Developing a GPU based Real-Time Particle Image Velocimetry System for Active Flow Control

by

Ivy Olivia Friesen

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering
University of Alberta

© Ivy Olivia Friesen, 2024

Abstract

The objective of this work was to create a Graphical Processing Unit (GPU)-based Real-Time Particle Image Velocimetry (RT-PIV) system for use in Active Flow Control (AFC) of a Turbulent Boundary Layer (TBL), and to evaluate the effect of effect of the flow control on the boundary layer.

The experiment involved using the Cross-Correlation (CC) method utilizing Fast Fourier Transform (FFT) to compute particle displacement between two image pairs. The FFT method was chosen for its low computational cost, and ability to implement calculations on the GPU. Computations were done in MATLAB using a NVIDIA GeForce GTX 1080 Ti Graphical Processing Unit (GPU). The calculated RT-PIV vector fields were size 3×30, with 3 vectors in the streamwise direction and 30 vectors in the wall-normal direction. The average time for the RT-PIV code to process on the GPU was found to be 1.20 ms, with a standard deviation of 0.07 ms. The RT-PIV AFC system was tested at 25 Hz in the University of Alberta wind tunnel. The time limiting factor for this test was the laser, the New Wave Research Gemini 30 Hz PIV Nd:YAG Laser system. A voice-coil actuator was used as the active surface actuator for flow control. This actuator was mounted flush to the floor of the wind tunnel and had a range of motion of $\pm$ 3 mm displacement. One streamwise vector from each vector field was used as the input to the controller. The calculated particle velocity between two frames was filtered using a first order Butterworth filter and a proportional control law was used to determine the signal sent to the actuator. This filter was used to attenuate higher frequency noise from the measured velocity.

A parametric study was done on varying the filter cutoff frequency (5 Hz or 10

Hz) of the lowpass Butterworth filter, as well as the location of the vector used for flow control (8 mm or 11 mm above the wind tunnel floor). The Reynolds shear stress plots for each of the four experiment iterations were plotted and compared with and without actuation. The trial using a filter cutoff frequency of 5 Hz and input vector 11 mm from the wind tunnel floor showed the most promising results with a decrease in Reynolds shear stress when actuation was occurring, however, more tests are needed to ensure these results are statistically significant.

Future work includes using a higher frequency, higher powered laser for better PIV image results, using a NVIDIA RTX 3090 GPU for faster algorithm speed, using an actuator with higher range of motion, and utilizing different control laws.

Preface

This thesis is an original work by Ivy Friesen. No part of this thesis has been previously published.

Specific contributions include reimplementing the existing PIV code on a GPU to improve its execution speed (up to 500 Hz) such that the overall flow control system can be used in the University of Alberta wind tunnel, experimentally implementing and testing the flow control system and its goal to reduce surface drag within the wind tunnel by reducing the fluctuating component of velocity, and evaluating the effects of the flow control method by examining the changes to the Reynolds stresses downstream of the location of the flow control actuator.

Acknowledgements

Firstly, I would like to thank my supervisors, Dr Martin Barczyk and Dr Sina Ghaemi. Without their guidance and knowledge this project would not have been possible.

I would like to thank Findlay McCormick for his work with the RT-PIV algorithm and for all his help with PIV setup. Rick Conrad of the Mechanical Engineering electronics shop for his help with the camera input and for suggesting the Teensy Microcontroller. Bradley Gibeau for his work creating and tuning the voice-coil actuator. Sen Wang for his knowledge of PIV and laser setup. Carson Plamondon for all his help with experiment setup and data collection. Daniel Aldrich for his work creating the University of Alberta Latex thesis template. Victor Chen at Mathworks for troubleshooting connecting the Basler camera to MATLAB.

I would like to thank everyone who read over my thesis including Isobel Tetreau and Jakub McNally, and my lab mates for always encouraging me. Finally, I would like to thank the support of my friends and family.

Table of Contents

List of Tables

List of Figures

List of Symbols

Latin

$< uu >$ streamwise component of Reynolds stress

$< uv >$ components of Reynolds shear stress

$< vv >$ wall-normal component of Reynolds stress

B linear coefficient from linear regression

c_0 minimum controller output

C_{out} output of the controller

d_p diameter of a particle

e tracking error

f_c cutoff frequency

f_s sampling frequency

i x location within an IW

j y location within an IW

K gain value of the controller

L characteristic length of a flow

m mass

p particle pixel displacement between paired frames

R^2 coefficient of determination

Re Reynold's Number

Stk diameter of a particle

t time

U streamwise fluid velocity

u an instantaneous streamwise particle velocity

u_τ frictional velocity

V voltage

V wall-normal fluid velocity

v an instantaneous wall normal particle velocity

v_∞ free stream fluid velocity

x distance in streamwise direction

y distance in wallnormal direction

Greek

Δt time between two paired laser pulses

Δx displacement in x between two paired image frames

Δy displacement in y between two paired image frames

δ_{99} 99% boundary layer thickness

λ viscous length scale

μ dynamic viscosity of a fluid

ν kinematic viscosity of a fluid

ρ density of a fluid

ρ_{air} density of air

σ stress

τ_f time characteristic

τ_p response time of a particle

τ_w shear stress of a fluid

Abbreviations

AFC Active Flow Control.

