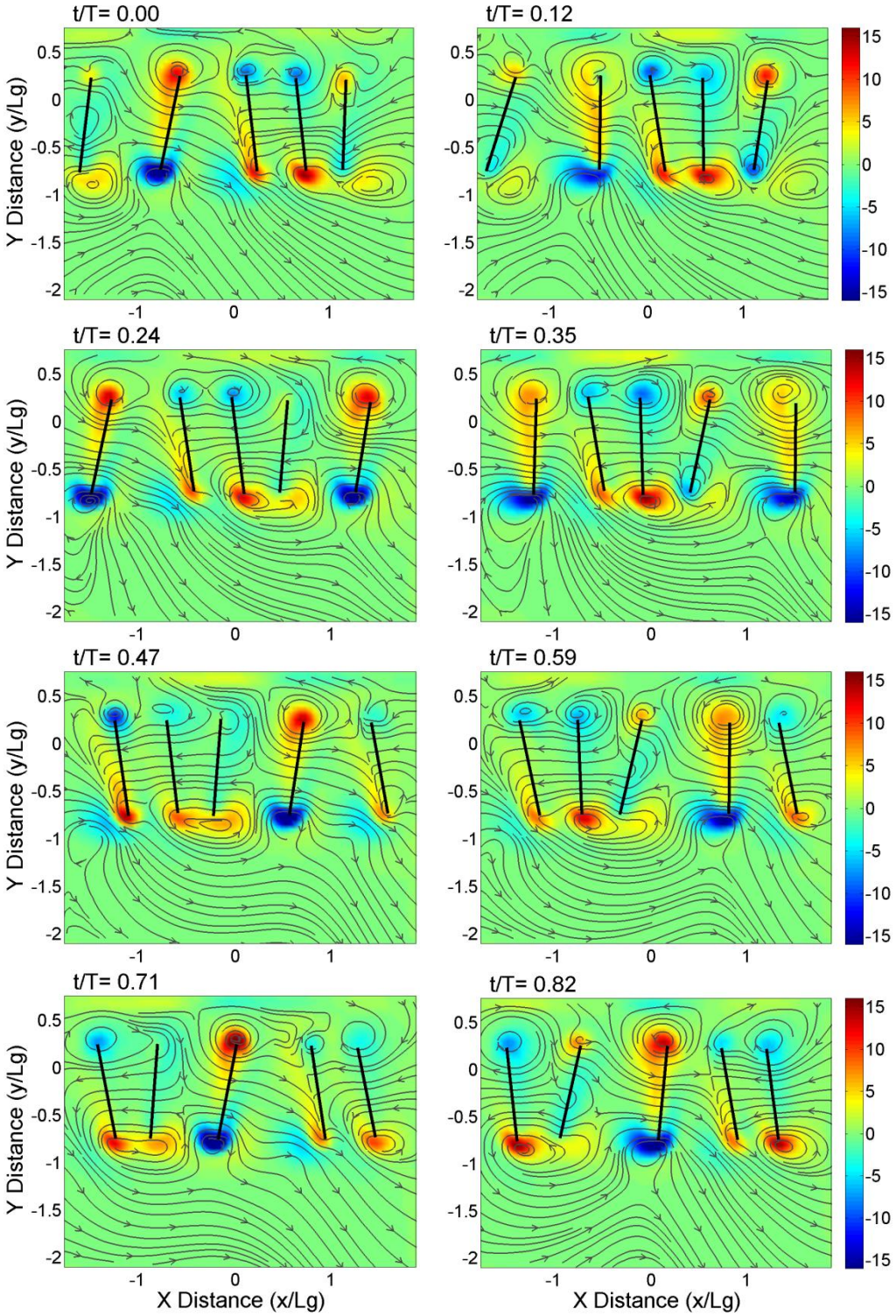

Figure 4.4: For ΔΦ =90º, the plots plane above are colored with the normalized z vorticity component and show streamlines of the x and y velocity components for eight different times throughout the cycle. The origin is at the root of the third gill. The black lines represent the gill locations at each phase.

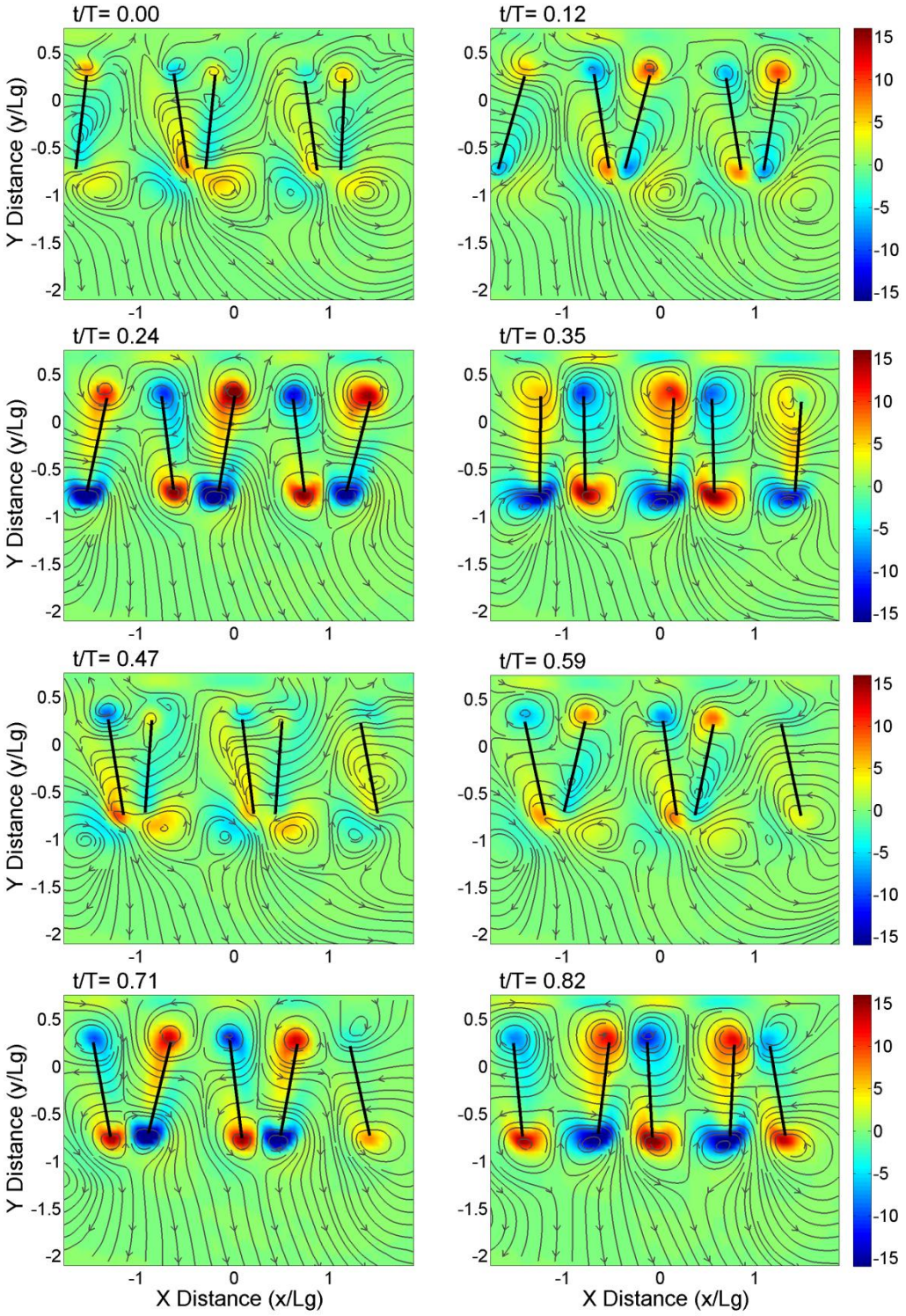

Figure 4.5: For ΔΦ =180º, the plots above are colored with the normalized z vorticity component and show streamlines of the x and y velocity components for eight different times throughout the cycle. The origin is at the root of the third gill. The black lines represent the gill locations at each phase.

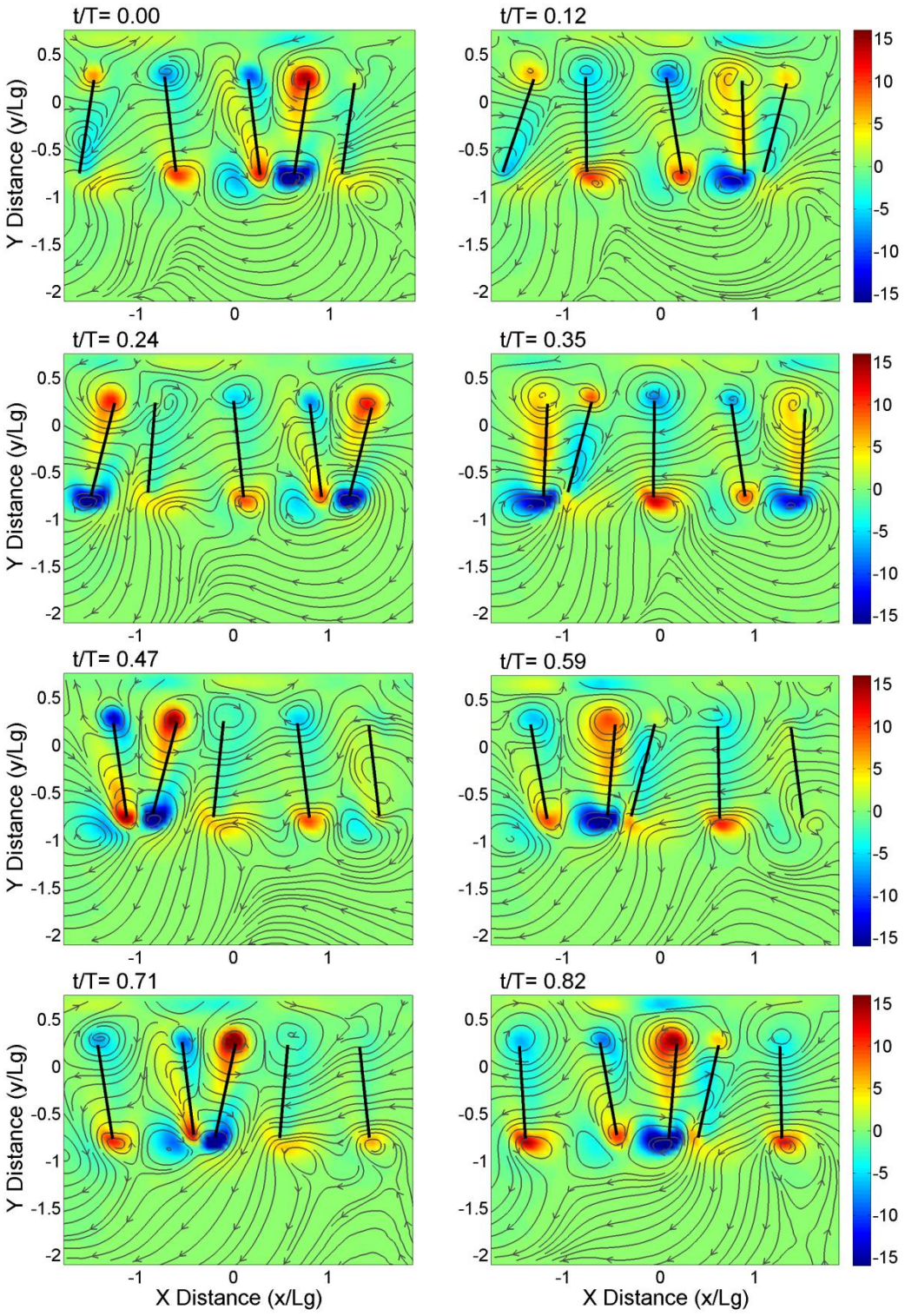

Figure 4.6: For ΔΦ =270º, the plots above are colored with the normalized z vorticity component and show streamlines of the x and y velocity components for eight different times throughout the cycle. The origin is at the root of the third gill. The black lines represent the gill locations at each phase.

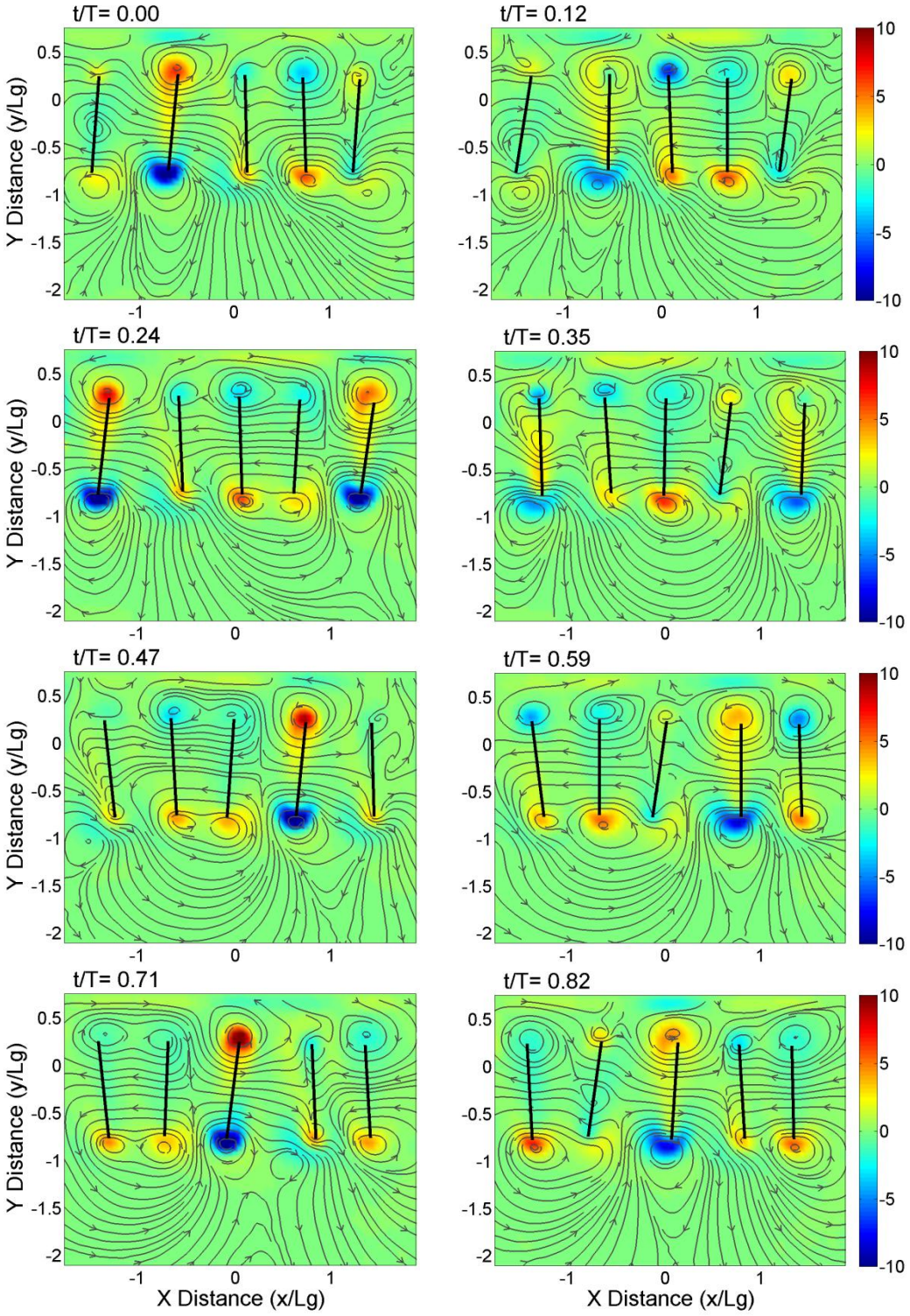

Figure 4.7: For ΔΦ =90º, but with smaller stroke and pitch amplitudes, the plots above are colored with the normalized z vorticity component and show streamlines of the x and y velocity components for eight different times throughout the cycle. The origin is at the root of the third gill. The black lines represent the gill locations at each phase.

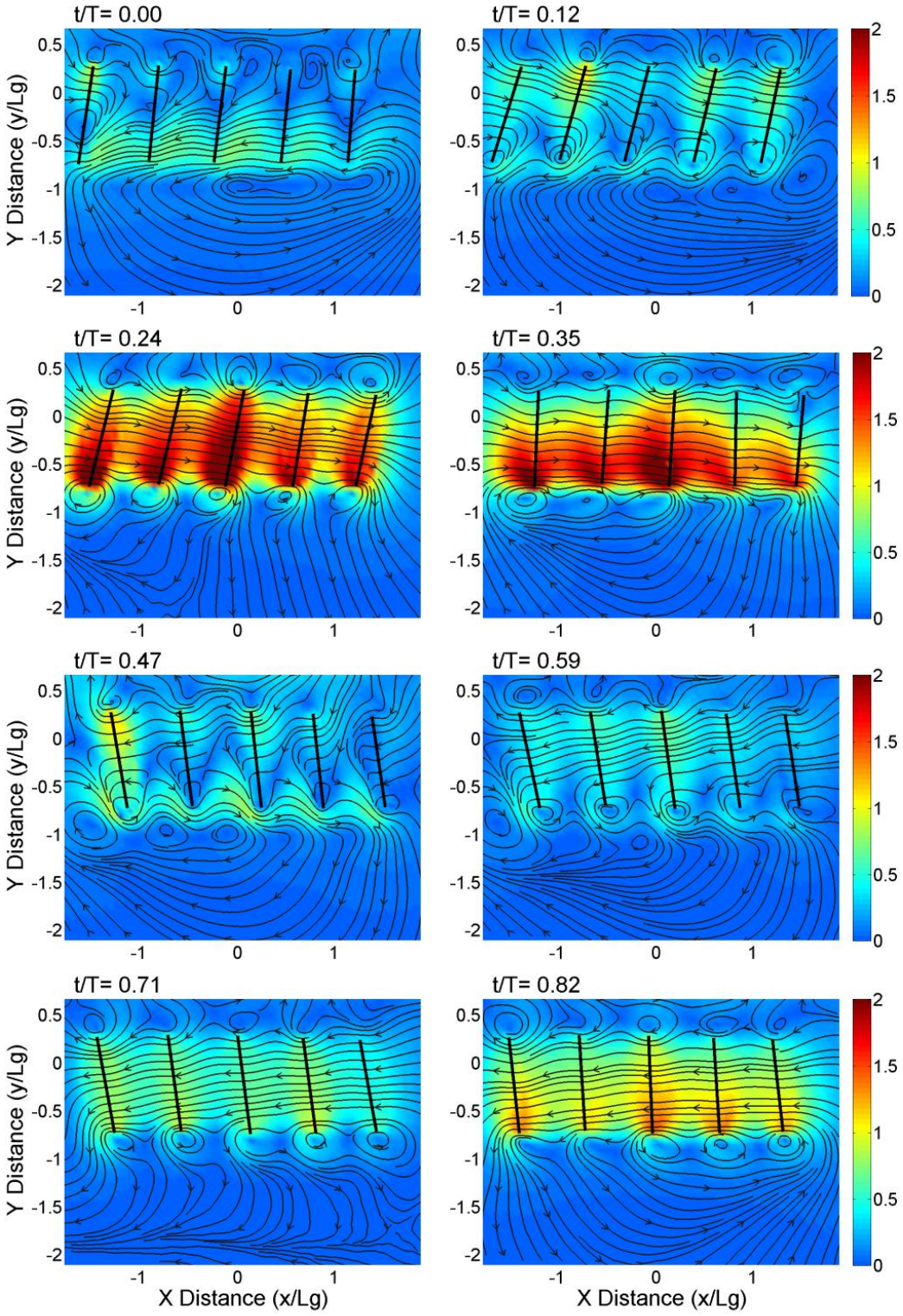

Figure 4.8: For ΔΦ =0º, the plots above are colored with the normalized velocity magnitude and show streamlines of the x and y velocity components for eight different times throughout the cycle. The origin is at the root of the third gill. The black lines represent the gill locations at each phase.