CUDA Compute Unified Device Architecture.

FFT-CC Fast Fourrier Transform Cross Correlation.

FOV Field of View.

FPGA Field Programmable Gate Array.

FPS Frames per second.

GPU Graphical Processing Unit.

IDE Integrated Development Environment.

IW Interrogation Window.

ML Machine Learning.

OS Operating System.

PDE Partial Differential Equation.

PID Proportional Integral Derivative.

PIV Particle Image Velocimetry.

ROI Region Of Interest.

RT-PIV Real-Time Particle Image Velocimetry.

TBL Turbulent Boundary Layer.

TTL Transistor-Transistor Logic.

USB Universal Serial Bus.

Chapter 1

Introduction

1.1 Motivation

Decreasing energy consumption is a common goal for many researchers. One area a lot of energy is consumed is energy spent to overcome drag. Any fluid moving relative to a solid will have zero velocity at the surface of the solid, known as the no-slip boundary condition. The resulting thin layer of fluid at the wall with a large velocity gradient is called the boundary layer [1]. A boundary layer forms due to the fluid's viscosity, which is a fluid's resistance to movement [1], which causes drag. Whenever a fluid moves relative to any solid, drag is produced due to the fluid's viscosity.

The boundary layer is of great interest to researchers, especially methods to decrease drag, and therefore decrease energy expended or energy lost due to friction. Some past methods to decrease drag include streamlining shape, which reduces separation of fluid at an object by aligning its shape with predicted flow streamlines [1], or using a chemical solution, which reduces drag by reducing wall shear stress [2]. Some examples of drag reducing mechanisms include the rounded streamlined shapes of cars [3], the shape and camber of airplane wings [4], or rounding curves in pipes [5], adding polymers or drag-reducing agents to pipes [2]. Another method to reduce drag is utilizing Active Flow Control (AFC), which senses and responds to the flow characteristics as they occur (for example, responding to velocity fluctuations). The

premise of AFC is that small externally sourced disturbances at the wall of a flow field (for example, disturbances caused by moving an actuator) within the boundary layer can result in changes to large scale flow characteristics [6]. The driving goal of this research is to create a proof of concept, real-time active control for an airflow with a turbulent boundary layer. A major goal of AFC is to reduce drag within the flow [7]. Researchers are investigating using AFC to improve energy consumption within the transportation industry such as decreasing the drag on trucks [7] and aircraft [8] and thus improve environmental effects of these industries by sensing and responding to flow parameters.

RT-PIV (real-time particle image velocimetry) involves running PIV calculations at speeds approaching the camera frame rate, allowing the velocity field vectors to be used for active flow control. PIV has historically been used for post-processing data, since calculations are computationally time consuming. In order to apply control schemes to the flow, it is necessary to obtain flow information in real time. In the research conducted in this thesis, the PIV estimates were computed using the Cross-correlation (CC) method by using Fast Fourier Transform (FFT). The 2D FFT can be used to calculate velocity by calculating the particle displacement in a small region of a PIV image, called an Interrogation Window (IW). In order to process PIV calculations sufficiently fast, while minimizing computation time, a Graphics Processing Unit (GPU) was utilized. GPUs are ideal for applications with many parallel calculations, such as image processing. To take advantage of the GPU, the image processing code was parallelized such that the cross-correlation calculations for different interrogation windows in an image frame could be completed simultaneously. This was done by arranging IWs in a three-dimensional matrix and performing calculations on the entire matrix. A voice-coil actuator was mounted on the flow surface, which allowed for the continuous surface at the wall to be deformed, thus influencing the flow.

1.2 Thesis Objectives

The objective of this research was to implement a GPU based RT-PIV system for velocity estimation coded in MATLAB. This makes flow control in the turbulent flow in the wind tunnel possible. A previously built voice coil was used as the flow actuator, consisting of a deformable membrane flush with the floor to ensure a continuous surface. The downstream effects of the flow control were then evaluated in terms of the flow boundary layer profile and Reynolds Stresses. The velocity vectors obtained from the upstream flow using PIV calculations were used as inputs to the proportional controller driving a voice-coil actuator. The complete system was tested in the University of Alberta's wind tunnel.

1.3 Statement of contributions

- Reimplementing the existing PIV code on a GPU to improve its execution speed (up to 500 Hz) such that the overall AFC system can be used in the University of Alberta wind tunnel. This was done by parallelizing the required FFT calculations, instead of completing computations sequentially.

- Experimentally implementing and testing the AFC and its goal to reduce surface drag within the wind tunnel by reducing the fluctuating component of velocity.