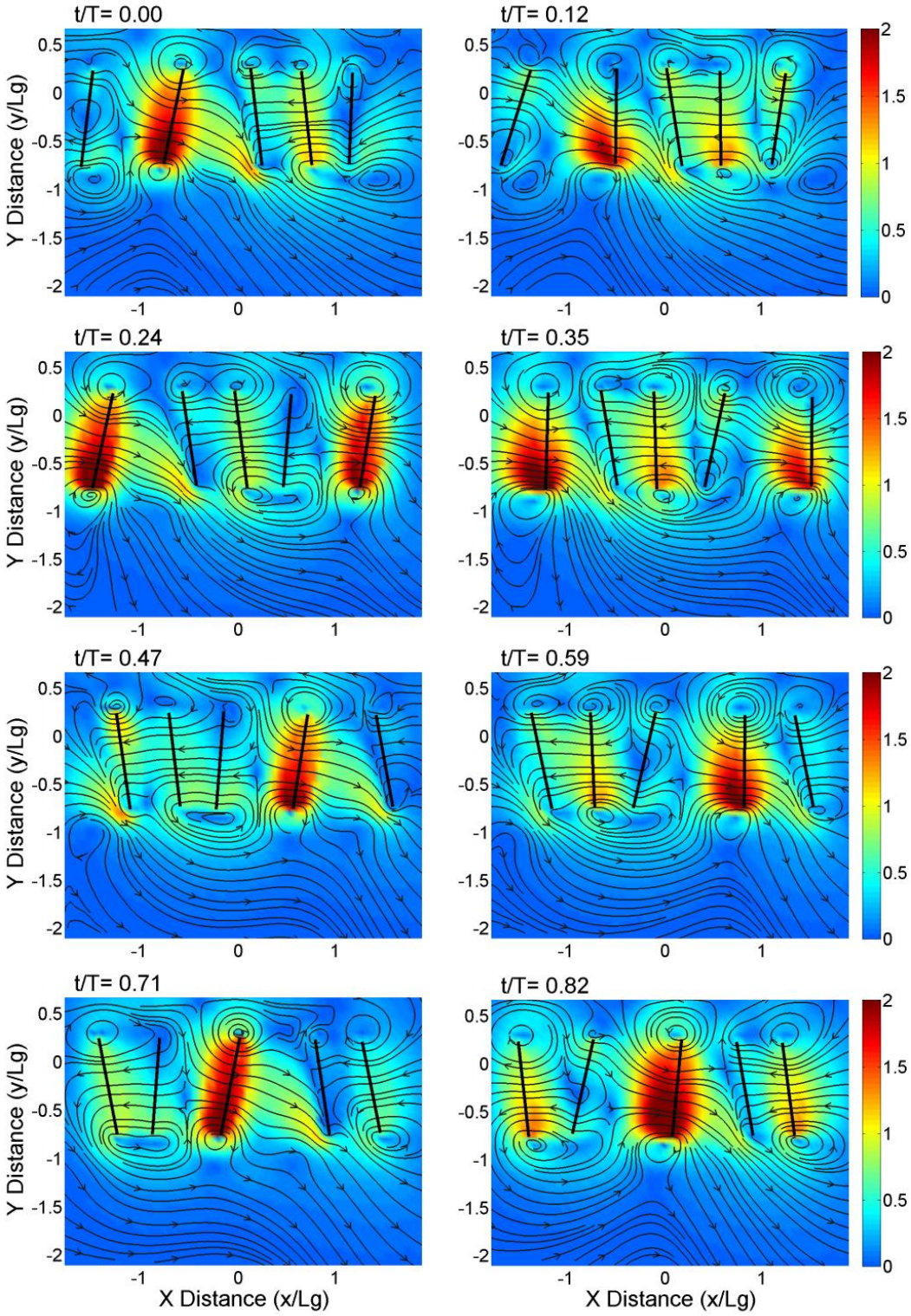

Figure 4.9: For ΔΦ =90º, the plots above are colored with the normalized velocity and show streamlines of the x and y velocity components for eight different times throughout the cycle. The origin is at the root of the third gill. The black lines represent the gill locations at each phase.

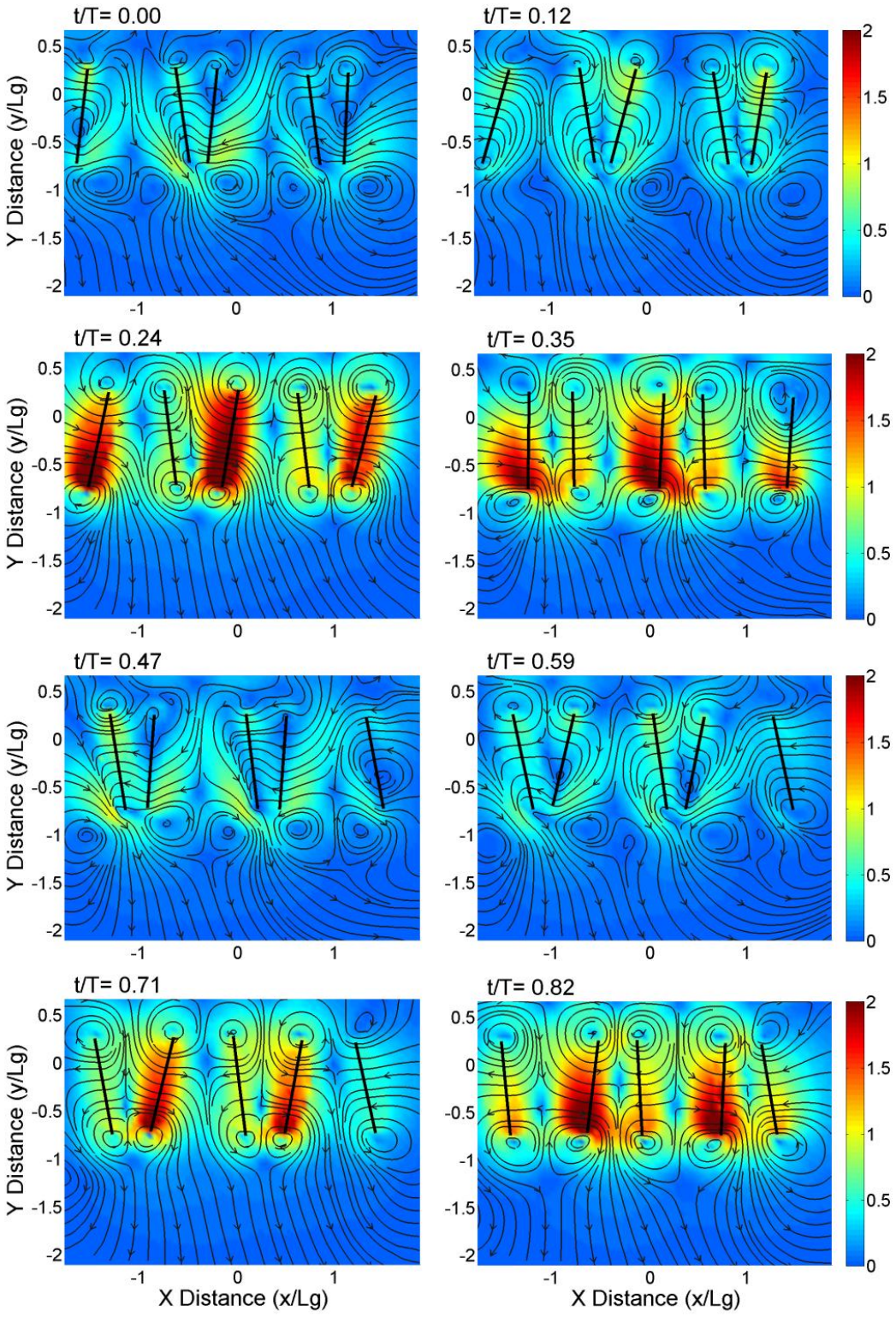

Figure 4.10: For ΔΦ =180º, the plots above are colored with the normalized velocity and show streamlines of the x and y velocity components for eight different times throughout the cycle. The origin is at the root of the third gill. The black lines represent the gill locations at each phase.

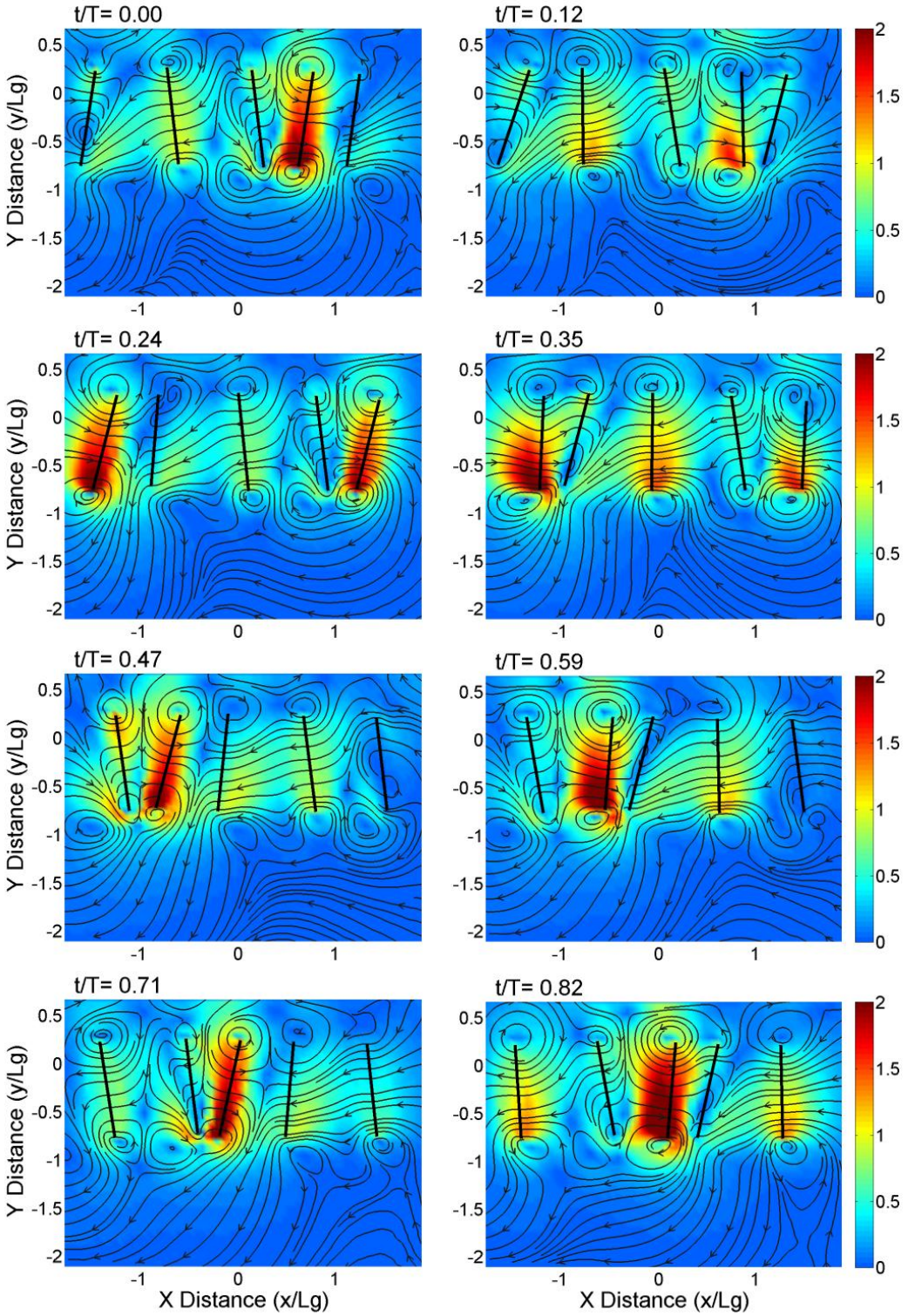

Figure 4.11: For ΔΦ =270º, the plots above are colored with the normalized velocity and show streamlines of the x and y velocity components for eight different times throughout the cycle. The origin is at the root of the third gill. The black lines represent the gill locations at each phase.

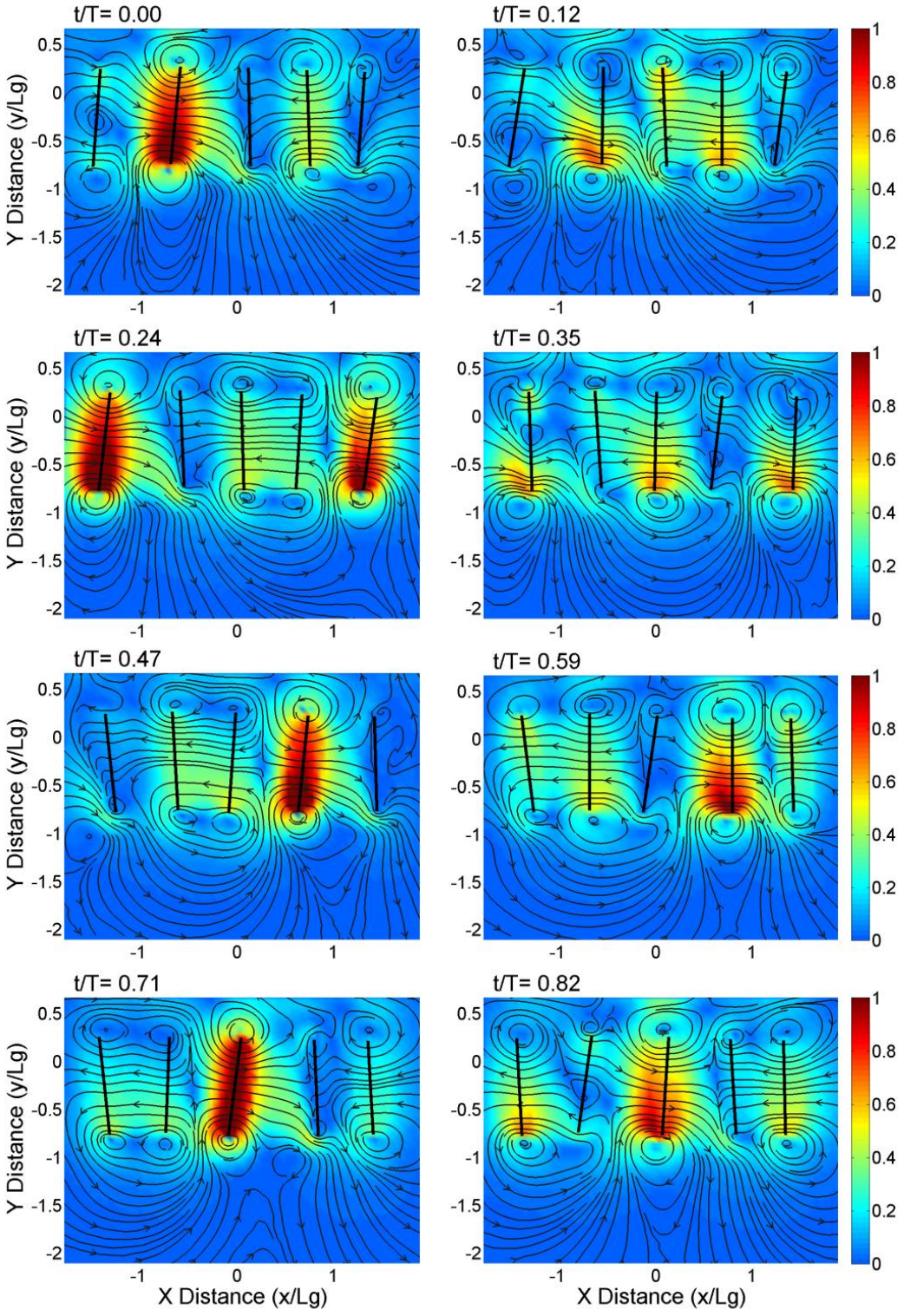

Figure 4.12: For ΔΦ =90º, but with smaller stroke and pitch amplitudes, the plots above are colored with the normalized velocity and show streamlines of the x and y velocity components for eight different times throughout the cycle. The origin is at the root of the third gill. The black lines represent the gill locations at each phase.

4.2.2 Mean Flow

The mean flow for each test case was calculated by averaging the velocity components over the cycle using data from seventeen phase angles throughout the cycle. The resulting flow fields are shown in Figure 4.14 through Figure 4.18. For the first test case, which uses a phase lag of 0º (synchronous motion), the flow in the *x*-*y* plane moves primarily laterally away from the array axis, with part of the flow re-circulating in the anterior (-*x*) region, while the other part of the flow re-circulates in the posterior (+*x*) direction. In the *x*-*z* plane, the flow comes from the anterior, posterior, and ventral (-*z*) side of the body and moves out towards the dorsal (+*z*) side. Again the areas of recirculation can be seen in the posterior and anterior direction. Because of the large recirculation seen in the *x*-*y* plane for many of the cases, whether the fluid is moving in or out of the posterior or anterior directions in the *x*-*z* plane is dependent on the location of the plane. The same is true for the location of the *x*-*y* plane. This first case has lower average velocities than the other three phase lags tested. This occurs because adjacent vortices do not get as close to each other, so the maximum induced velocity at each point in time is caused by a single vortex instead of two opposing vortices moving towards each other.

For the second test case ($\Delta\Phi = 90^{o}$), the *x*-*y* plane shows that the flow predominantly enters from the anterior (-*x*) direction and moves away from the centerline in the posterior (+*x*) direction. There is an area where the flow re-circulates in the posterior direction. The *x*-*z* plane shows the fluid moving from the anterior, posterior and ventral sides in the dorsal direction, with a bias in the posterior direction. An area of recirculation can be seen again on the posterior side. The

highest average velocities for this case occur to the anterior (-*x*) of the mean gill positions. It can also be seen that the area of higher positive vorticity stretches slightly along the gill.

For the third case ($\Delta\Phi = 180^o$), the *x*-*y* plane shows the flow moving away from the centerline with some of it re-circulating in both the anterior and posterior directions. The *x*-*z* plane shows flow coming in from the anterior, posterior and dorsal sides and flowing out the ventral side. While an inter-gill phase lag of 180^o creates symmetry in the stroke motion with respect to adjacent gills, the kinematics of the stroke program is asymmetric, making the area of peak average velocity also asymmetric around the average gill location. For this case, the highest average velocities occur to the left of the average gill positions. This asymmetry is also what causes the fluid to move slightly more towards the posterior (+*x*) direction instead of directly away from the centerline, as one might expect for a purely reciprocal motion.

In the fourth case ($\Delta\Phi = 270^o$), the *x*-*y* plane shows the fluid coming in from the posterior direction and flowing away from the centerline in the anterior direction. There is an area where the flow re-circulates in the anterior direction. The *x*-*z* plane shows the fluid coming in from the posterior, anterior, and dorsal directions, and flowing in the ventral direction. This is similar to the trend seen in the 90^o phase lag case, except in the anterior (+*x*) direction rather than the posterior direction. This is expected since the direction of the wave propagation is reversed in the two cases. For this case, the fluid between the gills diverges as it moves away from the centerline. Instead of being unidirectional, part of it moves to the left and part moves to right. When it reaches an adjacent gill, it is redirected, and when the fluid exits the area

between the gills, it all moves away from the centerline (-*y*) in the anterior (-*x*) direction. The areas with the highest average velocities in this case are located to the right of the average gill positions. The average gills are also closer to the areas of negative vorticity than to the areas of positive vorticity.

The fifth case, which uses a phase lag of 90º except with smaller stroke and pitch amplitudes than the previous case, has flow in the *x*-*y* plane coming in from the anterior direction and flowing away from the centerline in the posterior direction. This average flow pattern is similar to the previous case that uses the same phase lag, except the magnitudes of the velocities and vortices are much lower. This is expected because the gills are traveling a smaller distance in the same amount of time.

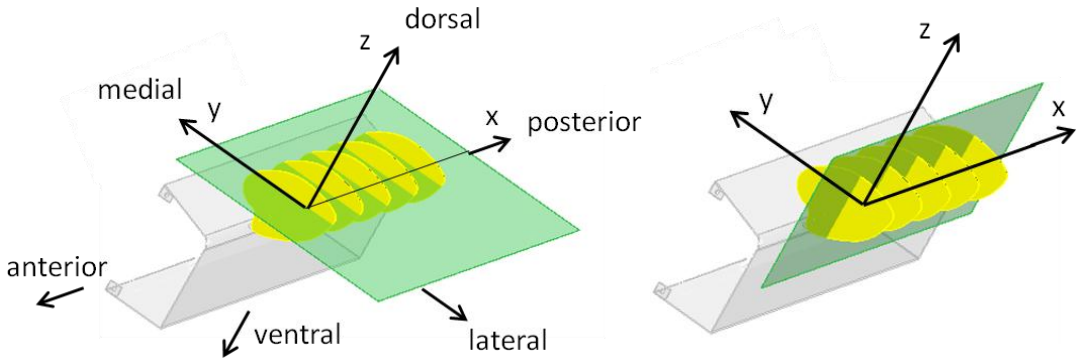

Figure 4.13: The left figure shows the location of the XY plane that the mean flow data is shown for. The right figure shows the location of the XZ plane that the mean flow data is shown for. The origin is located at the root of the central gill.

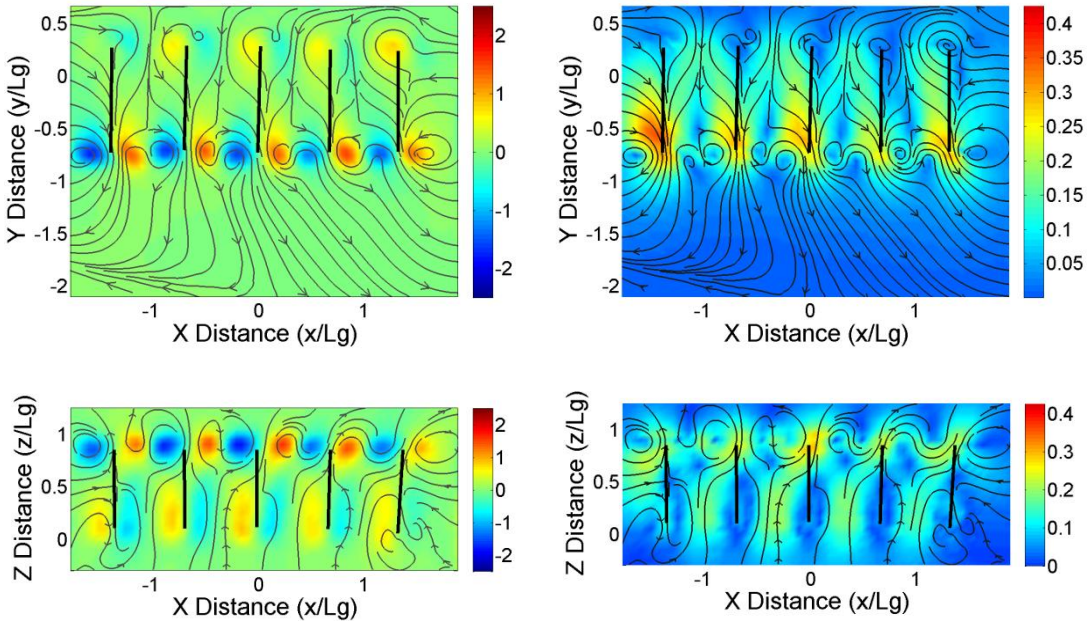

Figure 4.14: Mean flow for ΔΦ=0º. The left two plots show planes in the XY and XZ directions with streamlines and non-dimensionalized vorticity in the background. The right shows plots for the same planes, but with non-dimensionalized velocity in the background.

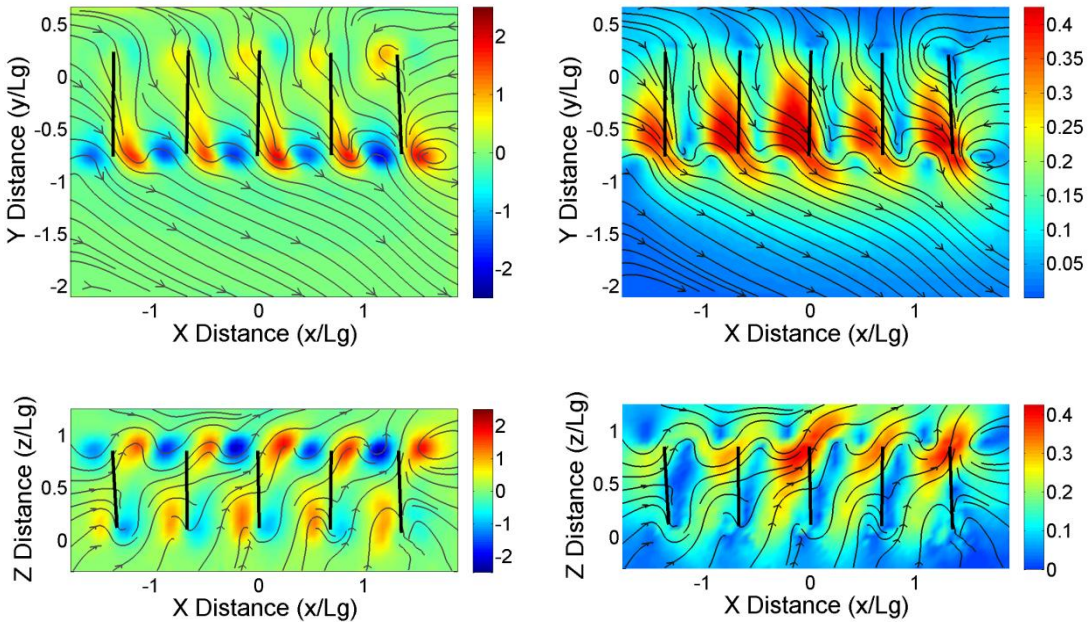

Figure 4.15: Mean flow for ΔΦ =90º. The left two plots show planes in the XY and XZ directions with streamlines and non-dimensionalized vorticity in the background. The right shows plots for the same planes, but with non-dimensionalized velocity in the background.

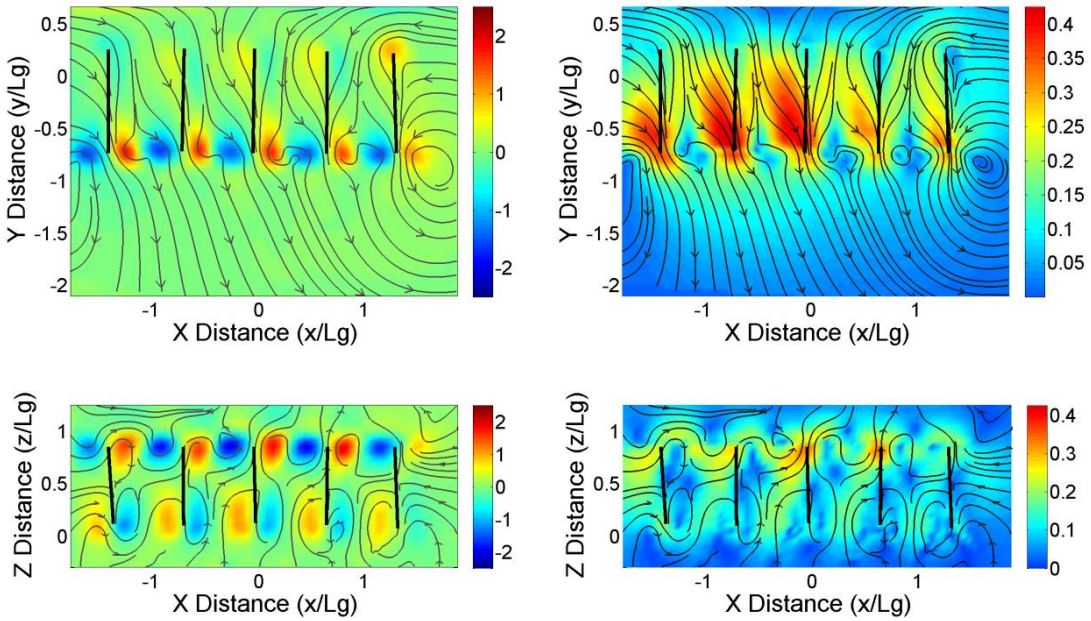

Figure 4.16: Mean flow for ΔΦ =180º. The left two plots show planes in the XY and XZ directions with streamlines and non-dimensionalized vorticity in the background. The right shows plots for the same planes, but with non-dimensionalized velocity in the background.

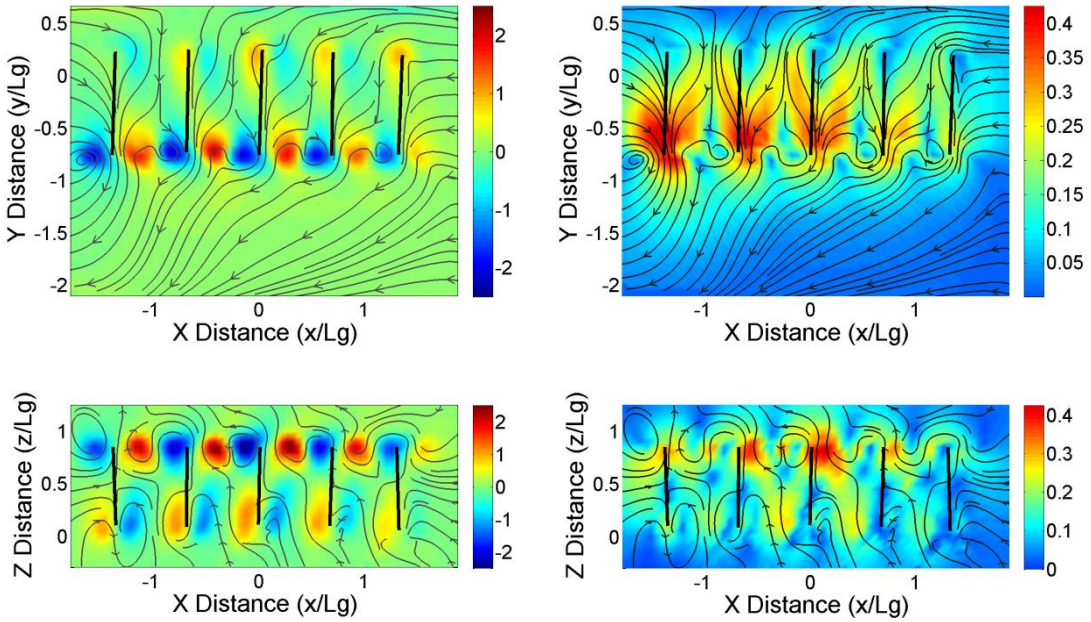

Figure 4.17: Mean flow for ΔΦ =270º. The left two plots show planes in the XY and XZ directions with streamlines and non-dimensionalized vorticity in the background. The right shows plots for the same planes, but with non-dimensionalized velocity in the background.

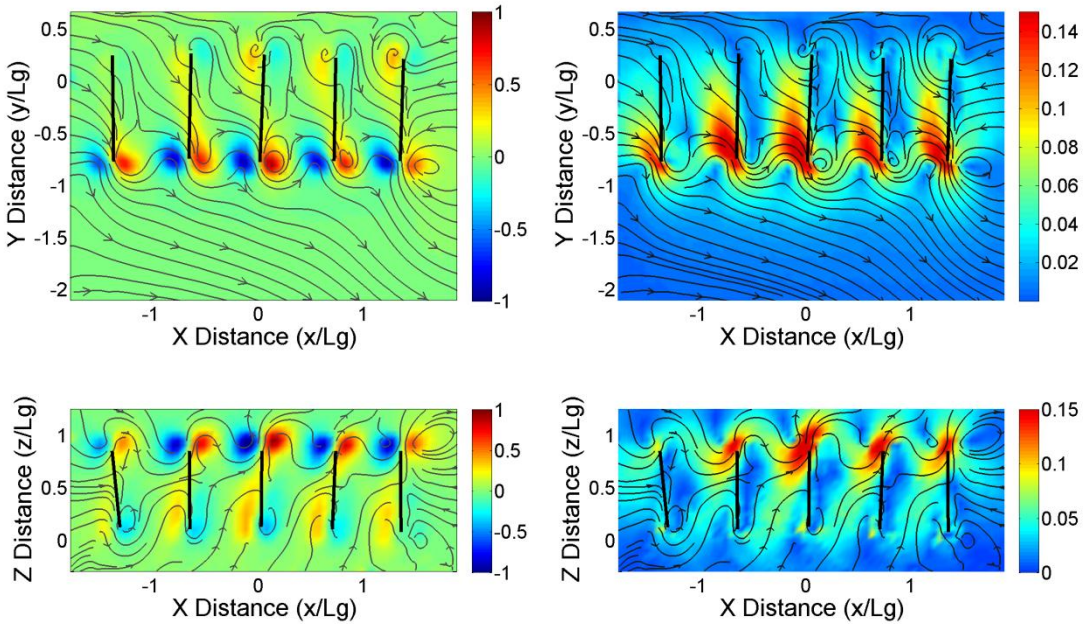

Figure 4.18: Mean flow for ΔΦ =90º, using half the stroke and pitch amplitude as used the previous cases. The left two plots show planes in the XY and XZ directions with streamlines and non-dimensionalized vorticity in the background. The right shows plots for the same planes, but with non-dimensionalized velocity in the background

4.3 Quantitative Parameters

4.3.1 Flux

In order to determine the pumping efficiency of the gill array, it is necessary to know how much fluid the array is moving. To determine this, a control volume was constructed around the array, seen in Figure 4.19, and the average amount of fluid flowing in and out of each surface on the volume was calculated. This was done by multiplying the time-averaged out-of-plane velocity components on each surface by the area surrounding each vector, then summing all of the flow moving in and all of the flow moving out. In the equation below, Q_{in} represents the volume of flow going into one side of the control volume. The second equation sums the flow

moving in for all six sides, giving the total volume of fluid moving into the control volume (total_ Q_{in}).

$$Q_{in} = \iint \overline{U}_{k_{in}} dx_i dx_j \cong \sum_{k_{in}=1}^{N_{in}} \overline{U}_{k_{in}} \delta x_i \delta x_j$$

i, j and k represent the directions relevant to the surface on the volume being analyzed. i and j represent the components in this plane, while k represent the out-of plane component.

$$total\_Q_{in} = \sum_{l=1}^{N_{sides}=6} Q_{in-l}$$

The results were non-dimensionalized by dividing the volumetric flow rate by L_g^3 f where L_g is the gill length, 0.04 m, and f is the frequency, 1.85 Hz (see Table 4.1). The sum of the flow in and out of the volume should equal zero, but due to the error in the measurement, the total is greater than this. The total uncertainty of the measurement was propagated through the flux calculations using the standard error from the variation in the flow field calculate from the ensemble of 30 images acquired at a single phase in the cycle. This error is caused by the slight variation in the period of each gill from cycle to cycle. The values of the total propagated uncertainty for each test case are approximately the same or greater than the sum of the flow going in and out of the control volume for each case. This propagation, found in Appendix C,

results in an uncertainty of about ±1% of the average of the total flow in and out of the control volume.

The amount of fluid that the array in each test case pumps is given by the average of the magnitudes of the total flow in and the total flow out. Comparing this value for the different test cases, the second test case, which uses a phase lag of 90º, has the highest value. The amount pumped for the 180º phase lag and the 270º phase lag cases respectively are about 4% and 6% smaller than the value for the 90º case. The amount pumped by the 0º phase lag case on the other hand is about 40% smaller than the value for the 90º phase lag case. The last case, which uses a phase lag of 90º but has smaller stroke and pitch amplitudes, results in the lowest amount of fluid pumped over a cycle giving a value approximately 82% smaller than the amount pumping in the first 90º phase lag case.

Figure 4.20 through Figure 4.24 provide three dimensional visualizations for each test case of where fluid is moving in and out of the control volume on average over a cycle. For the first case shown with the 0º phase lag, the most pumping for this case can be seen moving in and out of surface two, which is the dorsal ($+z$) side of the mayfly. This case had the lowest total pumping, which can be attributed to the lack of beneficial fluid interactions between adjacent gills due to the synchronous motion. For this case, the gills spend the first third of the cycle moving the fluid in the posterior direction then the next two thirds of the cycle are then spent moving the fluid back in the anterior direction. This causes flow patterns at the same location in the control volume to be directed in during part of the cycle and out during the rest of

the cycle, resulting low net flux. This case also had the least vortex interaction, which resulted to the lowest velocities of any of the cases, leading to lower pumping.

The second case, which uses a phase lag of 90º, draws the majority of its flow in through surfaces 2 and 3 (the anterior ($+x$) and dorsal ($+z$) directions) and forces flow out of surfaces 2, 4 and 5, in the posterior and dorsal directions away from the centerline of the body. In this case, the power stroke was in the opposite direction of the wave propagation, which allowed the gill to produce a power stroke largely unimpeded from neighboring gill motion. The close proximity of the positive and negative vortices on adjacent gills also caused high velocities in the direction away from the centerline of the body ($-y$). Throughout the entire cycle, the flow further from the gills is always moving approximately in the posterior($+x$) direction away from the centerline of the body ($-y$). These characteristics of the flow field over a cycle give high average fluxes.

The third case uses a phase lag of 180º, which primarily draws fluid in through surfaces 2 and 3 and out through surfaces 1, 2 and 5. This pattern shows that fluid is drawn in closest to the body and is forced out in directions away from the body. This phase lag produced the second highest flux. For this case, the power stroke is relatively isolated for part of its stroke and directly interacting with an adjacent gill moving in the opposite direction for the other part of its stroke. This allows a relatively large volume of high velocity fluid to move with the power stroke, and then be redirected by the adjacent gill in the direction away from the centerline of the body. The repetition of this pattern throughout the cycle causes the flow further

from the gills to consistently move away from the body, causing high outward flux in these directions.

The fourth case uses a phase lag of 270º, which causes the wave to propagate in the same direction as the power stroke of the individual gills. For this case, the majority of the fluid moves into the control volume through surfaces 2 and 3 and out of surfaces 2 and 5 towards the anterior direction. This case varies from the previous cases, because the retracting gill is not isolated, but is in close proximity to an adjacent gill that is at a velocity close to zero. This case shows that the flow is consistently moving away from the body and in the anterior direction throughout the entire cycle, but the interaction effects from this phase lag result in slightly lower average velocities than the case which uses a phase lag of 90º, resulting in slightly lower overall pumping.

The last case, which has a phase lag of 90º but uses stroke and pitch amplitudes that are half of the values used in earlier cases, shows a similar flow pattern to the earlier 90º case. The majority of the fluid is drawn in through surfaces 2 and3 and forced out through surfaces 2, 4 and 5. The amount of fluid forced out through surface 2 is about 5 times as much as what goes out of surfaces 4 and 5. In the previous case 90º case, more fluid went out of surface 2 than 4 or 5, but the amount was only larger by a factor of 2. Since the gills in this case are traveling half of the distance in the same amount of time, much lower velocity flow is generated. The vortices on the ends of the gills also do not get as close to each other as in the previous case, causing less fluid interaction. These factors both contribute to the lower flux value calculated for this test case.

Table 4.1: The non-dimensionalized volumetric flow rate going in and out for six sides of a control volume for the five different test cases. These were nondimensionalized by dividing by $f*L_{g,,}$ where f is the frequency of the cycle, 1.85 Hz, and L_g is the length of the gill 0.04 m. The test cases are distinguished by their phase lag, with s90 representing the case with the smaller amplitude stroke and pitch.

Phase Lag (ΔΦ)	0º		90º		180º		270º		s90º	
	In	Out	In	Out	In	Out	In	Out	In	Out
Surface 1	-9.26E-02	3.31E-02	-9.93E-02	4.26E-02	-3.77E-02	9.15E-02	-9.88E-02	5.46E-02	-1.52E-02	8.55E-03
Surface 2	-1.33E-01	1.64E-01	-2.55E-01	2.77E-01	-2.90E-01	2.35E-01	-2.50E-01	2.44E-01	-3.72E-02	7.22E-02
Surface 3	-7.81E-02	5.03E-02	-1.58E-01	0.00E+00	-1.51E-01	8.72E-03	-1.18E-01	6.48E-02	-3.79E-02	4.45E-06
Surface 4	-3.64E-02	4.10E-02	-7.57E-02	1.09E-01	-8.50E-02	2.79E-02	-9.35E-02	3.07E-06	-1.55E-02	1.38E-02
Surface 5	-9.43E-03	7.13E-02	-5.71E-04	1.78E-01	-4.60E-03	2.13E-01	-2.27E-03	1.97E-01	-5.27E-04	1.52E-02
Surface 6	-1.30E-02	8.62E-03	-1.46E-02	2.62E-03	-2.27E-02	3.56E-03	-1.35E-02	5.23E-03	-1.25E-03	3.61E-03
Total:	-0.362	0.368	-0.604	0.609	-0.591	0.580	-0.576	0.565	-0.107	0.113
Average magnitude of flow In and Out	0.365		0.606		0.585		0.570		0.110	

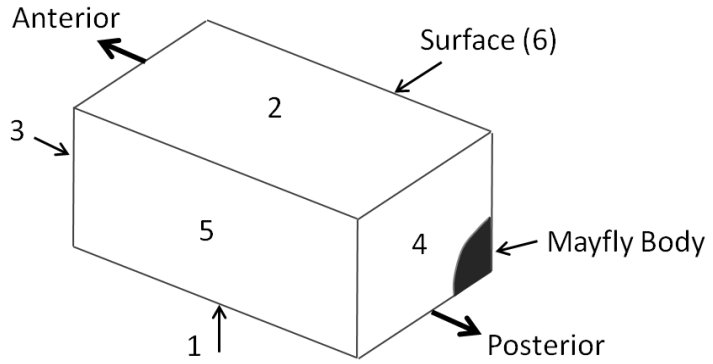

Figure 4.19: Control volume around the array of gills.

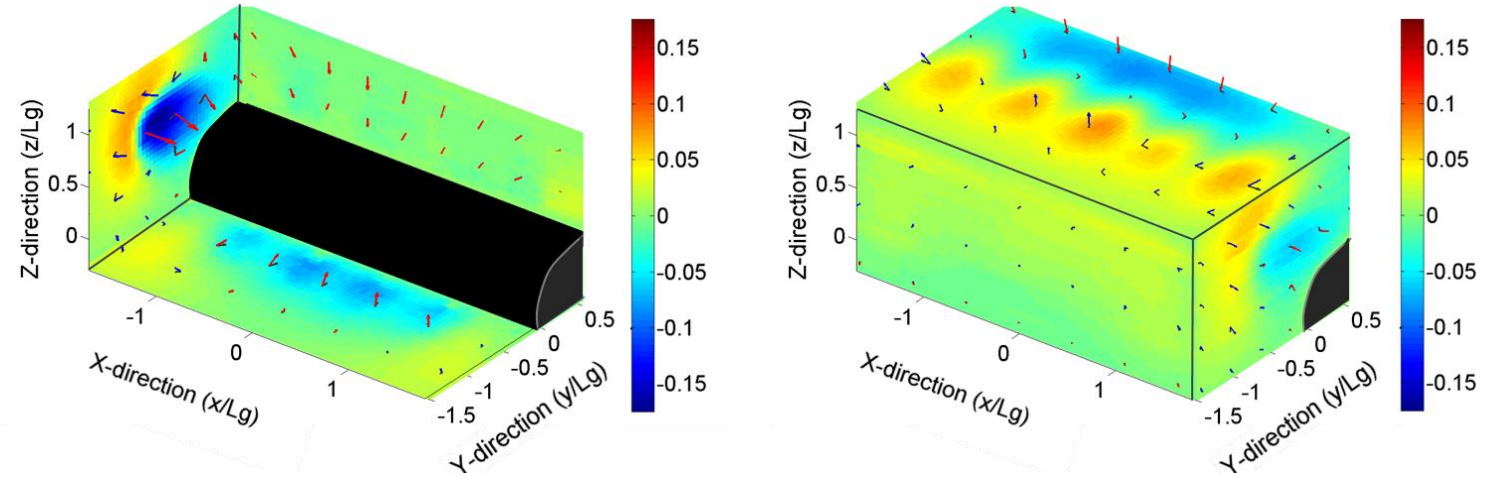

Figure 4.20: Flux control volume for ΔΦ=0º. The non-dimensionalized out-of-plane velocity component is colored where negative is flow in and positive is flow out as indicated by the arrows.