- Evaluating the effects of the flow control method by examining the changes to the Reynolds stresses downstream of the flow control actuator. The largest measured decrease in Reynolds stress $< uu >$ was found to be up to 20% in the trial with a filter cutoff frequency of 5 Hz and input to the control code location 11 mm from the wind tunnel floor.

1.4 Outline

This chapter (Chapter 1) describes the motivation and objective for this project, as well as a statement of contribution.

Chapter 2 contains background information including explanation of boundary layers and turbulent flows, PIV and the fast Fourier transform cross-correlation technique, a review of existing related RT-PIV work and identification of knowledge gaps. It also reviews active flow controls and methods used for this system.

Chapter 3 describes the hardware and software components of this system, as well as an overview of the system setup. The main hardware used for this system was the Basler a2A1920-160umBAS Basler ace 2 USB 3.0 camera, two Teensy LC microcontrollers, a NVDIA GeForce GTx 1080 Ti GPU, and the New Wave Gemini 30 Hz PIV Nd: YAG Lasers system. The main software utilized for PIV calculations was MATLAB, and the Teensydruino was used to program the Teensy board. Chapter 3 also describes the codes used including the RT-PIV code and the active control code.

Chapter 4 describes the results of the experiment and a comparison of the velocity vectors before and after flow control. The boundary layers and Reynolds stresses were examined and compared. It also contains a time-summary for how long each component of the experiment took.

Finally, Chapter 5 summarizes the conclusions and limitations of the work completed, and recommendations for future work.

Chapter 2

Background

This chapter outlines the background information and highlights some of the relevant literature. This chapter will discuss the turbulent boundary, define the requirements for PIV (Particle Image Velocimetry), and review relevant past RT-PIV (Real-time Particle Image Velocimetry) and AFC (Active Flow Control) studies. It will also detail relevant information on evaluating the Turbulent Boundary Layer (TBL) and Reynolds Stresses.

2.1 Boundary Layers in Turbulent Flows

The Reynolds number, Re, a dimensionless quantity that can be used to characterize the nature of a flow named for Osborne Reynolds.

$$Re = \rho v_\infty L/\mu \tag{2.1}$$

Reynolds number is defined as the ratio of inertial forces to viscous forces within a moving fluid. Reynolds number is a function of the density of the fluid, ρ, its free stream fluid velocity, v_∞, its dynamic viscosity, μ, and the characteristic length of the flow, L [1].

In this flow, the characteristic length L, is defined as the streamwise (the streamwise direction is the direction of the fluid flow) length from the start of the boundary layer to the point of the region of interest (where the RT-PIV measurements take place).

One method to determine where the boundary layer starts is to place a trip wire [9].

Flow can be classified as laminar (smooth layers of fluid), turbulent (highly disordered flow with many irregular velocity fluctuations) or transitional (between laminar and turbulent) [1]. At a large Reynolds number, the inertial forces dominate the viscous forces in the fluid. This causes large fluctuations within the fluid, and a disorganized, chaotic, turbulent flow. Likewise, at lower Reynolds numbers, the viscous forces are dominant, which causes the motion of the fluid to remain organized, or "in line", resulting in laminar flow [1].

Typically, fluid flowing past a solid will have zero velocity at the surface of the solid due to the non-zero viscosity of the fluid. This is referred to as the "no slip" boundary condition. This creates a velocity profile similar to the one shown in Figure 2.1, known as the boundary layer [1]. The fluid velocity at the surface of the wall is zero, and the velocity grows until the free stream velocity of the fluid is reached some distance from the surface.

A boundary layer forms when any solid object and any fluid move relative to each other. The boundary layer can also be found in internal flows, such as flow in a pipe [10] or external such as flow over a solid. Internal flows have applications such as oil transport or household water supplies. External flows have applications such as flow over a truck or airplane wing [11]. This means boundary layers have relevance in sectors including transportation, energy, etc. [7]. Prandtl's Boundary layer theory stated that even if a main flow can be modelled as inviscid, in the region close to the wall viscous effects are large [12]. Decreasing drag within these boundary layers can result in improvements in efficiency in these industries [13]. This is one major reason why researchers are interested in the boundary layer.

The streamlines within a boundary layer rarely remain coherent all the way downstream. In real fluid flows, the boundary layer often becomes turbulent. The primary focus of the research conducted in this thesis applies to turbulent boundary layers. A boundary layer is classified as laminar when $\mathrm{Re} < 10^5$, transitional when