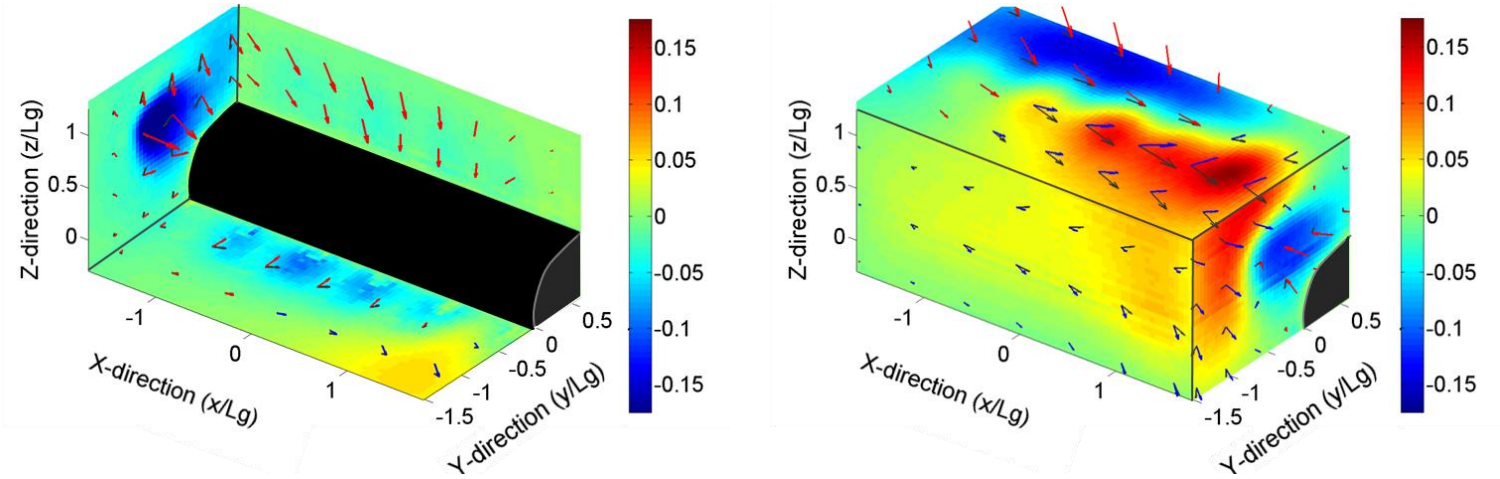

Figure 4.21: Flux control volume for ΔΦ =90º. The non-dimensionalized out-of-plane velocity component is colored where negative is flow in and positive is flow out as indicated by the arrows.

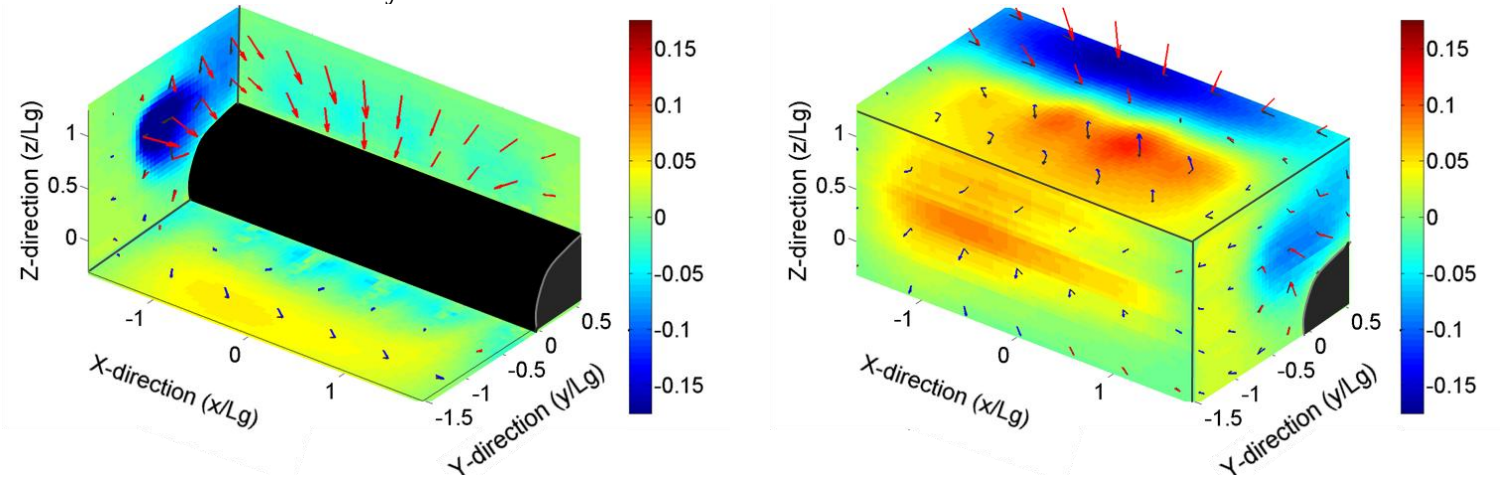

Figure 4.22: Flux control volume for ΔΦ =180º. The non-dimensionalized out-of-plane velocity component is colored where negative is flow in and positive is flow out as indicated by the arrows.

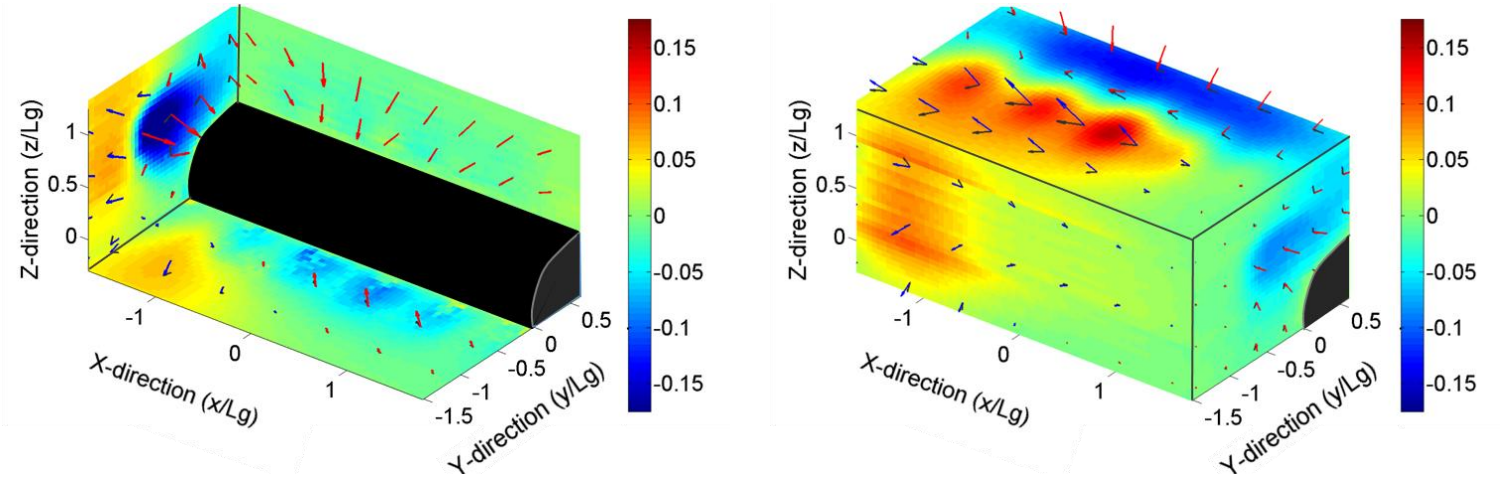

Figure 4.23: Flux control volume for ΔΦ =270º. The non-dimensionalized out-of-plane velocity component is colored where negative is flow in and positive is flow out as indicated by the arrows.

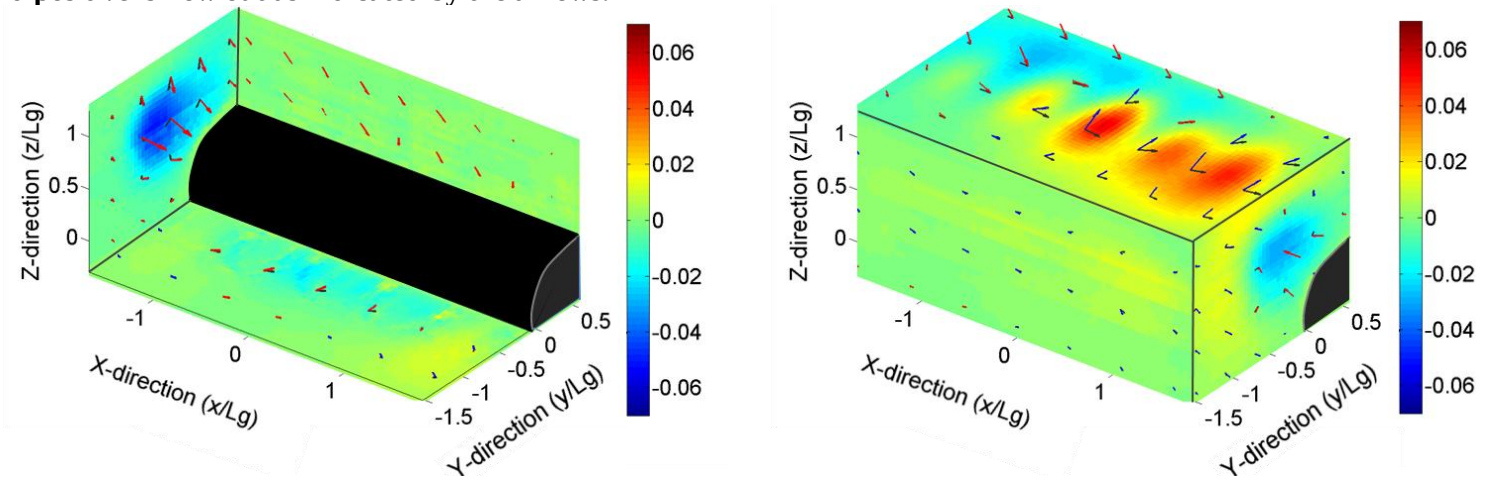

Figure 4.24: Flux control volume for ΔΦ =90º, but with half the stroke and pitch amplitudes. The non-dimensionalized out-of-plane velocity component is colored where negative is flow in and positive is flow out as indicated by the arrows

4.3.2 Dissipation

The amount of dissipation in a fluid system is the amount of energy lost due to irreversible viscous shear work done on the fluid, and is calculated by:

$$\varphi = \mu\left[2\left(\frac{\delta U}{\delta x}\right)^2 + 2\left(\frac{\delta V}{\delta y}\right)^2 + 2\left(\frac{\delta W}{\delta z}\right)^2 + \left(\frac{\delta V}{\delta x} + \frac{\delta U}{\delta y}\right)^2 + \left(\frac{\delta W}{\delta y} + \frac{\delta V}{\delta z}\right)^2 + \left(\frac{\delta U}{\delta z} + \frac{\delta W}{\delta x}\right)^2\right]$$

μ = dynamic viscosity

$U, V \text{ and } W$ = three measured velocity components

$\delta x, \delta y \text{ and } \delta z$ =the distance between each velocity vector

Due to the relatively small inertia of the fluid in these flows, the energy imparted to the fluid by the gill plate motion is dissipated within a short distance of the gill surface. In a companion study to the work of Sensenig et al (2010), direct numerical simulations of a mayfly gill array confirm that the average work done by the mayfly gills over a cycle is within 2% of the total dissipation of energy over a complete cycle in a volume within 1 gill length of the array boundary (K. Abdelaziz et al, personal communication, March, 2011). Using this information, it is possible to compare the average amount of work done by the gill arrays for each test case using the dissipation within the control volume as an equivalent surrogate for the work performed by the gills.

The total dissipation was calculated using the three measured components of the velocity over the same control volume as was used for the flux calculations. The total dissipation at each of the phases throughout the cycle was then divided by the number of phases, giving the cycle-averaged dissipation for each test case. The resulting average dissipations were non-dimensionalized by $\rho f^3 L_g^5$, where ρ is the density, f is the frequency of the cycle, and L_g is the gill length. These results are listed in Table 4.2.

Table 4.2: Non-dimensionalized average dissipation in the control volume for the five test cases. These values were non-dimensionalized $\rho f^3 L_g^5$. The test cases are distinguished by their phase lag, with s90º representing the case with the smaller amplitude stroke and pitch.

Phase Lag ($\Delta\Phi$)	0º	90º	180º	270º	s90º
Cycle-Averaged Dissipation $\phi / \rho f^3 L_g^5$	2.699	4.518	5.437	5.000	0.990

The uncertainty in the dissipation was calculated using the uncertainty in the measurement velocities as described in Appendix C. This resulted in uncertainty values of approximately ±0.001 or ±0.03% of the dissipation. This uncertainty seems optimistic, but at least reflects that the uncertainty is in an acceptable range. Comparing the time averages of the dissipation shows that of the first four test cases, the gills in the 0º phase lag case did the least amount of work. The 180º degree test case required the most work followed by the 270º test case. The 180º degree case required about 20% more work than the 90º case, and the 270º degree case required about 10% more work than the 90º case. The last test case, which had a 90º phase lag, but used stroke and pitch ranges that were half the amplitude of the previous cases, required the least amount of work.

Figure 4.25 through Figure 4.29 show where the dissipation occurs throughout the cycle for each of the different test cases. For the 0º phase lag case, the highest dissipation occurs on the tip of the gill furthest from the body during the retraction stroke. This is where the high velocities and velocity gradients occurs. Higher dissipation also occurs on the leading end gill when the array first starts protracting or retracting. Because adjacent gills are always moving in the same directions, and keep comparable distances from each other, dissipation is comparatively weak. For the 90º phase lag case, the highest areas of dissipation are once again seen near the tips of the gills furthest from the body. Because adjacent gills are moving at different velocities and are sometimes moving in opposite directions, the volume of high dissipation is larger on the retracting gill than it was in the 0º case. Due to the fact that the retracting gill is relatively far from the adjacent gills, the magnitude of this dissipation is smaller than what is seen in the 180º and 270º cases. There are also additional areas of moderate dissipation in between the slower moving gills during protraction, but the slower velocities help mitigate the magnitude in this part of the stroke. Although the slower velocities should logically produce smaller dissipation magnitudes, the duty cycle of the protraction is longer, and hence they should have a longer duration of contribution to the overall net dissipation than the retraction stroke. A more detailed analysis of the dissipation of a unit gill should be investigated to answer the relative contributions of the various phases of the stroke cycle.

The 180º phase lag case, which had the highest average dissipation, exhibits areas of extremely high dissipation in between gills that are moving at high speeds either towards or away from each other. The other portion of the cycle, when the gills

are moving at slower velocities and beginning to reverse the direction of their movement, exhibits much lower dissipation. For the 270º phase lag case, the retracting gill, which is moving at the highest velocity, is relatively close to adjacent gills. The highest areas of dissipation occur around the retracting gill when it is moving away from the protracting gill to its left and when it is approaching the gill to its right, which is protracting or in the process of reversing the direction of its movement. The close proximity of these gills causes higher volumes and magnitudes of dissipation than was exhibited for the 90º phase lag.

The last test case, which had a 90º phase lag, but used amplitude and stroke ranges that were half of the previous cases, shows similar dissipation patterns to the previous 90º case. The magnitudes of this case are much smaller, so the scale on Figure 4.29 was reduced so the dissipation patterns could still be seen. It shows that the highest areas of dissipation occur around the retracting gill caused by its high speed and the low speed of the adjacent gills.

These patterns show that the highest dissipation is caused by adjacent gills performing opposing motion when at least one of the gills is moving at a high velocity. The closer the gills are when this occurs, the higher the amount of dissipation. As reflected in the results, the lowest amount of dissipation will occur when there are no adjacent gills moving in opposite directions.

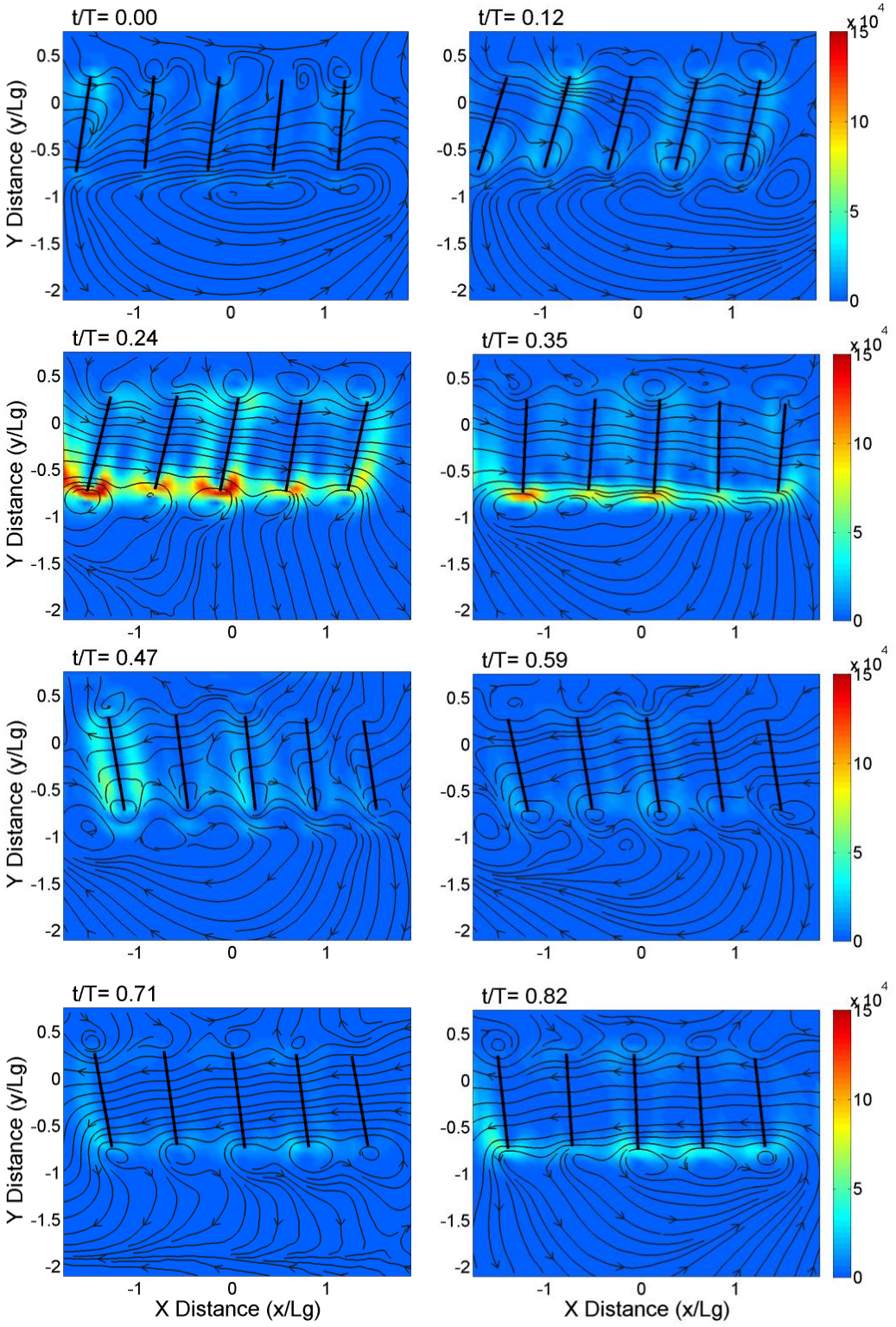

Figure 4.25: For ΔΦ=0º between the gills' movement, the plots above are colored with non-dimensionalized dissipation and show streamlines of the x and y velocity components.

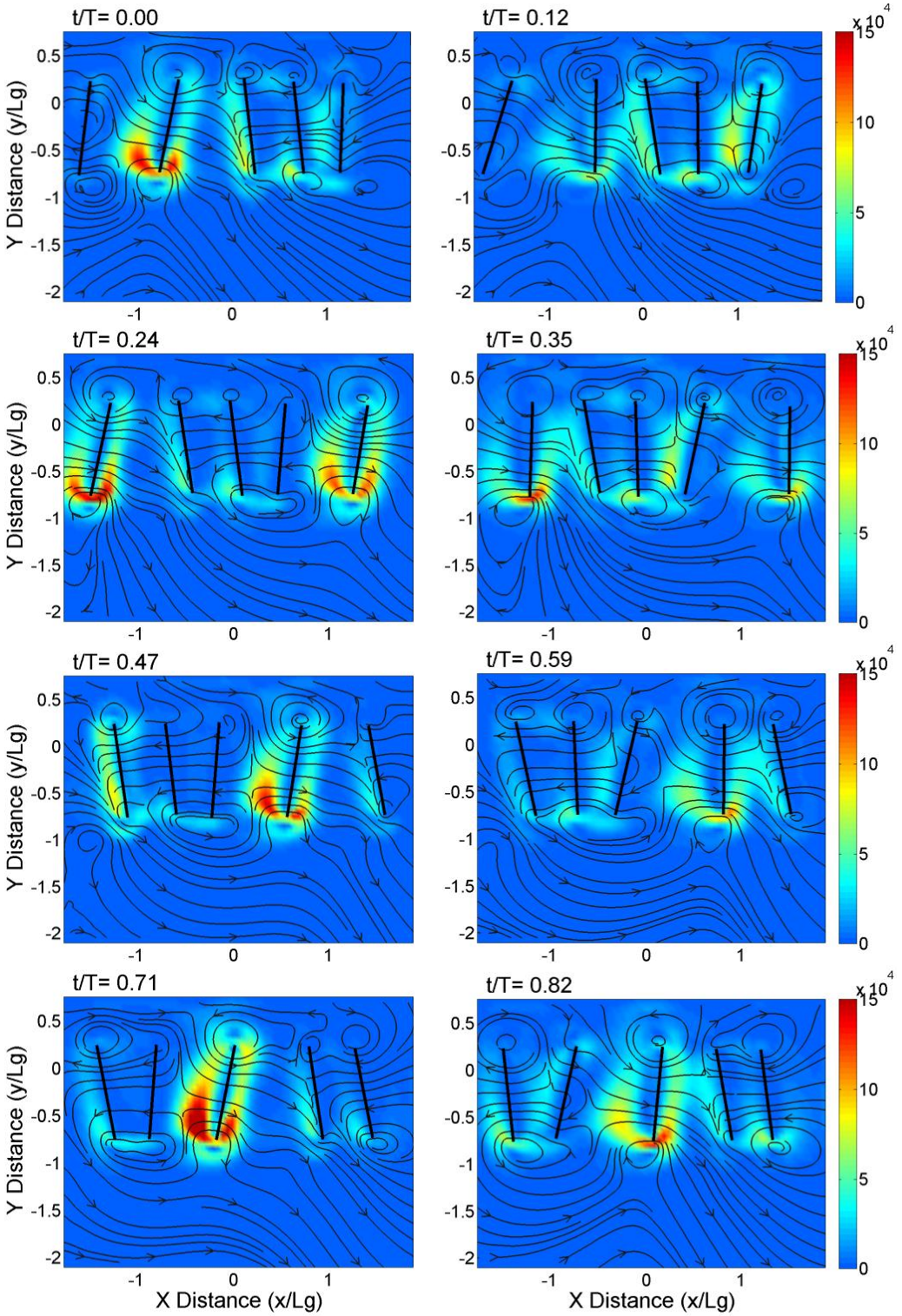

Figure 4.26: For ΔΦ =90º between the gills' movement, the plots above are colored with non-dimensionalized dissipation and show streamlines of the x and y velocity components.

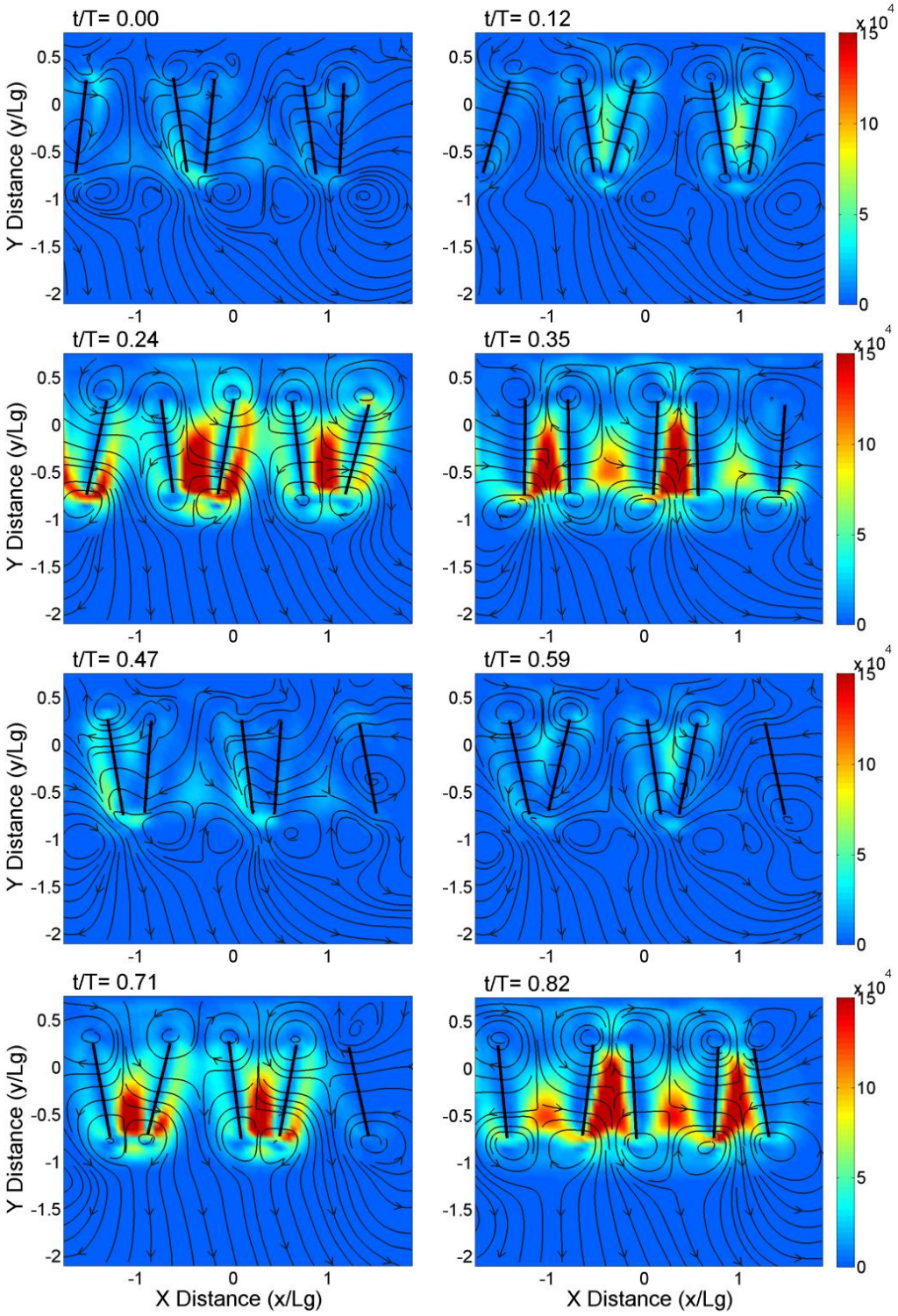

Figure 4.27: For ΔΦ =180º between the gills' movement, the plots above are colored with non-dimensionalized dissipation and show streamlines of the x and y velocity components.

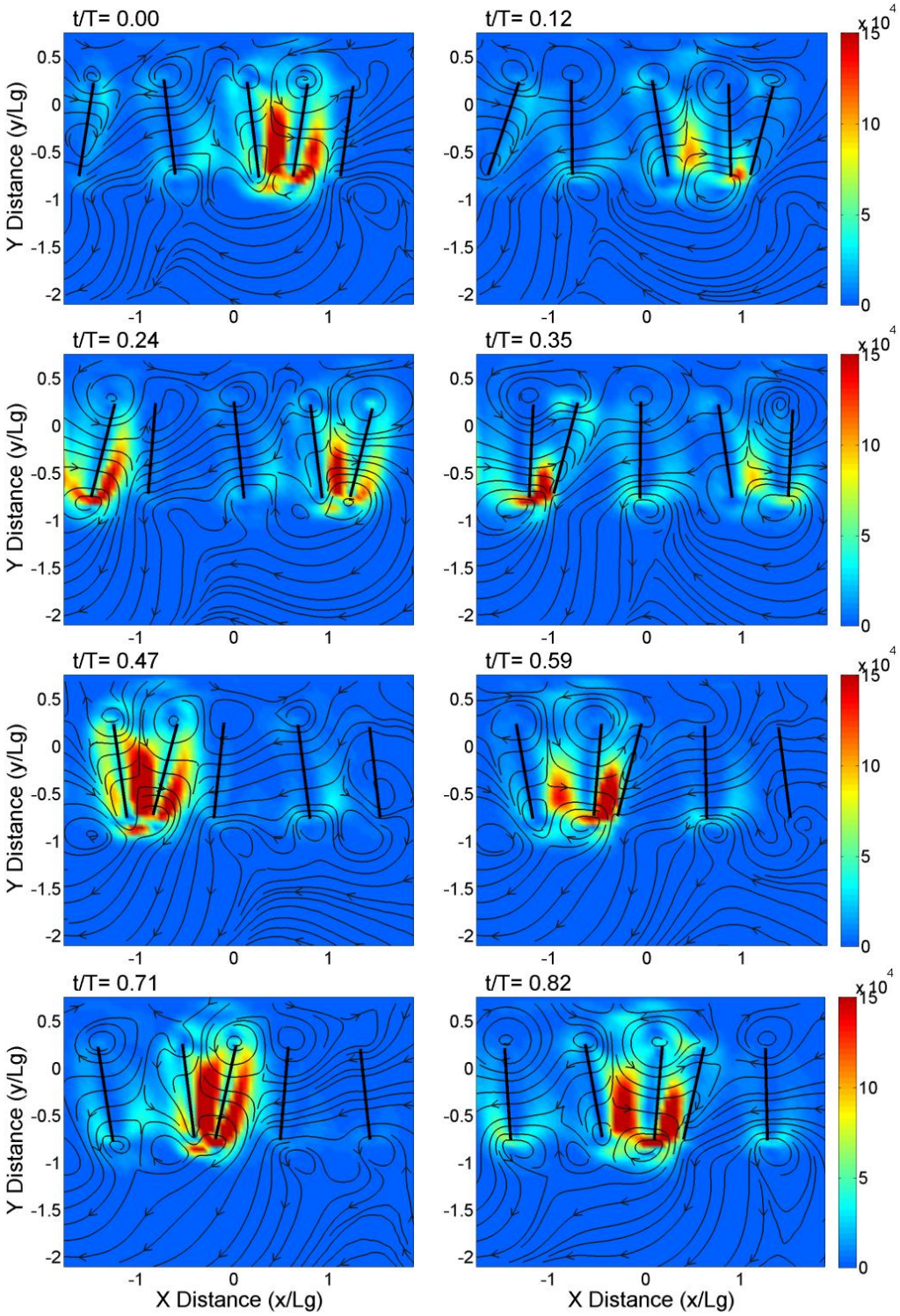

Figure 4.28: For ΔΦ =270º between the gills' movement, the plots above are colored with non-dimensionalized dissipation and show streamlines of the x and y velocity components.

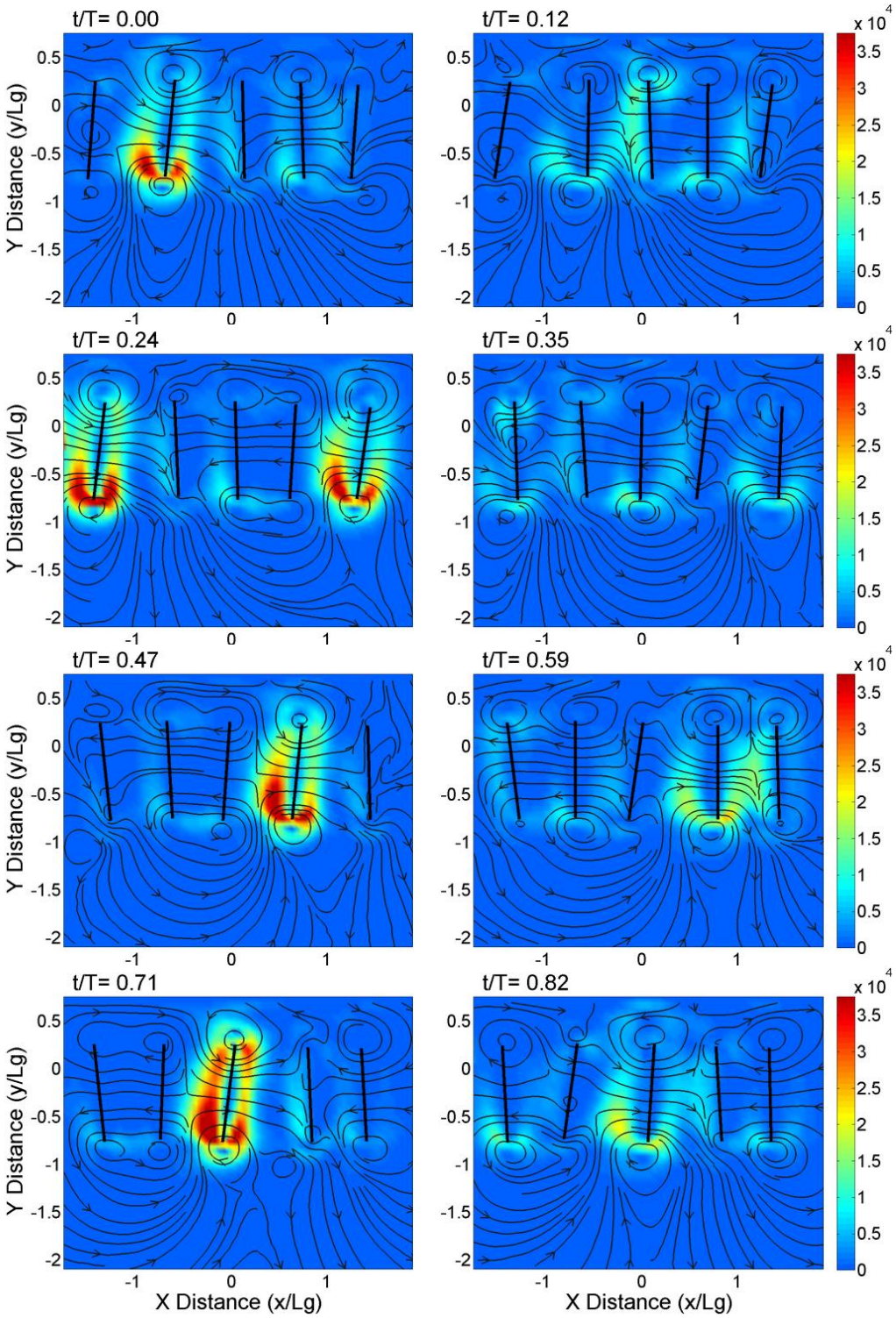

Figure 4.29: For ΔΦ =90º between the gills' movement, but with smaller amplitude stroke and pitch angles, the plots above are colored with non-dimensionalized dissipation and show streamlines of the x and y velocity components.

4.3.3 Efficiency

To determine which case exhibits the best overall performance, a metric to describe the "pumping efficiency" as a mass-specific volume flux is introduced. The mass-specific volume flux was calculated by dividing the net pumped volume flux by the average dissipation (representing the work done by the gills). Uncertainty propagation resulted in uncertainties in the efficiencies of ±0.001. The efficiencies calculated are listed in Table 4.3.

Table 4.3: Efficiency is calculated for each of the five test cases by dividing the cycle-averaged, non-dimensional flux by the cycle-averaged, non-dimensional dissipation. The test cases are distinguished by their phase lag, with s90º representing the case with the smaller amplitude stroke and pitch.

Phase Lag ($\Delta\Phi$)	0º	90º	180º	270º	s90º
Flux ($Q/ L_g^3 f$)	0.365	0.606	0.585	0.57	0.11
Dissipation ($\phi/ \rho f^3 L_g^5$)	2.699	4.518	5.437	5.000	0.990
Efficiency	0.135	0.134	0.108	0.114	0.111

This calculation reveals that the phase lags that yield the highest efficiency are 0º and 90º. With the given uncertainty, there is no statistical difference between the efficiencies for these two phase lags. The lowest efficiency, which is about 20% lower than the 90º phase lag efficiency, was produced by the case with the 180º phase lag. The efficiencies for the 270º phase lag case and the smaller amplitude 90º case are similar and are about 15% smaller than the efficiency for the 90º case.

It was hypothesized that the case with a phase lag of 90º would function with the highest pumping efficiency, as this motion is the most similar to that of the live mayfly. While this is true, a phase lag of 0º resulted in the same efficiency. A mayfly nymph uses its gills to circulate water around its body in order to maintain the necessary amount of oxygen in it that allows it to breathe. It is possible that the

mayfly uses a phase lag of 90º rather than 0º, because it is sometimes beneficial to circulate the water in the most time efficient manner, which would require a higher pumping rate. Since the amount that the array pumps with a phase lag of 0º is 40% lower than the amount pumped with the array that uses a 90º phase lag, the 90º phase lag would be more effective for this purpose.

4.3.4 Kinematic Limitations

In order for this experiment to operate with the same stroke and pitch ranges for each of the four phase lags, the stroke and pitch amplitudes had to be reduced from those used by the actual mayfly nymph. This was necessary, because phase lags of 180º and 270º caused adjacent gills and control linkages to collide when the full amplitudes were used. The most limiting phase lag was 270º, which caused the gills to collide when amplitudes above about 70% of those of the actual mayfly were used. The 180º phase lag was also limiting, and caused the gills to collide at amplitudes above about 82% of those of the actual mayfly. This was determined through trial and error with the robotic array of gills. Phase lags of 0º and 90º both allowed the full amplitudes to be used without any of the gills colliding.

While the kinematics in the experiment were simplified compared to those of the mayfly, and the gills were centered around a different angle, this still provides insight on the limitations that different phase lags might impose on the gill motion. If it is necessary for the mayfly nymph to be time efficient about pumping water around it in order to breathe, it may only be able to achieve this by using a high pumping rate that requires high amplitude motion. In this case, it would be necessary to use a

phase lag that allows for a large range of motion. Since a 0º phase lag would never cause the gills to collide, it has the largest versatility of stroke and pitch range. Because the pumping rate of the 90º phase lag is still much higher than that for a 0º phase lag, this might still be the necessary phase lag for time efficient pumping.

Chapter 5

Conclusion

As the size of devices continues to be reduced, traditional pumps used at high Reynolds numbers become inefficient. Many types of micropumps have been developed to meet specific needs, but a versatile pump that is effective with a variety of fluid properties, fluid volumes and flow rates has not emerged. A specific versatility challenge that arises is the development of a pump that is effective in the Reynolds number range from 1-100, which is relatively understudied. One approach to addressing this transitional Reynolds number regime is to observe how animals that function in this realm accommodate for this change in fluid interactions. For this reason, a specific animal of interest is the mayfly nymph (*Centroptilum triangulifer),* which uses an array of oscillating plates for ventilation in this range Reynolds number range. Investigating specific kinematic and geometric details of mayfly gills can provide insight on why the mayfly uses these particular kinematics and how these can be used effectively for an engineering device.

For the current experiment, a two-degree-of-freedom robotic oscillating plate array was constructed, which allowed for variations in the kinematics beyond what is exhibited by the animal. The specific properties that were varied and compared in this experiment are the phase lag between the gills and the stroke and pitch amplitudes. Stereo PIV was used for five different test cases to measure all three components of the unsteady velocity field over a three dimensional volume surrounding the array. For the first cases, data was taken using four different phase

lags: 0°, 90°, 180° and 270°, all with the same programmed stroke and pitch amplitude. For the fifth test case, a phase lag of 90° was used, but the amplitudes of the stroke and pitch were reduced to half of those used in the previous cases. The quantitative measurements taken for each case allowed for the net pumping rate, flow induced dissipation and a ratio of these two, representing a specific flux efficiency, to be directly computed. These quantities as well as qualitative observations, both detailed in Chapter 4, lead to the following conclusions.

1. Mayfly gill arrays are found in nature to use a phase lag of 90°. The measurements of the robotic mayfly array indicate that this phase angle may be preferable to larger phase delays (180° or 270°), due to a higher pumping efficiency. The pumping efficiency for $\Delta\Phi = 90°$ phase lag was found to be equivalent to the case of $\Delta\Phi = 0°$ phase lag, but the $\Delta\Phi = 90°$ produces a significantly higher pumping rate. Observations associated with this conclusion include:
    a. Phase lags that cause vortices on adjacent gills to rotate in opposite directions produce the highest pumping rates (exhibited for 90°, 180° and 270° phase lags).
    b. Phase lags that have slower protracting gills on both sides of the faster retracting gill, produce larger amounts of dissipation. As seen for phase lags of 180° and 270°, closer proximities of the protracting gills to the retracting gill and higher velocities at these points cause higher total dissipation.

c. Arrays that function with phase lags where the power stroke is relatively far from adjacent gills have the highest efficiencies (seen for phase lags of 0° and 90°).