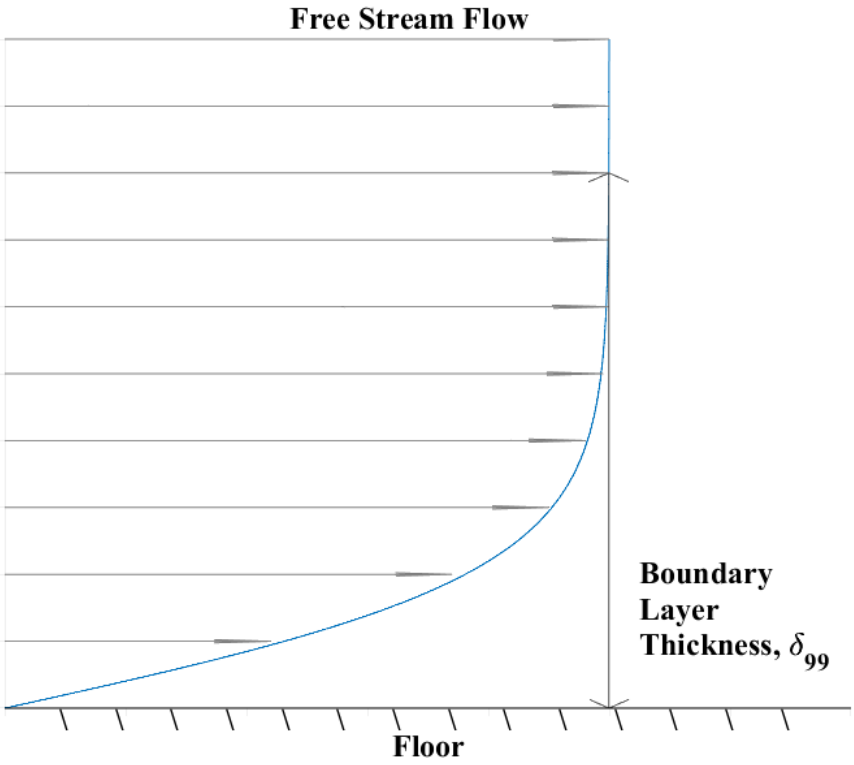


Figure 2.1: An illustration of a boundary layer. The profile is shown in blue, and streamwise velocity vectors are shown in grey. Note the zero-velocity condition at the floor due to the no slip boundary condition.

$10^5 < Re < 10^6$, and fully turbulent when $Re > 10^6$. [1].

The thickness of the boundary layer, δ is defined as the wall-normal (the wall normal direction is direction orthogonal to the wall) distance at which the fluid flow velocity reaches a specific percentage of the free stream velocity, usually specified as 99% [1]. For a turbulent boundary layer, one example of an empirical equation is the one-seventh power law to determine the thickness of the boundary layer at a distance, x, from the trip location of the flow. This equation is approximated as follows in Equation 2.2 [1] :

$$\delta = x(0.16)/(Re)^{1/7} \tag{2.2}$$

2.2 Mean Velocity Profile

Another way to describe the TBL is through the mean velocity profile [14]. A sample mean velocity profile at Re = 56,000 [15] is shown in Figure 2.2. First, the region nearest to the wall of the flow is called the viscous sublayer. In the viscous sublayer, shear stress dominates Reynolds Stresses [14]. After the viscous sublayer lies the buffer layer. The buffer layer is the transition region between the viscosity-dominated part of the flow (viscous sublayer) and the turbulence-dominated parts of the flow [14]. Next is known as the logarithmic layer. Together, the viscous sublayer, the buffer layer, and the logarithmic layer are called the inner layer, although the logarithmic layer is also part of the outer layer. The logarithmic layer is sometimes known as the overlap layer, as it is the transition between the inner and outer layers. Finally, the region at the end of the boundary layer is called the defect layer.

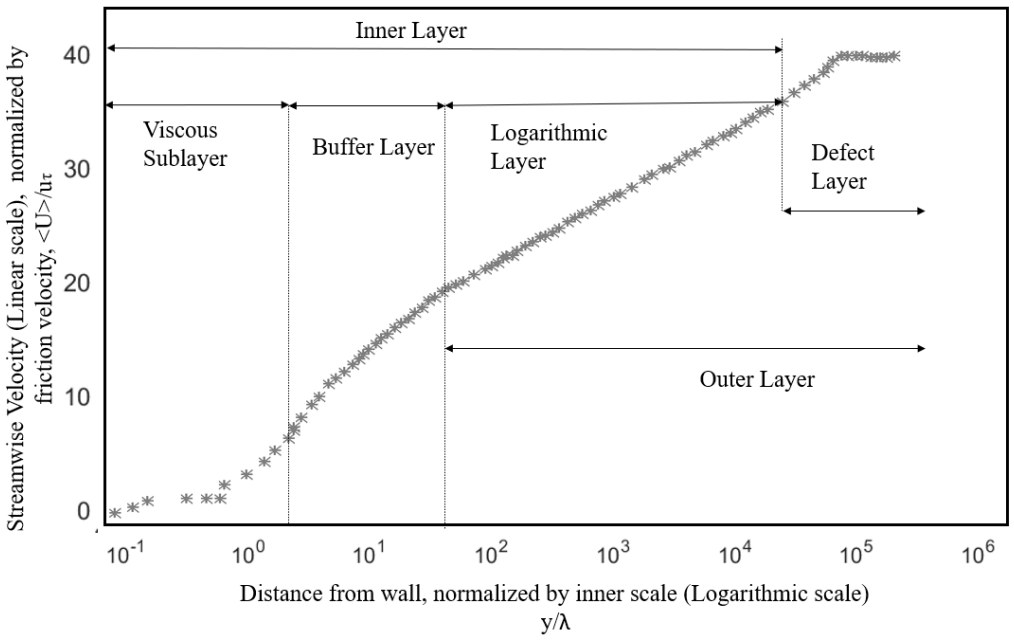


Figure 2.2: The mean velocity profile, which shows the regions of the TBL: viscous sublayer, buffer layer, logarithmic layer, and defect layer. Data from Re = 56,000 [15].