2. The results from the case with a 90° phase lag, but lower stroke and pitch amplitudes imply that as the amplitudes are reduced, the net flux and dissipation production do not decrease at the same rate. This causes the efficiency to decrease as the amplitudes are decreased. Since the maximum velocity of the gills for this case is half of the maximum velocity of the gills for the other cases, the Reynolds number when defined by the maximum velocity instead of the frequency of the cycle has decreased. Since it has been observed with other pumps that the efficiency decreases with Reynolds number, this decrease in efficiency is not unexpected. To determine if there is a reliable trend between the stroke and pitch amplitudes and the efficiency, it would be necessary to perform a further study testing a wider range of stroke and pitch amplitudes.
3. Arrays with phase lags that allow larger stroke and pitch amplitudes to be reached are advantageous if pumping rates are required that necessitate higher amplitudes. Arrays with phase lags of 0° and 90° can reach higher stroke and pitch amplitudes than the 180° or 270° cases without adjacent gills colliding, potentially making them more advantageous

This study only begins to examine the influence of different kinematic parameters on the pumping efficiency and the hydrodynamic mechanisms that generate the net flow.

Much work still remains in examining the flow fields associated with kinematic and geometric variations of the mayfly gill array. Specific velocity measurements that would make the results of this study more comprehensive are for variations of the central angle and amplitudes of the stroke and pitch. Additional parameters that would contribute to determining how the array of gills could be optimized for pumping are the gill spacing, plate shape and the Reynolds number. A more specific application of the mayfly pumping technique is a chemical sensor. Chemical sensors require molecules to reach a sampling surface for the properties of the fluid to be tested. Since the current study focused on the net flow moving in and out of a control volume, further research should be done to measure the rate that the target species, oxygen for the live mayfly, reach the surfaces of the mayfly. Determining the influence of the kinematic and geometric properties mentioned above on mass transport would provide insight for optimizing a chemical sensor.

Appendix A

```
// Robotic mayfly gill program

/* This program I controls 5 gills for a robotic array of mayfly gills.  These gills are
 * controlled by 2 servo motors each.  The servo motors are connected to an SCC32
 * servocontroller, which receives the commands from this program through a serial
 * port.  This program determines the pattern of the stroke and pitch that the gills
 * move in, as well as the frequency of the oscillation.  This program can be modified
* to make the gills move with four different phase lags between them: 0, 90, 180 or
 * 270 degrees. The amplitude of the stroke and pitch angles can also be adjusted.
* This program is written in C#.
 * */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace Rowing_Phase_Variations_2
{
  class Program
  {
     static void Main(string[] args)
     {
//============================================================
    // Open and configure serial port

    System.IO.Ports.SerialPort serialPort = new System.IO.Ports.SerialPort();

    // Close the serial port if it is already open
    if (serialPort.IsOpen) {
       serialPort.Close();
    }
    try {
       // Configure serial port
       serialPort.PortName = "COM1";
       serialPort.BaudRate = 115200;
       serialPort.Parity = System.IO.Ports.Parity.None;
       serialPort.DataBits = 8;
       serialPort.StopBits = System.IO.Ports.StopBits.One;

       //Open the serial port
       serialPort.Open();
    }
```

```
    catch (Exception ex)  {
        Console.WriteLine("Couldn't open the Serial Port!");
        Console.WriteLine(ex.ToString()); //Report the actual error
    }

//============================================================
    // Set the frequency of the cycle and the number of commands per cycle

    double frequency = 2;                       // [Hz] for the gills to perform one cycle
    int number_of_commands = 17;                // use 17 for 2 Hz, use 125 for .2 Hz
    double total_time = 1 / frequency;          // Period
    double time_interval = total_time / number_of_commands;
    int time_interval_int = Convert.ToInt32(time_interval * 1000);
    int length = (number_of_commands + 1);

    // Print information so user is aware of the operating conditions
    Console.WriteLine("The total cycle time is: {0} seconds", total_time);
    Console.WriteLine("The time intervals are: {0}", time_interval);
    Console.WriteLine("Time interval int:{0}", time_interval_int);

//============================================================
    // Initiate all variables

    // Declare time variables
    double[] time = new double[length];
    double[] time2 = new double[length];
    double[] time3 = new double[length];
    double[] time4 = new double[length];
    int[] time_int = new int[length];

    // Declare stroke variables
    double[] position1 = new double[length];
    double[] position2 = new double[length];
    double[] position3 = new double[length];
    double[] position4 = new double[length];
    double[] position5 = new double[length];

    // Define physical lengths for calculating pitch
    double R1a = 1.9; // a corresponds to the metal arm
    double R1b = 2.8; // b corresponds to the radius of servo arm
    double R2a = 2.5; // 1-5 are the servo numbers
    double R2b = 3.2;
    double R3a = 2.8;
    double R3b = 3.2;
    double R4a = 2.7;
    double R4b = 2.8;
```

```
double R5a = 3.0;
double R5b = 3.0;

// Declare pitch variables
double[] alpha1 = new double[length];        // pitch angle
double[] delta1 = new double[length];        // difference between 2 arms
double[] gamma1 = new double[length];     // angle difference between 2 servos
double[] p_position1 = new double[length]; // angle servo must be at for pitch
double[] alpha2 = new double[length];
double[] delta2 = new double[length];
double[] gamma2 = new double[length];
double[] p_position2 = new double[length];
double[] alpha3 = new double[length];
double[] delta3 = new double[length];
double[] gamma3 = new double[length];
double[] p_position3 = new double[length];
double[] alpha4 = new double[length];
double[] delta4 = new double[length];
double[] gamma4 = new double[length];
double[] p_position4 = new double[length];
double[] alpha5 = new double[length];
double[] delta5 = new double[length];
double[] gamma5 = new double[length];
double[] p_position5 = new double[length];

// Declare remaining variables
string[] time_string = new string[time.Length];
double[] time_servo = new double[time.Length];
int[] time_servo_int = new int[time.Length];
string[] stroke_string1 = new string[position1.Length];
string[] pitch_string1 = new string[position1.Length];
string[] stroke_string2 = new string[position1.Length];
string[] pitch_string2 = new string[position1.Length];
string[] stroke_string3 = new string[position1.Length];
string[] pitch_string3 = new string[position1.Length];
string[] stroke_string4 = new string[position1.Length];
string[] pitch_string4 = new string[position1.Length];
string[] stroke_string5 = new string[position1.Length];
string[] pitch_string5 = new string[position1.Length];
double[] stroke_servo1 = new double[position1.Length];
double[] pitch_servo1 = new double[position1.Length];
double[] stroke_servo2 = new double[position1.Length];
double[] pitch_servo2 = new double[position1.Length];
double[] stroke_servo3 = new double[position1.Length];
double[] pitch_servo3 = new double[position1.Length];
double[] stroke_servo4 = new double[position1.Length];
```

```
    double[] pitch_servo4 = new double[position1.Length];
    double[] stroke_servo5 = new double[position1.Length];
    double[] pitch_servo5 = new double[position1.Length];
    double[] stroke_servo_wn1 = new double[position1.Length];
    double[] pitch_servo_wn1 = new double[position1.Length];
    double[] stroke_servo_wn2 = new double[position1.Length];
    double[] pitch_servo_wn2 = new double[position1.Length];
    double[] stroke_servo_wn3 = new double[position1.Length];
    double[] pitch_servo_wn3 = new double[position1.Length];
    double[] stroke_servo_wn4 = new double[position1.Length];
    double[] pitch_servo_wn4 = new double[position1.Length];
    double[] stroke_servo_wn5 = new double[position1.Length];
    double[] pitch_servo_wn5 = new double[position1.Length];

// ========================================================
    // Determine phase lag

    // This program is written to control the gills with 4 different phase lags
    // between them: 0, 90, 180 and 270.  Only one of these phase lags is actually
    // used at a time. The code for the other phase lags is commented out and can
    // be uncommented when needed.

    for (int i = 0; i <= length - 1; i++) {

    // phase lag = 0 degrees

    time[i] = time_interval * i;
    time4[i] = time[i];
    time3[i] = time[i];
    time2[i] = time[i];

    //phase lag = 90
    /*
    time[i] = time_interval * i;
    time4[i] = time_interval * i + total_time * .25;
    if (time4[i] > total_time){
    time4[i] = time4[i] - total_time;
    }
    time3[i] = time_interval * i + total_time * .5;
    if (time3[i] > total_time){
    time3[i] = time3[i] - total_time;
    }
    time2[i] = time_interval * i + total_time * .75;
    if (time2[i] > total_time){
    time2[i] = time2[i] - total_time;
    }   */
```

```
    // phase lag = 180
    /*
    time[i] = time_interval * i;
    time4[i] = time_interval * i + total_time * .5;
    if (time4[i] > total_time){
       time4[i] = time4[i] - total_time;
    }
    time3[i] = time_interval * i;
    if (time3[i] > total_time){
       time3[i] = time3[i] - total_time;
    }
    time2[i] = time_interval * i + total_time * .5;
    if (time2[i] > total_time){
       time2[i] = time2[i] - total_time;
    }   */

    // phase lag = 270
    /*
    time[i] = time_interval * i;
    time4[i] = time_interval * i + total_time * .75;
    if (time4[i] > total_time){
    time4[i] = time4[i] - total_time;
    }
    time3[i] = time_interval * i + total_time * .5;
    if (time3[i] > total_time){
    time3[i] = time3[i] - total_time;
    }
    time2[i] = time_interval * i + total_time * .25;
    if (time2[i] > total_time){
    time2[i] = time2[i] - total_time;
    } */

    time_int[i] =Convert.ToInt32(time[i] * 1000);

//============================================================
    // Define angles for the stroke and pitch at each time step for each of the 5 gills

    // Stroke equation
    // For the different phase lags, 64% of the stroke (29.9 degrees) was used

    position1[i] = 0.64*(-116390045.158509 * Math.Pow(time[i], 10)
          +266045373.553205 * Math.Pow(time[i], 9) - 246871200.957511 *
          Math.Pow(time[i], 8) + 117757776.899578 * Math.Pow(time[i], 7)
          - 29762862.2008976 * Math.Pow(time[i], 6) + 3568716.37400224 *
          Math.Pow(time[i], 5) - 152710.423058667 * Math.Pow(time[i], 4) +
```

```
            21623.3710732189 * Math.Pow(time[i], 3) - 4745.31713106107 *
            Math.Pow(time[i], 2) + -99.0884892629213 * Math.Pow(time[i], 1)
             + 23.1848021279438 * Math.Pow(time[i], 0));
        position2[i] = 0.64*( -116390045.158509 * Math.Pow(time2[i], 10) +
            266045373.553205 * Math.Pow(time2[i], 9) - 246871200.957511 *
            Math.Pow(time2[i], 8) + 117757776.899578 * Math.Pow(time2[i], 7)
            - 29762862.2008976 * Math.Pow(time2[i], 6) + 3568716.37400224 *
            Math.Pow(time2[i], 5) - 152710.423058667 * Math.Pow(time2[i], 4) +
            21623.3710732189 * Math.Pow(time2[i], 3) - 4745.31713106107 *
            Math.Pow(time2[i], 2) + -99.0884892629213 * Math.Pow(time2[i], 1)
             + 23.1848021279438 * Math.Pow(time2[i], 0));
        position3[i] = 0.64*(-116390045.158509 * Math.Pow(time3[i], 10) +
            266045373.553205 * Math.Pow(time3[i], 9) - 246871200.957511 *
            Math.Pow(time3[i], 8) + 117757776.899578 * Math.Pow(time3[i], 7)
            - 29762862.2008976 * Math.Pow(time3[i], 6) + 3568716.37400224 *
            Math.Pow(time3[i], 5) - 152710.423058667 * Math.Pow(time3[i], 4) +
            21623.3710732189 * Math.Pow(time3[i], 3) - 4745.31713106107 *
            Math.Pow(time3[i], 2) + -99.0884892629213 * Math.Pow(time3[i], 1)
             + 23.1848021279438 * Math.Pow(time3[i], 0));
    position4[i] = 0.64*(-116390045.158509 * Math.Pow(time4[i], 10) +
            266045373.553205 * Math.Pow(time4[i], 9) - 246871200.957511 *
            Math.Pow(time4[i], 8) + 117757776.899578 * Math.Pow(time4[i], 7)
            - 29762862.2008976 * Math.Pow(time4[i], 6) + 3568716.37400224 *
            Math.Pow(time4[i], 5) - 152710.423058667 * Math.Pow(time4[i], 4) +
            21623.3710732189 * Math.Pow(time4[i], 3) - 4745.31713106107 *
            Math.Pow(time4[i], 2) + -99.0884892629213 * Math.Pow(time4[i], 1)
            + 23.1848021279438 * Math.Pow(time4[i], 0));
    position5[i] = 0.64*(-116390045.158509 * Math.Pow(time[i], 10) +
            266045373.553205 * Math.Pow(time[i], 9)- 246871200.957511 *
            Math.Pow(time[i], 8) + 117757776.899578 * Math.Pow(time[i], 7)
            - 29762862.2008976 * Math.Pow(time[i], 6) + 3568716.37400224 *
            Math.Pow(time[i], 5)- 152710.423058667 * Math.Pow(time[i], 4) +
            21623.3710732189 * Math.Pow(time[i], 3)- 4745.31713106107 *
            Math.Pow(time[i], 2) + -99.0884892629213 * Math.Pow(time[i], 1)
             + 23.1848021279438 * Math.Pow(time[i], 0));

        // Pitch equation
        // For the different phase lags, 64% of the pitch (22.1 degrees) was used

        alpha1[i] = 0.64*(-7692847089.22796 * Math.Pow(time[i], 12) +
            21518450265.0879 * Math.Pow(time[i], 11) - 26006170949.7297 *
            Math.Pow(time[i], 10) + 17730678634.0559 * Math.Pow(time[i], 9)
             - 7454174175.65348 * Math.Pow(time[i], 8) + 1975573951.37683 *
            Math.Pow(time[i], 7) + -320771535.184523 * Math.Pow(time[i], 6) +
            28673933.0945493 * Math.Pow(time[i], 5) +  -996532.331218552 *
            Math.Pow(time[i], 4) + 5983.73234934935 * Math.Pow(time[i], 3) +
```

```
            -6057.66274011221 * Math.Pow(time[i], 2) + 482.534460207715 *
            Math.Pow(time[i], 1) + 8.13131430604209 * Math.Pow(time[i], 0));
        alpha2[i] = 0.64 * (-7692847089.22796 * Math.Pow(time2[i], 12) +
            21518450265.0879 * Math.Pow(time2[i], 11)
            - 26006170949.7297 * Math.Pow(time2[i], 10) + 17730678634.0559 *
            Math.Pow(time2[i], 9) - 7454174175.65348 * Math.Pow(time2[i], 8) +
            1975573951.37683 * Math.Pow(time2[i], 7) + -320771535.184523 *
            Math.Pow(time2[i], 6) + 28673933.0945493 * Math.Pow(time2[i], 5) +
             -996532.331218552 * Math.Pow(time2[i], 4) + 5983.73234934935 *
            Math.Pow(time2[i], 3) + -6057.66274011221 * Math.Pow(time2[i], 2) +
            482.534460207715 * Math.Pow(time2[i], 1) + 8.13131430604209 *
            Math.Pow(time2[i], 0));
        alpha3[i] = 0.64 * (-7692847089.22796 * Math.Pow(time3[i], 12) +
            21518450265.0879 * Math.Pow(time3[i], 11) - 26006170949.7297 *
            Math.Pow(time3[i], 10) + 17730678634.0559 * Math.Pow(time3[i], 9)
            - 7454174175.65348 * Math.Pow(time3[i], 8) + 1975573951.37683 *
            Math.Pow(time3[i], 7) +-320771535.184523 * Math.Pow(time3[i], 6) +
            28673933.0945493 * Math.Pow(time3[i], 5) +-996532.331218552 *
            Math.Pow(time3[i], 4) + 5983.73234934935 * Math.Pow(time3[i], 3) +
             -6057.66274011221 * Math.Pow(time3[i], 2) + 482.534460207715 *
            Math.Pow(time3[i], 1) + 8.13131430604209 * Math.Pow(time3[i], 0));
        alpha4[i] = 0.64 * (-7692847089.22796 * Math.Pow(time4[i], 12) +
            21518450265.0879 * Math.Pow(time4[i], 11)
            - 26006170949.7297 * Math.Pow(time4[i], 10) + 17730678634.0559 *
            Math.Pow(time4[i], 9)- 7454174175.65348 * Math.Pow(time4[i], 8) +
            1975573951.37683 * Math.Pow(time4[i], 7) +-320771535.184523 *
            Math.Pow(time4[i], 6) + 28673933.0945493 * Math.Pow(time4[i], 5) +
             -996532.331218552 * Math.Pow(time4[i], 4) + 5983.73234934935 *
            Math.Pow(time4[i], 3) +-6057.66274011221 * Math.Pow(time4[i], 2) +
            482.534460207715 * Math.Pow(time4[i], 1) +8.13131430604209 *
            Math.Pow(time4[i], 0));
        alpha5[i] = 0.64 * (-7692847089.22796 * Math.Pow(time[i], 12) +
            21518450265.0879 * Math.Pow(time[i], 11)
             - 26006170949.7297 * Math.Pow(time[i], 10) + 17730678634.0559 *
            Math.Pow(time[i], 9)- 7454174175.65348 * Math.Pow(time[i], 8) +
            1975573951.37683 * Math.Pow(time[i], 7) + -320771535.184523 *
            Math.Pow(time[i], 6) + 28673933.0945493 * Math.Pow(time[i], 5) +
             -996532.331218552 * Math.Pow(time[i], 4) + 5983.73234934935 *
            Math.Pow(time[i], 3) + -6057.66274011221 * Math.Pow(time[i], 2) +
            482.534460207715 * Math.Pow(time[i], 1) + 8.13131430604209 *
            Math.Pow(time[i], 0));

//=============================================================
    // Calculate angle for pitch servo

    // The pitch servo creates the pitch by moving relative to the stroke servo.
```

```
    // Geometric calculations must be performed to convert the pitch angle of
    // the gill into the relative pitch angle of the  pitch servo motor.
    // Must be done for each of the 5 gills.

  // Difference between the two arms
    delta1[i] = R1a *Math.Sin(alpha1[i] * Math.PI / 180);
// angle difference between 2 servo arms
    gamma1[i] =Math.Asin(delta1[i] / R1b) * 180 / Math.PI;
// angle servo must be at to create specified pitch
    p_position1[i] = position1[i] + gamma1[i];

    delta2[i] = R2a *Math.Sin(alpha2[i] * Math.PI / 180);
    gamma2[i] =Math.Asin(delta2[i] / R2b) * 180 / Math.PI;
    p_position2[i] = position2[i] + gamma2[i];

    delta3[i] = R3a *Math.Sin(alpha3[i] * Math.PI / 180);
    gamma3[i] = Math.Asin(delta3[i] / R3b) * 180 / Math.PI;
    p_position3[i] = position3[i] + gamma3[i];

    delta4[i] = R4a *Math.Sin(alpha4[i] * Math.PI / 180);
    gamma4[i] = Math.Asin(delta4[i] / R4b) * 180 / Math.PI;
    p_position4[i] = position4[i] + gamma4[i];

    delta5[i] = R5a *Math.Sin(alpha5[i] * Math.PI / 180);
    gamma5[i] = Math.Asin(delta5[i] / R5b) * 180 / Math.PI;
    p_position5[i] = position5[i] + gamma5[i];
    }

// ============================================================
    // Define time for each signal to be sent. Convert position angles to
    // positions in microseconds for servo commands. The central position
    // for a servomotor is 1500 microseconds

    for (int m = 0; m <= time.Length - 2; m++)
    {
    time_servo[m] = (time_int[m + 1] - time_int[m]);
    time_servo_int[m] = Convert.ToInt32(time_servo[m]);
    time_string[m] = time_servo_int[m].ToString();// convert to string

    // Stroke

    //Position for Servo #0
    stroke_servo1[m] = 1500 + position1[m] * 10 + 15; // convert to servo positons
    stroke_servo_wn1[m] = Math.Round(stroke_servo1[m]); // round to whole number
    stroke_string1[m] = stroke_servo_wn1[m].ToString(); //convert to string
```

```
    //Position for Servo #2
    stroke_servo2[m] = 1500 + position2[m] * 10 - 115;
    stroke_servo_wn2[m] =Math.Round(stroke_servo2[m]);
    stroke_string2[m] = stroke_servo_wn2[m].ToString();

    //Position for Servo #4
    stroke_servo3[m] = 1500 + position3[m] * 10 + 07;
    stroke_servo_wn3[m] =Math.Round(stroke_servo3[m]);
    stroke_string3[m] = stroke_servo_wn3[m].ToString();

    //Position for Servo #6
    stroke_servo4[m] = 1500 + position4[m] * 10 + 30;
    stroke_servo_wn4[m] =Math.Round(stroke_servo4[m]);
    stroke_string4[m] = stroke_servo_wn4[m].ToString();

    //Position for Servo #8
    stroke_servo5[m] = 1500 + position5[m] * 10 + 95; //+ 31;
    stroke_servo_wn5[m] = Math.Round(stroke_servo5[m]);
    stroke_string5[m] = stroke_servo_wn5[m].ToString();

    // Pitch

    //Position for Servo #1
// adjustment for displacement between 2 arms
    pitch_servo1[m] = 1500 - p_position1[m] * 10 + 13;
    pitch_servo_wn1[m] =Math.Round(pitch_servo1[m]);
    pitch_string1[m] = pitch_servo_wn1[m].ToString();

    //Position for Servo #3
    pitch_servo2[m] = 1500 - p_position2[m] * 10 + 33;
    pitch_servo_wn2[m] =Math.Round(pitch_servo2[m]);
    pitch_string2[m] = pitch_servo_wn2[m].ToString();

    //Position for Servo #5
    pitch_servo3[m] = 1500 - p_position3[m] * 10 + 76;
    pitch_servo_wn3[m] =Math.Round(pitch_servo3[m]);
    pitch_string3[m] = pitch_servo_wn3[m].ToString();

    //Position for Servo #7
    pitch_servo4[m] = 1500 - p_position4[m] * 10 + 105;
    pitch_servo_wn4[m] =Math.Round(pitch_servo4[m]);
    pitch_string4[m] = pitch_servo_wn4[m].ToString();

    //Position for Servo #9
    pitch_servo5[m] = 1500 - p_position5[m] * 10 + 42;
    pitch_servo_wn5[m] =Math.Round(pitch_servo5[m]);
```

```
    pitch_string5[m] = pitch_servo_wn5[m].ToString();

    Console.WriteLine("Stroke: " + stroke_string3[m] + "Pitch: " + pitch_string3[m]);
    }

// ============================================================
    // Give servos an initial position

    // The servos are connected to ports on the SSC32 servo controller that
    // are labeled with numbers ranging from #0 to #31.  These numbers are
    // used to distinguish which commands are sent to which servomotor.

    try {
    serialPort.Write("#21 P" + stroke_string1[0] + " #23 P" + pitch_string1[0] +
                "#17 P" + stroke_string2[0] + " #19 P" + pitch_string2[0] +
                "#6 P" + stroke_string3[0] + " #7 P" + pitch_string3[0] +
                "#3 P" + stroke_string4[0] + " #4 P" + pitch_string4[09] +
                "#0 P" + stroke_string5[0] + " #1 P" + pitch_string5[0] + " \r");
    }
    catch (Exception ex)
    {
    Console.WriteLine("Couldn't send the command");
    Console.WriteLine(ex.ToString()); //Report the actual error
    }
    Console.WriteLine("Sent: #0 P" + stroke_string1[0] + " <cr>" +
Environment.NewLine);

// ============================================================
    // Ask user how many times to loop the movements

    Console.WriteLine("How many times would you like to loop the movement?");

    string input = Console.ReadLine();
    double inputAsNumber;
    if (double.TryParse(input, out inputAsNumber) == true)
    {
       Console.WriteLine("You entered a valid number: {0}", inputAsNumber);
    }
    else
    {
       Console.WriteLine("You entered an invalid number");
    }

// ============================================================
    // Give servos commands
```

```
   for (int i = 1; i <= inputAsNumber; i++) {
   for (int j = 0; j <= time.Length - 2; j++) {
   try {

      // The value of j in the if statement determines which phase in the cycle
      // is recorded in the PIV setup.  j ==16 means that the PIV cycle is triggered
      // for the 16th phase angle (There are 17 total, since 17 commands are sent).
      if (j ==16)    {
      serialPort.Write("#21 P" + stroke_string1[j] + " #23 P" + pitch_string1[j] +
                  "#17 P" + stroke_string2[j] + " #19 P" + pitch_string2[j] +
                  "#6 P" + stroke_string3[j] + " #7 P" + pitch_string3[j] +
                  "#3 P" + stroke_string4[j] + " #4 P" + pitch_string4[j] +
                  "#0 P" + stroke_string5[j] + " #1 P" + pitch_string5[j] +
                  "#31 P2000 \r");
      }
      else    {
      serialPort.Write("#21 P" + stroke_string1[j] + " #23 P" + pitch_string1[j] +
                  "#17 P" + stroke_string2[j] + " #19 P" + pitch_string2[j] +
                  "#6 P" + stroke_string3[j] + " #7 P" + pitch_string3[j] +
                  "#3 P" + stroke_string4[j] + " #4 P" + pitch_string4[j] +
                  "#0 P" + stroke_string5[j] + " #1 P" + pitch_string5[j] +
                  "#31 P0000 \r");
      }
   }

   catch (Exception ex)
   {
   Console.WriteLine("Couldn't send the command");
   Console.WriteLine(ex.ToString()); //Report the actual error
   }

   // There is a pause in between when the commands are sent to the servo motors
   // Generally when using an SSC 32, commands are sent every 30 ms
   Thread.Sleep(time_servo_int[j] - 2);
   }
   }

// ============================================================
   //Close serial port.

   try {
      serialPort.Close();
   }
   catch (Exception ex) {
      Console.WriteLine("Couldn't close the serial port. Here's what it said: " +
Environment.NewLine);
```

```
        Console.WriteLine(ex.ToString());
      }
      // Wait for [Enter] before we close
        Console.ReadLine();
      }
    }
  }
```

Appendix B

```
/ ------------------------------------------------------------
// Copyright (C) LaVision GmbH.  All Rights Reserved.
// ------------------------------------------------------------
// Filename:  MYMEDIANSELECTION.CL
//
// Description: User Batch Processing function
//
// Date:    Tue Oct 26, 2010 15:02:35
// ------------------------------------------------------------

/// <!-------------------------------------------------------------------------------------------->
// This program sorts 40 vector fields based on how close they are to the median of
 // the 40 vector fields.  The 30 vector fields that are the closest to the median are
// selected.  The original PIV images associated with these 30 vector fields are then
// returned so that they can be reprocessed.  This program was used in DaVis 7 as a
// "user function" and uses the DaVis command language.  When the program is put
// in the operations list in DaVis batch processing, the operation list makes it run
// through the program once for each image in a set.  At the end of the program,
// certain parameters determine whether the program should be run on each image
// again or whether there should be another "pass".  This program has 2 passes.  The
// first pass is to load and sort the vector fields.  The second pass is to return the
// appropriate PIV images.
/// <!-------------------------------------------------------------------------------------------->//

 /// ========================================================
// Initiate all variables

int PassCount; int PassCount2;  int count1; int count2; int count3; int count4;
int i2; int j2; int k2; int i3; int j3; int i4; int j4; int i5; int j5; int i6; int i7; int j7; int k7;
int i8; int j8;  int n1; int n2;
int try2;  int time =0; int mm; int Nx; int Ny;

float imageIndex[91]; float imageIndex2[91];  float lastIndex;
float ax[40]; float ay[40]; float az[40];  float tempx; float tempy; float tempz;
float rx[40]; float ry[40]; float rz[40];  float temprx; float tempry; float temprz;
float errt[40]; float err[40]; float temp_err;

// Buffer Assignments (a buffer must be assigned a number so it can be called again)
int medianx=50; int mediany =51; int medianz=52;
int medianrx = 53;  int medianry = 54; int medianrz=55;
int resx = 300; int resy = 340; int resz=380;
int errx = 56; int erry = 57; int errz=58; int err_tot=59;
float Vx=100; float Vy=140; float Vz=180;
```

```
int temp_index;  int comp[41];  string mySetFile;
int index[40]; // controls the number of images to be returned

void MYMEDIANSELECTION()
{
 // startup macro
          BP_AddUserFunction( IBP_SRC_VECTOR ,"user function",
         "MyMedianSelection", "MyMedianSelection");
}
/// <!-------------------------------------------------------------------------------------------->
/// @file MYMEDIANSELECTION
/// @par Prefix: MyMedianSelection
/// <!-------------------------------------------------------------------------------------------->

#GROUP MyMedianSelection Parameter
static int MyMedianSelection_Parameter1;

/// <!-------------------------------------------------------------------------------------------->
/// <!--                        MyMedianSelection_Dialog                                    -->
/// <!-------------------------------------------------------------------------------------------->
/// @brief Parameter dialog
/// <!-------------------------------------------------------------------------------------------->
void MyMedianSelection_Dialog()
// Open small parameter dialog to change operation specific parameter
{
       int did = Dialog( "MyMedianSelection_Dlg", 100, 100, 400, 400,
       "MyMedianSelection_DlgName" );
        int y = 10;
        // .. add items with user function parameter
        AddItem( did, 5,  DITEM_GROUP,  10,  y, 380, 55, "MyMedianSelection
       parameter:", "" );
        y += 25;
        AddItem( did, 10, DITEM_TEXT,  20,  y, 200, 20, "Images to return:", "" );
        AddItem( did, 11, DITEM_EDIT, 220,  y,  60, 20, "",
       "MyMedianSelection_Parameter1" );
        AddItem( did, 12, DITEM_TEXT, 285,  y,  60, 20, "images", "" );
        y += 25;
        ShowDialog(did);
}

/// <!-------------------------------------------------------------------------------------------->
/// <!--                        MyMedianSelection_Dlg_Event                                 -->
/// <!-------------------------------------------------------------------------------------------->
/// @brief Event handler of the parameter dialog
/// <!-------------------------------------------------------------------------------------------->
```

```
/// @param p_nItem Dialog item number
/// <!------------------------------------------------------------------------------------------->
void MyMedianSelection_Dlg_Event(int p_nItem)
{
    // dialog event handler
    int nDId = GetDialogId("MyMedianSelection_Dlg");
    ApplyDialog(nDId);
    switch (p_nItem)
        {
        case EVENT_RETURN:  // RETURN event
        break;
        }
     UpdateDialog(nDId);
}

/// <!------------------------------------------------------------------------------------------->
/// <!--                    MyMedianSelection_ResultType                    -->
/// <!------------------------------------------------------------------------------------------->
/// @brief Return the result type of the operation
/// <!------------------------------------------------------------------------------------------->
/// @param p_nSourceType Current source type: IBP_SRC_IMAGE (image) or
///   IBP_SRC_VECTOR (vector)
/// <!------------------------------------------------------------------------------------------->
/// @return IBP_RESULT_VEC (image) or IBP_RESULT_VEC (vector)
/// <!------------------------------------------------------------------------------------------->
int MyMedianSelection_ResultType( int p_nSourceType )
{
      return IBP_RESULT_IMG;
}

/// <!------------------------------------------------------------------------------------------->
/// <!--                    MyMedianSelection_StoreGroups                    -->
/// <!------------------------------------------------------------------------------------------->
/// @brief Return the name of the CL variable group with the operation settings
/// <!------------------------------------------------------------------------------------------->
/// @return Name of the CL variable group
/// <!------------ ------------------------------------------------------------------------------>
string MyMedianSelection_StoreGroups()
{ // Store this variable group in the result setfile
     return "MyMedianSelection Parameter";
}
/// <!------------------------------------------------------------------------------------------->
/// <!--                 MyMedianSelection_GetDefaultFileOrSetName           -->
/// <!------------------------------------------------------------------------------------------->
/// @brief Return the operation specific default store name.
/// <!------------------------------------------------------------------------------------------->
```

```
/// @return The default store name.
/// <!-------------------------------------------------------------------------------------------->
string MyMedianSelection_GetDefaultFileOrSetName()
// Return the operation specific store name (for default setfiles)
{
    return "MyMedianSelect";
}
/// <!-------------------------------------------------------------------------------------------->
/// <!--                        MyMedianSelection_Operation                                    -->
/// <!-------------------------------------------------------------------------------------------->
/// @brief This is the main operation macro which is called for each source
///    image/vector.
/// <!-------------------------------------------------------------------------------------------->
/// Implement your operation in this function.
/// <!-------------------------------------------------------------------------------------------->
/// @param p_nBuffer Number of the source buffer, store the result in this buffer.
/// @param p_nImageIndex Number of the current image/vector (1-n)
/// <!-------------------------------------------------------------------------------------------->
/// @return IBP_OP_STORE  ok, store buffer in result data set
///   IBP_OP_NOSTORE  ok, but no storage
///   IBP_OP_ERROR  error, abort operation
/// <!-------------------------------------------------------------------------------------------->
int MyMedianSelection_Operation( int p_nBuffer, int p_nImageIndex )
{
   if(p_nImageIndex==1 && time != 1){
       PassCount=0;
       time=1;
   }
switch(PassCount){
/// ============================================================
// Case 0 loads all the vector fields in the set into designated buffers. On the last pass
// (the 40th image for my data), it calculates the median of all of the images then sorts
// the images based on which are most similar to the median
   case 0:
      count1++;
      PassCount2=0;

      // Loads vector field into buffer so it can be accessed in the next pass
      B[p_nImageIndex] = B[p_nBuffer];

      // Separate U, V and W velocity components
      VectorOperation(p_nBuffer,Vx+p_nImageIndex-1, V_OP_VX,0);
      VectorOperation(p_nBuffer,Vy+p_nImageIndex-1, V_OP_VY,0);
      VectorOperation(p_nBuffer,Vz+p_nImageIndex-1, V_OP_VZ,0);
```

```
      // Extract x and y coordinates
       Nx=GetBufferNX(Vx);
       Ny=GetBufferNY(Vy);

/// ============================================================

  // After all images have been assigned to buffers, enter if statement to
  // start median calculation (There are 40 vector fields in my set, so this would be
  // on the 40th pass)
       if (p_nImageIndex>39){
          count1 =0;
          count2++;
          PassCount=1;          // When the program is repeated, it will enter the second
                                // section of the switch statement

       // Create buffers necessary for median calculation and sorting
          CreateFloatBuffer(medianx, Nx, Ny);
          CreateFloatBuffer(mediany, Nx, Ny);
          CreateFloatBuffer(medianz, Nx, Ny);
          CreateFloatBuffer(medianrx, Nx, Ny);
          CreateFloatBuffer(medianry, Nx, Ny);
          CreateFloatBuffer(medianrz, Nx, Ny);
          CreateFloatBuffer(resx, Nx, Ny);
          CreateFloatBuffer(resy, Nx, Ny);
          CreateFloatBuffer(resz, Nx, Ny);
          CreateFloatBuffer(errx, Nx, Ny);
          CreateFloatBuffer(erry, Nx, Ny);
          CreateFloatBuffer(errz, Nx, Ny);
          CreateFloatBuffer(err_tot, Nx, Ny);
/// ============================================================
// Find Median
         for (i2 = 0; i2<Nx; i2++){
             for(j2=0;j2<Ny; j2++){

               for(k2=0; k2<40; k2++){
                  ax[k2] = pix[Vx+k2,i2,j2];
                  ay[k2] = pix[Vy+k2,i2,j2];
                  az[k2] = pix[Vz+k2,i2,j2];
               }
              //Sort vector
              for(i3=0; i3<40; i3++){
                  for(j3=i3+1; j3<40; j3++){
                     if(ax[i3]>ax[j3]){
                        tempx = ax[j3];
                        ax[j3]=ax[i3];
                        ax[i3]=tempx;
```

```
                }
                if(ay[i3]>ay[j3]){
                  tempy = ay[j3];
                  ay[j3]=ay[i3];
                  ay[i3]=tempy;
                }
                if(az[i3]>az[j3]){
                  tempz = az[j3];
                  az[j3]=az[i3];
                  az[i3]=tempz;
                }
              }
            }
            pix[medianx, i2, j2]= (ax[19]+ax[20])/2;  // medianx buffer is buffer 50
            pix[mediany, i2, j2]= (ay[19]+ay[20])/2;  // mediany buffer is buffer 51
            pix[medianz, i2, j2]= (az[19]+az[20])/2;  // medianz buffer is 52
          }
        }
/// ============================================================
// Calculate Residuals (original vectors-median vectors)
        for(n2=0; n2<40;n2++){
                B[resx+n2] = B[Vx+n2]-B[medianx];
                B[resy+n2] = B[Vy+n2]-B[mediany];
                B[resz+n2] = B[Vz+n2]-B[medianz];
        }
         // Find median of residuals
         for (i7 = 0; i7<Nx; i7++){
           for(j7=0;j7<Ny; j7++){
              for(k7=0; k7<40; k7++){
                rx[k7] = abs(pix[resx+k7,i7,j7]);
                ry[k7] = abs(pix[resy+k7,i7,j7]);
                rz[k7] = abs(pix[resz+k7,i7,j7]);
              }
            //Sort vector
            for(i8=0; i8<40; i8++){
               for(j8=i8+1; j8<40; j8++){
                if(rx[i8]>rx[j8]){
                  temprx = rx[j8];
                  rx[j8]=rx[i8];
                  rx[i8]=temprx;
                }
                if(ry[i8]>ry[j8]){
                  tempry = ry[j8];
                  ry[j8]=ry[i8];
                  ry[i8]=tempry;
                }
```

```
            if(rz[i8]>rz[j8]){
              temprz = rz[j8];
              rz[j8]=rz[i8];
              rz[i8]=temprz;
            }
          }
        }
        pix[medianrx, i7, j7]= (rx[19]+rx[20])/2;  // medianx buffer is buffer0
        pix[medianry, i7, j7]= (ry[19]+ry[20])/2;  // medianу buffer is buffer
        pix[medianrz, i7, j7]= (rz[19]+rz[20])/2;  // medianz buffer is
    }
}

// Calculate error from residual
n1=0;
for(n1=0; n1<40;n1++){
 errt[n1]=0;
 for (i4=0; i4<Nx; i4++){
    for(j4=0; j4<Ny; j4++){
       pix[errx,i4,j4] = abs(pix[resx+n1,i4,j4])/(pix[medianrx,i4,j4]+.1);
       pix[erry,i4,j4] = abs(pix[resy+n1,i4,j4])/(pix[medianry,i4,j4]+.1);
       pix[errz,i4,j4] = abs(pix[resz+n1,i4,j4])/(pix[medianrz,i4,j4]+.1);
       pix[err_tot,i4,j4] = sqrt(pix[errx,i4,j4]*pix[errx,i4,j4]+pix[erry,i4,j4]*
                                 pix[erry,i4,j4]+pix[errz,i4,j4]*pix[errz,i4,j4]);
       errt[n1] = pix[err_tot,i4,j4] +errt[n1];
}
}
err[n1] = errt[n1]/(Nx*Ny);
}
// index is the image number that will be associated with the error value
index = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40};
for(i5=0; i5<40; i5++){
    for(j5=i5+1; j5<40; j5++){
       if(err[i5]>err[j5]){  // Sorts smallest to largest error
           temp_index =index[j5];
           index[j5]=index[i5];
           index[i5]=temp_index;
           temp_err = err[j5];
           err[j5]=err[i5];
           err[i5]=temp_err;
        }
     }
}
int p1;
// clear comp variable
```

```
          for (p1=0; p1<41;p1++){
            comp[p1]=0;
          }
          // assign comp 1s for the 30 image with the smallest errors
          for (i6=0; i6<30; i6++){
            mm= index[i6];
            comp[mm] =1;
          }
      }
    break;
/// ========================================================
// The second time the program is run (case 1), the image files associated
// with the vector fields are identified and it is determined, based on the results of how
// close the vector fields were to the median, whether the image should be returned for
// the final set.  30 images are returned for the final set.

case 1:
      PassCount2=3;
      // identify file route that current files are in
      if (count3==0){
          // This gets the name of file that the images are originally taken from
           mySetFile = BP_GetRootSourceFile();
           // Turn these off if in a multi operation loop
           mySetFile = SET_GetSourceSet(mySetFile);
           mySetFile = SET_GetSourceSet(mySetFile);
           count1=0;
       }
      count3++;
      imageIndex2[count3]=p_nImageIndex;
      lastIndex= p_nImageIndex;
      FreeBuffer(count4+59); // Clears buffers

      // Load image files associated with vector fields with smallest error
      if(comp[count3]==1){
           SET_LoadBuffer(mySetFile,count3,count4+60);
           B[p_nBuffer] = B[count4+60];
           count4++;
      }
      if (count3>39){
        count4=0;
        PassCount = 2;
      }
    break;
 }
/// ========================================================
 // Determine whether to store 'theBuffer' (the current image) or not.  It will be stored
```

```
// if it was in the top 30 (‘comp’ would have been assigned a 1)
if (PassCount2==3 && comp[count3]==1){
   comp[count3]=0;  // Set back to 0 so that in the next loop comp =0
   if (p_nImageIndex==40){ // Clear variables at the end of the process
       PassCount=0;
       PassCount2=0;
       count3=0;
       count2=0;
       count1=0;
    }
    return IBP_OP_STORE;              // Store the buffer
}
else
    if (PassCount2==3 && p_nImageIndex==40){
       PassCount2=0;
       count3=0;
       count2=0;
       count1 =0;
     }
     return IBP_OP_NOSTORE;        // Do not store the buffer
}
/// <!-------------------------------------------------------------------------------------------->
/// <!--                        MyMedianSelection_GetInfoString                                 -->
/// <!-------------------------------------------------------------------------------------------->
/// @brief Return information string with operation details.
/// <!-------------------------------------------------------------------------------------------->
/// Return an information string which is added to the result setfile. This
/// information is displayed in the DaVis Project Manager -> 'Properties' card ->
'Batch Jobs'
/// <!-------------------------------------------------------------------------------------------->
/// @return Information string
/// <!-------------------------------------------------------------------------------------------->
string MyMedianSelection_GetInfoString()
{
          return "MyMedianSelection information...";
}

/// <!-------------------------------------------------------------------------------------------->
/// <!--                        MyMedianSelection_CheckSource                                   -->
/// <!-------------------------------------------------------------------------------------------->
/// @brief Function to check the souce data set.
/// <!-------------------------------------------------------------------------------------------->
/// This function is called just before the ..._StartExecute(..) macro.
/// Use this macro to check the source or overwrite the dialog
/// settings min, max or increment.
/// <!-------------------------------------------------------------------------------------------->
```

```
/// @param p_nSourceMode Type of the current source: IBP_IN_*
/// @param p_sSetFileName Name of source file (if source = dataset or file)
/// @param p_rnMin Source data set start index (1-..)
/// @param p_rnMax Source data set end index (1-..)
/// @param p_rnInc Image increment
/// <!------------------------------------------------------------------------------------------>
/// @return 0 (ok) or -1 to disable the 'Start Processing' button
/// <!------------------------------------------------------------------------------------------>
int MyMedianSelection_CheckSource(int p_nSourceMode, string p_sSetFileName,
int& p_rnMin, int& p_rnMax, int& p_rnInc)
{
        return 0;
}
/// <!------------------------------------------------------------------------------------------>
/// <!--                        MyMedianSelection_StartExecute                              -->
/// <!------------------------------------------------------------------------------------------>
/// @brief This function is called just before the batch processing loop.
/// <!------------------------------------------------------------------------------------------>
/// The macro can be used to initialize the operation, for example get a temporary
///  buffer.
/// <!------------------------------------------------------------------------------------------>
/// @param p_nNumImages Number of source images
/// <!------------------------------------------------------------------------------------------>
/// @return 0 = ok, -1 = abort operation
/// <!------------------------------------------------------------------------------------------>
int MyMedianSelection_StartExecute(int p_nNumImages)
{
         return 0;
}
/// <!------------------------------------------------------------------------------------------>
/// <!--                        MyMedianSelection_ResultBuffers                             -->
/// <!------------------------------------------------------------------------------------------>
/// @brief Return the number of additional result buffers.
/// <!------------------------------------------------------------------------------------------>
/// @return Number of additional result buffers.
/// <!------------------------------------------------------------------------------------------>
int MyMedianSelection_ResultBuffers()
{
        return 0;
}
/// <!------------------------------------------------------------------------------------------>
/// <!--                        MyMedianSelection_GetOperationMode                          --
>
/// <!------------------------------------------------------------------------------------------>
/// @brief Return the supported operation mode (if DaVis version >= 7.1)
/// <!------------------------------------------------------------------------------------------>
```

```
/// @return IBP_OPMODE_STREAM : supports streaming: 1 result buffer for each
/// source buffer, otherwise return IBP_OPMODE_DEFAULT
/// <!-------------------------------------------------------------------------------------------->
int MyMedianSelection_GetOperationMode()
{
        return IBP_OPMODE_STREAM;
}
/// <!-------------------------------------------------------------------------------------------->
/// <!--                    MyMedianSelection_OverwriteSource                    -->
/// <!-------------------------------------------------------------------------------------------->
/// @brief The return value (TRUE/FALSE) controls if the 'overwrite source' store
///  mode is available.
/// <!-------------------------------------------------------------------------------------------->
/// CAUTION: The 'overwrite source' store mode overwrites the source data set!
/// <!-------------------------------------------------------------------------------------------->
/// @return TRUE ('overwrite source' available, otherwise FALSE
/// <!-------------------------------------------------------------------------------------------->
int MyMedianSelection_OverwriteSource()
{
        return FALSE;   // default: 'overwrite source' is not available
}
/// <!-------------------------------------------------------------------------------------------->
/// <!--                      MyMedianSelection_EndExecute                      -->
/// <!-------------------------------------------------------------------------------------------->
/// @brief Finalize function.
/// <!-------------------------------------------------------------------------------------------->
/// This function is called just after the batch processing loop, also when it has
/// been stopped by the spacebar ' ' key. It can be used for any final action.
/// If the functions calculates additional results (see <...>_ResultBuffers),
/// the results must be copied into 'p_nBuffer', 'p_nBuffer+1', ...
/// The titles of this buffers will be used as postfixes for the file names.
/// <!-------------------------------------------------------------------------------------------->
/// @param p_nBuffer Use this buffer number to store additional result.
/// @param p_nNumImages Number of additional results
/// <!-------------------------------------------------------------------------------------------->
/// @return 0 = ok, -1 = abort processing
/// <!-------------------------------------------------------------------------------------------->
int MyMedianSelection_EndExecute(int p_nBuffer, int p_nNumImages)
{
        return 0;
}
/// <!-------------------------------------------------------------------------------------------->
/// <!--                  MyMedianSelection_OperationLoopAgain                  -->
/// <!-------------------------------------------------------------------------------------------->
/// @brief This macro can be used to repeat the complete operation loop.
/// <!-------------------------------------------------------------------------------------------->
```

```
/// This macro is called after every complete operation loop. It can be used to compute
/// a result in a first loop (for example average image) and use this result in a second
/// loop to compute the final result images.
/// <!----------------------------------------------------------------------------------------->
/// @return FALSE: don't repeat the operation loop and go ahead, TRUE = repeat the
///  operation loop
/// <!----------------------------------------------------------------------------------------->
int MyMedianSelection_OperationLoopAgain()
{
   if (PassCount==1)
        return TRUE;          // Run the loop again
    else
        return FALSE;         // Do not run the loop again
}
```