$$u_\tau = \sqrt{\tau_w/\rho} \tag{2.3}$$

The mean velocity profile shown in Figure 2.2 is normalized. The streamwise velocity, $< U >$, has been normalized by u_τ, the frictional velocity, which is calculated as shown in Equation 2.3 [16].

$$\tau_w = \mu \frac{\partial U}{\partial y}_{y=0} \tag{2.4}$$

τ_w, the shear stress at the wall, is calculated as shown in Equation 2.4.

$$\lambda = \nu/u_\tau \tag{2.5}$$

The distance from the wall, y, has been normalized by the viscous length scale λ, as shown in Equation 2.5. The viscous length scale λ, is calculated as shown in Equation 2.5.

2.3 Reynolds Stresses

Reynolds stress is called a stress because of its units, force divided by area [17]. It exists in flows with unsteady or fluctuating velocity components. Three components of Reynolds stresses will be calculated, $< uu >$, $< uv >$, and $< vv >$. These are elements of a symmetric second-order stress tensor [14]. In this experiment, only 2D components are calculated (streamwise and wall-normal), since 2D PIV is used.

$$\delta\sigma = \rho < uv > \tag{2.6}$$

Reynolds stress have a similar effect as viscous stress, therefore, decreasing the fluctuating wall-normal component of velocity within the fluid would decrease the Reynolds stresses within the fluid, and this would decrease the drag on the flow. $< uu >$ and $< vv >$ are normal stress components and $< uv >$ is a shear stress component [14]. Because Reynolds stress are derived from the fluctuating portions of

fluid velocity, decreasing Reynolds stress decreases turbulence within the fluid, and thus decreases the frictional drag [18].

2.4 PIV Summary and Background

Particle Image Velocimetry (PIV) is a non-intrusive flow measurement technique. A brief overview of the PIV process is described in this section [19]. To conduct PIV, a fluid flow is first seeded with small, reflective particles which are selected to minimize sinking in the fluid that is under examination. An ideal particle would be neutrally buoyant, however, in practice it is not easy to find a particle that meets this requirement when air is the working fluid. Particles are selected so that they will follow the fluid without altering any of its properties [20]. A laser beam pulses on the flow to create a thin plane of light (sometimes called a laser sheet). This laser sheet is thin (about 1 mm thick), so that only the particles which pass directly through this illuminated plane become illuminated. A camera is placed orthogonal to the laser sheet's plane. A schematic of this setup is shown in Figure 2.3.

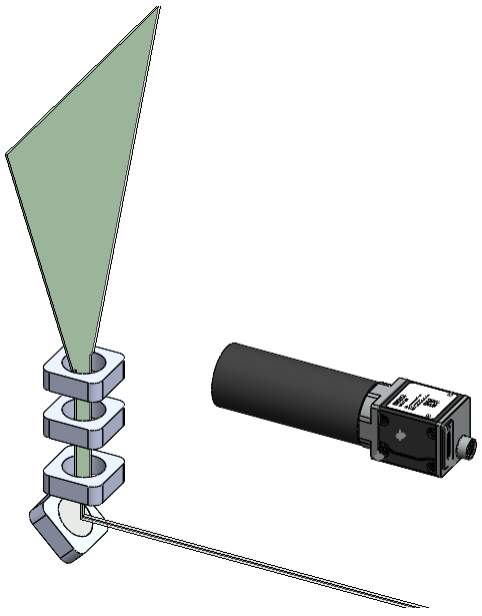


Figure 2.3: Diagram of a typical PIV setup, showing the camera position and the generation of the illuminated laser sheet.

By taking pairs of photographs of the flow at a known time shift, the velocity vectors of the particles in the laser sheet plane can be determined by calculating the distance particles move between each of the two frames in a set. One method of calculating this distance is the Cross Correlation (CC) method. This method will be detailed in Section 2.7. Figure 2.4 shows an example of a sub section of a pair of PIV images that would be used to calculate the velocity vectors of the particles.

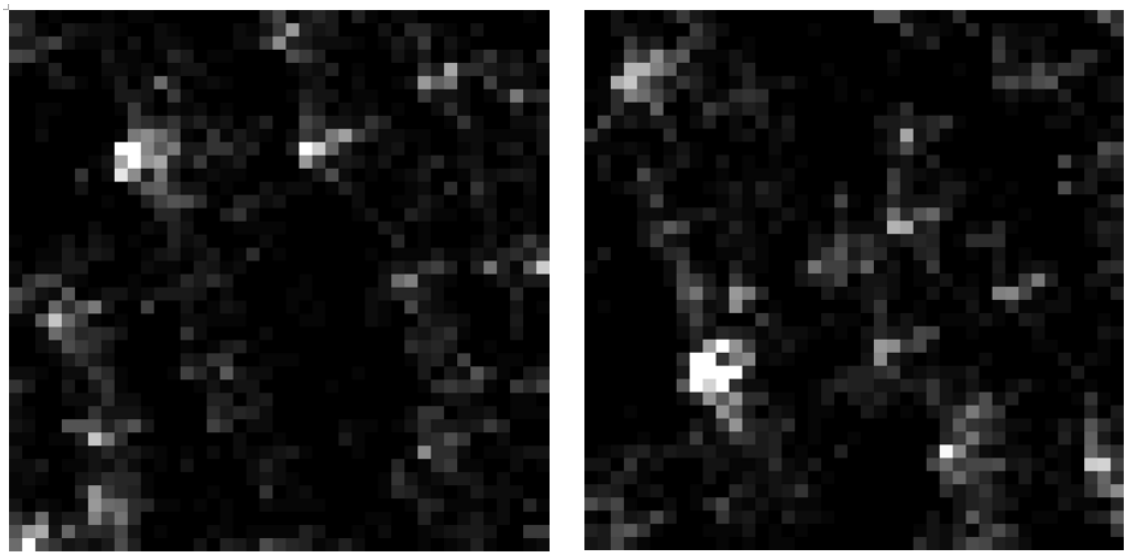


Figure 2.4: A sample of a section of a PIV image pair. It can be seen that the particle has moved downwards from image 1 to image 2.

The instantaneous streamwise velocity of the particles, u, can be approximated by determining the horizontal distance the particles travel in a single image pair, x, and the time between those frames, Δt:

$$u = x/\Delta t \tag{2.7}$$

Likewise, the instantaneous wall-normal velocity of the particles v can be approximated by determining the vertical distance the particles travel in a single image pair, y.

$$v = y/\Delta t \tag{2.8}$$

The time between two PIV frames (equal to the time between when the illuminating laser pulses occurred) is determined by the laser triggers. However, finding the distance individual particles travel in a frame is not trivial. Calculating the velocity vectors of the particles is a computationally intensive task. Historically, this has been done offline on collected data. With the advancement of computational power in the last decade, it is now possible to complete PIV calculations online [21], just as data is collected.

The first documented quantitative experiments using photography to visualize flow are attributed to Nayler and Frazer in 1917. They took 2D images of unsteady flow of particles around a cylinder on cinematic film, then marked particle location manually, using a pin. This was repeated for several frames to form streaks [22].

Another early known usage of particles to study flow is credited to Luwdig Prandtl and his colleagues in the late 1920s [23]. Prandtl's films were intended to illustrate flow separation but are now recognized as one of the first popular instances of PIV images.

1977 saw many publications on a novel technique for determining fluid flow. This technique, named Laser Speckle Photography (LSP), involved seeding a flow with particles and shining a ruby laser. The speckles were photographed in double exposures and the resulting patterns were used to determine the fluid flow. The main difference between LSP and PIV is that LSP arises from two speckle patterns (exposures) on a single film, and PIV works with image pairs [24]. In 1977 Grousson and Mallick published a technique that expanded the use of speckle photography from determining the instantaneous fluid velocity at a single point, to determine overall fluid flow pattern. At the same time, Barker and Fourney published a technique to map lines of constant velocity in a fluid flow. Later that year, Simpkins and Dudderar expanded the technique to work on dynamic fluid flows. LSP continued to evolve until the introduction of PIV in 1984 [22].

Modern PIV setups can make use of digital cameras capable of capturing PIV

images at a much higher frame rate [19] and higher resolution [25]. Additionally, the use of digital images allow for a single illumination on each frame (rather than two pulses on one frame as was done historically) [19]. This removes ambiguity in particle flow direction, information that is especially important in complex flows.

2.5 Particle Selection

The particles selected for PIV must have the correct physical properties as well as optical properties in order to be effective for PIV [19]. Since PIV is an indirect flow measurement technique (i.e., the particles are being tracked rather than the fluid itself), it is imperative that particles are selected so that they will accurately follow the fluid flow.

First, the particles should be selected to be close in density to the fluid where possible. If particles are much heavier than the fluid medium, errors in the calculated velocity will arise due to gravitational forces on the particles. Likewise, if the particles are much lighter than the fluid medium, differences will arise in the calculated velocity due to the buoyant forces on the particles. In flows (such as gas flows) where it is not possible to match the density of the particles with the density of the fluid, the particle diameter must be selected such that it is sufficiently small enough to minimize the gravitational effects. However, particles cannot be too small such that the camera will not be able to detect the light that is scattered by the particles. Therefore, smoke or fog are common choices for particles when gas is the working fluid for PIV [19].