Appendix C

Flux Uncertainty Propagation

The average velocity over an entire cycle was calculated using data taken at 17 different times throughout the cycle. This was done using:

$$\overline{U} = \frac{1}{17}\sum_{i=1}^{N=17} U_i$$

U_i= Velocity field at each individual time

$\overline{U}$= Average velocity over time

The uncertainty in the average velocity is give by:

$$\Delta\overline{U}^2 = \frac{1}{17}\sum_{i=1}^{N=17} \Delta U_i^2$$

ΔU_i = Standard error in U_i given by the $RMS/\sqrt{30}$ calculated from 30 phase-locked vector fields.

The volumes of flow moving in and out of the control volume are given by:

$$Q_{in} = \iint \overline{U}_{k_{in}} dx_i dx_j \cong \sum_{k_{in}=1}^{N_{in}} \overline{U}_{k_{in}} \delta x_i \delta x_j$$

$$Q_{out} = \iint \overline{U}_{k_{out}} dx_i dx_j \cong \sum_{k_{out}=1}^{N_{out}} \overline{U}_{k_{out}} \delta x_i \delta x_j$$

i, j and k represent the directions relevant to the surface on the volume being analyzed. i and j represent the components in this plane, while k represent the out-of plane component.

$\overline{U}_{k_{in}}$ = The component of the average velocity going into a surface of the

control volume.

$\overline{U}_{k_{out}}$ = The component of the average velocity going out of a surface of the control volume.

δx = The distance between each velocity vector. This stays constant over the entire volume.

N_{in} = The number of vectors going into a surface of the control volume.

N_{out} = The number of vectors going out of a surface of the control volume.

The uncertainty in the volume flows in and out are given by:

$$\Delta Q_{in}^2 = \sum_{m_{in}=1}^{N_{in}} (\delta x_i \delta x_j \overline{U}_{k_{in}-m})^2 \left[\left(\frac{\Delta \overline{U}_{k_{in}-m}}{\overline{U}_{k_{in}-m}} \right)^2 + \left(\frac{\Delta \delta x_i}{\delta x_i} \right)^2 + \left(\frac{\Delta \delta x_j}{\delta x_j} \right)^2 \right]$$

$$\Delta Q_{out}^2 = \sum_{m_{out}=1}^{N_{out}} (\delta x_i \delta x_j \overline{U}_{k_{out}-m})^2 \left[\left(\frac{\Delta \overline{U}_{k_{out}-m}}{\overline{U}_{k_{out}-m}} \right)^2 + \left(\frac{\Delta \delta x_i}{\delta x_i} \right)^2 + \left(\frac{\Delta \delta x_j}{\delta x_j} \right)^2 \right]$$

$\Delta \delta x_i$ = the uncertainty in the grid spacing.

In the x and y directions $\Delta \delta x_i$ is given by the RMS of the fit from the calibration. Because of the self calibration steps, this value is approximately 0.002 pixels for each camera, which is negligibly small. The experiment was moved manually in the z-direction to take different planes of data. It was aligned with a scale that measures to the closest millimeter, which gives an uncertainty of ±0.1 mm for the grid spacing in the z-direction.

The total flux for the whole volume is then calculated by summing the total amount of flow going in and summing the total amount of flow going out of each surface.

$$total\_Q_{in} = \sum_{l=1}^{N_{sides}=6} Q_{in-l}$$

$$total\_Q_{out} = \sum_{l=1}^{N_{sides}=6} Q_{out-l}$$

These should theoretically be equal, but since there is error in the flow measurement this is not necessarily true. The uncertainty in the total flows in and out are given by:

$$\Delta total\_Q_{in}^2 = \sum_{l=1}^{N_{sides}=6} \Delta Q_{in-l}^2$$

$$\Delta total\_Q_{out}^2 = \sum_{l=1}^{N_{sides}=6} \Delta Q_{out-l}^2$$

Dissipation Uncertainty Propagation

In order to calculate the dissipation, it is necessary to take derivatives of the velocity components over space. The derivatives for this were calculated using a second order central finite-difference scheme:

$$\frac{\partial U_i}{\partial x_j} \cong \frac{\delta U_i}{\delta x_j} = \frac{U_i(x_j + \delta x_j) - U_i(x_j - \delta x_j)}{2\delta x_j}$$

δx_j = grid spacing

U_i = Measured velocity

It has been shown that the error in the derivative is a combination of the error due to uncertainty in the velocity measurement and the truncation error of the central difference scheme. In the equation below, the first term on the right is the truncation

error and the second term on the right is the noise error, due to the uncertainty in the velocity measurement.

$$\left|\frac{\partial \widetilde{U}_i}{\partial x_j} - \frac{\delta U_i}{\delta x_j}\right| = O(\delta x^2) + O(\Delta U_i/\delta x)$$

$\widetilde{U}_i$ = True velocity

U_i = Measured velocity

ΔU_i= Uncertainty in the measured velocity

For the central difference scheme used the truncation error and noise error are given by Foucaut and Stansislas (2002). The uncertainty in the derivative is given by the equation below with the truncation error as first term on the right and the noise error as the second term on the right.

$$\Delta \frac{\delta U_i}{\delta x_j} = \frac{\delta x_j^2}{3!}\frac{\partial^3 U_i}{\partial x_j^3} + 0.71\frac{\Delta U_i}{\delta x_j}$$

To determine the order of magnitude of the truncation error, the third derivative of the velocity with respect to space was approximated using:

$$\frac{\partial^3 U_i}{\partial x_j^3} \cong \frac{\delta^3 U_i}{\delta x_j^3} = \frac{U_i(x_j + 2\delta x_j) - 3U_i(x_j + \delta x_j) + 3U_i(x_j) - U_i(x_j - \delta x_j)}{\delta x_j^3}$$

These values were then substituted into the truncation error equation and compared to the values of the noise error. On average the value for the truncation error is approximately 15% of the value for the noise error. It was deemed that in comparison to the noise error, the truncation error was negligibly small, so it is neglected in this propagation.

Since the flow is incompressible, the dissipation was calculated using the equation below:

$$\varphi = \mu\left[2\left(\frac{\delta U}{\delta x}\right)^2 + 2\left(\frac{\delta V}{\delta y}\right)^2 + 2\left(\frac{\delta W}{\delta z}\right)^2 + \left(\frac{\delta V}{\delta x} + \frac{\delta U}{\delta y}\right)^2 + \left(\frac{\delta W}{\delta y} + \frac{\delta V}{\delta z}\right)^2 + \left(\frac{\delta U}{\delta z} + \frac{\delta W}{\delta x}\right)^2\right]$$

μ = dynamic viscosity

$U, V \text{ and } W$ = Three measured velocity components

$\delta x, \delta y \text{ and } \delta z$ =the distance between each velocity vector

The uncertainty in the dissipation at each point in a volume at each time is given by

$$\Delta\varphi^2 = \mu\left[\left(4\frac{\delta U}{\delta x}\right)^2\left(\Delta\frac{\delta U}{\delta x}\right)^2 + \left(4\frac{\delta V}{\delta y}\right)^2\left(\Delta\frac{\delta V}{\delta y}\right)^2 + \left(4\frac{\delta W}{\delta z}\right)^2\left(\Delta\frac{\delta W}{\delta z}\right)^2 + 4\left(\frac{\delta V}{\delta x} + \frac{\delta U}{\delta y}\right)^2\left(\Delta\frac{\delta V}{\delta x}\right)^2 + 4\left(\frac{\delta V}{\delta x} + \frac{\delta U}{\delta y}\right)^2\left(\Delta\frac{\delta U}{\delta y}\right)^2 + 4\left(\frac{\delta W}{\delta y} + \frac{\delta V}{\delta z}\right)^2\left(\Delta\frac{\delta W}{\delta y}\right)^2 + 4\left(\frac{\delta W}{\delta y} + \frac{\delta V}{\delta z}\right)^2\left(\Delta\frac{\delta V}{\delta z}\right)^2 + 4\left(\frac{\delta U}{\delta z} + \frac{\delta W}{\delta x}\right)^2\left(\Delta\frac{\delta U}{\delta z}\right)^2 + 4\left(\frac{\delta U}{\delta z} + \frac{\delta W}{\delta x}\right)^2\left(\Delta\frac{\delta W}{\delta x}\right)^2\right]$$

The dissipation was summed over the control volume to give a total dissipation at each point in time.

$$\varphi_{total} = \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\sum_{k=1}^{N_z}\varphi_{ijk}\,\delta x\,\delta y\,\delta z$$

N_x= number of values in the x direction

N_y= number of values in the x direction

N_z= number of values in the x direction

The uncertainty in this calculation is given by:

$$\Delta\varphi_{total}{}^2 = \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\sum_{k=1}^{N_z}(\varphi_{ijk}\,\delta x_i\,\delta x_j\,\delta x_k)^2\left[\left(\frac{\Delta\varphi_{ijk}}{\varphi_{ijk}}\right)^2+\left(\frac{\Delta\delta x_i}{\delta x_i}\right)^2+\left(\frac{\Delta\delta x_j}{\delta x_j}\right)^2+\left(\frac{\Delta\delta x_k}{\delta x_k}\right)^2\right]$$

The average of the total dissipation over time was then calculated.

$$\bar{\varphi}_{total} = \frac{1}{17}\sum_{i=1}^{N=17}\varphi_{total-i}$$

The uncertainty in the average dissipation is given by:

$$\Delta\bar{\varphi}_{total} = \frac{1}{17}\sum_{i=1}^{N=17}\Delta\varphi_{total-i}$$

Bibliography

Alben, S., Spears, K, Garth, S., Murphy, D & Yen, J. (2010). Coordination of multiple appendages in drag-based swimming. *J. R. Soc. Interface* **7**, 1545-1557.

Alexeev, A., Yeomans, J. M. & Balazs, A. C. (2008). Designing Synthetic, Pumping Cilia That Switch the Flow Direction in Microchannels, *Langmuir*, **24** (21), 12102 12106.

Bart, S. F., Tavrow L. S., Mehregany M. & Lang J. H. (1990). Microfabricated electrohydrodynamic pumps, *Sensors and Actuators A: Physical*, **21**(1), 193–197.

Bourouina, T., Bosseboeuf, A. & Grandchamp, J. P. (1997). Design and simulation of an electrostatic micropump for drug-delivery applications *J. Micromech. Microeng.* **7**, 186–188.

Childress, S. & Dudley, R. (2004). Transition from ciliary to flapping mode in a swimming mollusk: flapping flight as a bifurcation in Re_{ω}, *J. Fluid Mech.*, **498**, 257-288.

Dittrich, P.S., Tachikawa, K., & Manz, A. (2006*).* Micro Total Analysis Systems. Latest Advancements and Trends, *Anal. Chem.*, 2006, **78** (12), 3887–3908.

Foucaut, J. & M. Stansislas (2002). Some considerations on the accuracy and frequency response of some derivative filters applied to particle image velocimetry fields, *Meas. Sci. Technol*. **13**, 1058-1071.

Hussong, J., Breugem, W-P., & Westerwee,l J. (2010). A continuum model for flow induced by metachronal coordination between beating cilia. `Manuscript submitted for publication.`

Jang, J. S. & Lee, S. S. (2000). Theoretical and experimental study of MHD (magnetohydrodynamic) micropump, *Sensors Actuators* A: Physical, **80**(1), 84–89.

Jiang, L. *et al* (2002). Closed-loop electroosmotic microchannel cooling system for VLSI circuits, *IEEE Trans. Compon.Packag. Technol.* **25**(3), 347–355.

Laser, D. J. & Santiago, J. G. (2004). A review of micropumps, *J. Micromechanics and. Microengineering* **14**(6), 35-64.

LaVan, D. A., McGuire, T. & Langer, R. (2003). Small-scale systems for *in vivo* drug delivery, *Nature Biotechnology* **21**, 1184 – 1191.

Lawson, N. J., & Wu, J. (1997). Three-dimensional particle image velocimetry: Error analysis of stereoscopic techniques. *Meas. Sci. Technol.* **8**, 894-900.

Medtronic MiniMed Inc. Product information, Medtronic Minimed 2007 Implantable Insulin Pump System, Retrieved January 2011 from <www.minimed.com>

Melcher, J. R. (1981). *Continuum Electromechanics.* Cambridge, MA: MIT Press.

Prasad, A. K. (2000). Stereoscopic particle image velocimetry. *Exp. Fluids.* **29**, 103-116.

Prasad, A. K. & Jenson, K. (1995). Scheimpflug stereocamera for particle image velocimetry in liquid flows. *Appied Optics,* **34**(30), 7092-7099.

Richter, A. & Sandmaier, H. (1990). An electrohydrodynamic micropump, *Micro Electro Mechanical Systems, 1990. Proceedings, An Investigation of Micro Structures, Sensors, Actuators, Machines and Robots. IEEE* , 99-104.

Sensenig, A., Shultz, J., & Kiger, K. T. (2009). The rowing-to-flapping transition: ontogenetic changes in gill-plate kinematics in the nymphal mayfly *Centroptilum triangulifer* (Ephemeroptera, Baetidae), *Zool. J. Linnean Soc.*, **98**(3), 540-555.

Sensenig, A., Kiger, K. T. & Shultz, J. (2010). Transitional ventilation mechanisms in nymphal mayfly *Centroptilum triangulifer*, *Journal of Experimental Biology,* **213**(19), 3319-3331.

Squires, T.M., Messinger, R.J. & MAnalis, S.R. (2008). Making it stick: convection, reaction and diffusion in surface-based biosensors, *Nature Biotechnology*, **26**(4), 417-426.

Thomas, L.J. & Bessman, S.P. (1975). Prototype for an implantable micropump powered by piezoelectric disk benders, *Trans. Am. Soc. Artif. Organs,* **21**, 516–520.

Timonen, J. V. I., Johans, C., Kontturi, K., Walther, A., Ikkala, O. & Ras, R. H. A., (2010). "A Facile Template-Free Approach to Magnetodriven, Multifunctional Artificial Cilia," *Applied Material Interfaces*, **2**(8), 2226–2230.

van de Pol, F.C.M., van Lintel, H.T.G., Elvenspoek, M., & Fluitman, J.H.J. (1990). A thermopneumatic micropump based on micro-engineering techniques, *Sensors and Actuators A: Physical,* **21-23**(1), 198–202.

van Lintel, H. T. G., van de Pol, F. C. M. & Bouwstra S. (1988). A piezoelectric micropump based on micromachining of silicon, *Sensors and Actuators* **15,** 153–67

Vitko, J., Franz, D.R., Alper, M., Biggins, P.D.E., Brady, L.D., Bruckner-Lea, C., Burge, H.A., Ediger, R., Hollis, M.A., Laughlin, L.L., Mariella, R.P., McFarland, A.R. & Schaudies, R.P. (2004). *Sensor Systems for Biological Agent Attacks:*

Protecting Buildings and Military Bases, The National Academies Press, Washington, DC.

Walker, J. A. (2002). Functional morphology and virtual models: Physical constraints on the design of oscillating wings, fins, legs, and feet at intermediate Reynolds numbers, *J. Integ. and Comp. Biol.*, **42**, 232-242.

Wang, X., Wang, S., Gendhar, B., Cheng, C., Byun, C.K., Li, G., Zhao, M., Liu, S. (2009). Electroosmotic pumps for microflow analysis, *Trends in Analytical Chemistry*, **28**(1), 64-74.

Woias, P. (2005). Micropumps—past, progress and future prospects, *Sensors and Actuators B: Chemical*, **105**(1), 28-38.