The Stokes Number can be used to quantify the accuracy of the particle's ability to trace the flow [19]. Stokes Number, Stk, is defined as the ratio of particle response time to characteristic time scale of the flow.

$$\tau_p = d_p^2 \rho_p / 18\mu \tag{2.9}$$

The particle response time τ_p can be calculated knowing the particle diameter, d_p, the particle density, ρ_p and the fluid dynamic viscosity, μ, as shown in Equation 2.9.

$$\tau_f = \delta / v_\infty \tag{2.10}$$

The characteristic time scale in the boundary layer of a turbulent flow can be taken as the ratio of boundarylayer thickness, δ, and free stream velocity, v_∞[19], as shown in Equation 2.10.

$$Stk = \tau_p / \tau_f \tag{2.11}$$

Stokes Number can then be calculated, as shown in Equation 2.11. If the Stokes Number is below 0.1, the particle is said have appropriate ability to trace the flow [19].

The particle seeding density must also be large enough such that sufficient particles are visible on each frame for velocity calculations. Particles must also be mixed uniformly throughout the fluid to ensure both that there are particles in each image to track, and that particles are not overlapping and unable to be tracked in an image. Each interrogation window within an image must have sufficient particles for calculation. It is also recommended to keep the image background dark to create higher contrast between the particles and the background, as well as to more easily identify particles from noise [19]. Finally, particles must also be able to reflect the light source used.

2.6 Laser and Image Recording

Lasers are typically used as the light source for PIV. One reason for this is that they provide a monochromatic light source. This decreases the amount of noise in each image. The wavelength and particles are selected such that the particles will reflect a large portion of the laser's light. Another reason is that the laser beam can be accurately aligned to form a flat sheet of light. This is ideal for imaging as photographing a flat sheet decreases chromatic aberrations. A single slice of the fluid flow is illuminated, giving a picture of fluid flow in the illuminated plane. Additionally, short and energetic laser pulses are optimal for illuminating particle locations at a specific instance of time and to avoid motion blur [19].

Typically, two laser cavities are used in tandem and coordinated to pulse at a known time shift. The camera shutter is set to be open during each of the two laser pulses, so that each image pair will capture a small particle displacement. Lasers are triggered using a Transistor-Transistor Logic (TTL) signal. A TTL signal is a voltage signal with alternating high and low values.

Rapid development in digital imaging meant film PIV images were quickly replaced. Before the advent of digital photography, a single image with a double exposure would typically be used since the time to advance film would be too much for gaining meaningful results from two images. With the advent of digital technologies, taking two separate images is now preferred since flow direction can be determined and eliminates particle overlap. Digital cameras provide immediate images and are now the most often used to record PIV images. Two commonly used type of solid-state electric imaging sensors are the CCD (charged couple device) sensor, and the CMOS (Complementary Metal Oxide Semiconductor) sensor. Although image quality can be less on the CMOS sensor, it is preferred over the CCD sensor for real time applications because CMOS sensors are capable of a higher frame rate [19]. Conventionally, images are taken in grey scale or converted from colour to grey scale before cross-correlation

is done [26] for ease of cross-correlation.

2.7 Cross Correlation Algorithm

The most common method to calculate the displacement between the particles is utilizing a cross-correlation algorithm. The cross-correlation algorithm requires two single exposure images [27]. A single image with a double exposure would require using an autocorrelation method instead of a cross correlation method.

The cross-correlation algorithm first uses an interrogation window (IW) on each image. This is a small, equivalently located (the IW from each image must be chosen to be examining most of the same particles) square subsection of each image to preform the cross-correlation procedure on. Each PIV image is divided into IWs, as shown in Figure 3.13. The location of the interrogation window can be shifted by a predetermined number of pixels in the direction of the fluid flow to better capture the motion of the particles, as long as the initial pixel shift is accounted for again after correlation. This is to better correlate common particles instead and decrease the risk of calculating random correlations. The particle density should be five to ten particles per IW with the particles moving no more than a quarter the size of the IW between frames [28].

The cross correlation (CC) method utilizing fast-Fourier transform(FFT) is used to calculate the particle displacement in each interrogation window of the image pair. The fast-Fourier transform FFT algorithm computes the Discrete Fourier Transform (DFT) of a signal, which converts the signal from time domain to frequency domain [29]. DFT is utilized since only a finite number of data points (finite number of pixels) can be measured each frame [30]. FFT is one method to compute a DFT rapidly. Conversely, the inverse fast-Fourier transform (IFFT) converts the signal from frequency domain back to time domain.

The correlation theorem states that the cross-correlation of two functions is the product of the Fourier transform of one function multiplied by the complex conjugate

of the other function [31]. Using the correlation theorem is less computationally expensive than taking convolution of two images directly. This means that the cross-correlation of each interrogation window can be calculated by taking the inverse Fourier transform of the product of the first interrogation window and the complex conjugate of the second [19]. Computational cost can further be decreased by using the Fast Fourier Transform. To take advantage of the speed boost the interrogation window must be of a size that is of a power of 2.

$$\mathscr{F}[X] = \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} e^{(-2\pi i/m)jp} e^{(-2\pi i/n)kq} X_{j+1,k+1} \tag{2.12}$$

The 2D discrete Fourier transform, denoted as $\mathscr{F}$ is shown in Equation 2.12. The equation shows how the discrete signal, and n-by-m matrix X is transformed to $\mathscr{F}[X]$ through the summation of multiplication by an exponential function [32].

A simplified equation to demonstrate the implementation of the FFT-based CC algorithm is shown in Equation 2.13. Letting $\mathscr{F}$ denote the DFT and $\mathscr{F}^{-1}$ denote the inverse DFT, IW_1 and IW_2 denote the two respective IW stacks in the image pair, the cross correlation CC can be calculated. The complex conjugate is denoted as *.

$$CC = \mathscr{F}^{-1}(\mathscr{F}(IW_2)\mathscr{F}(IW_1)^*) \tag{2.13}$$

The FFT- based Cross-correlation calculations were implemented in MATLAB as shown in Listing 2.1. This process as implemented in MATLAB is illustrated below in a schematic shown in Figure 2.5.

Listing 2.1: Calculating the FFT Cross Correlation Matrix in MATLAB

```
FFTCC=real(fftshift(fftshift(ifft2(fft2(rot90(IW1,2)).*fft2(IW2)),1),2));

%real takes only the real portion of the results

%fftshift takes the FFT and shifts the zero-frequency component to the center
%of the array

%ifft2 takes the 2D inverse FFT

%fft2 takes the 2D FFT

%rot90 rotates the array counterclockwise. In this case it is rotated 2*90 degrees.
```

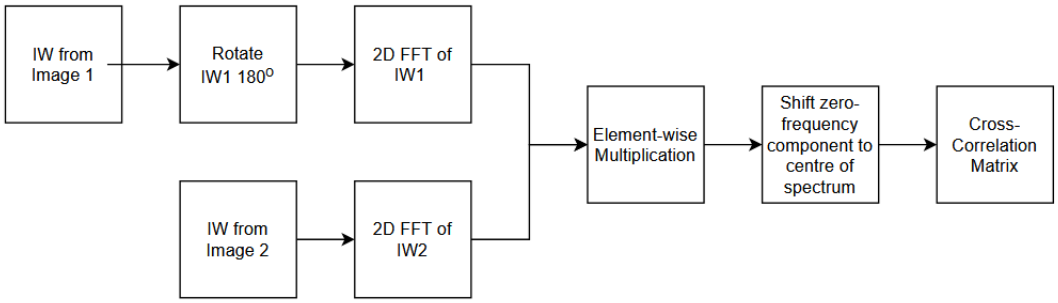


Figure 2.5: Process of calculating the FFT-CC matrix from the two IWs.

An example of a resulting cross correlation matrix is plotted in Figure 2.6 below. The location of the peak indicates the particle displacement vector between these two IWs [28].

Once the cross-correlation data is determined, a peak detection method is applied to determine the location of the peak to sub-pixel accuracy. Without using this step, the particle displacement precision would be limited to half a pixel. The peak finding method works by taking into account the values of the correlation itself [19]. An example of a peak detection method is the Gaussian Peak Finding method [33]. These formulas return the x and y values which define the location of the maximum correlation with sub-pixel accuracy. Letting R be the correlation matrix and i and j being the coordinates defining the maximum value in the correlation matrix, x and y can be found as shown in Listing 2.2.

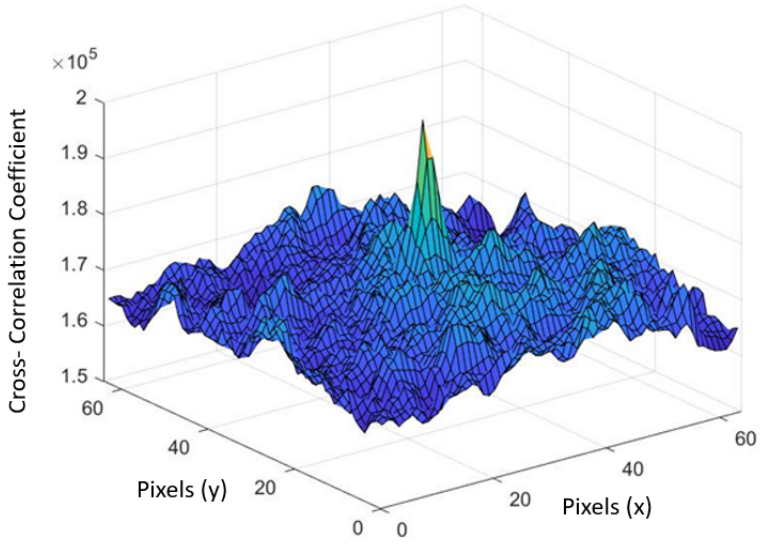


Figure 2.6: Example cross correlation graph for the correlation between two interrogation windows. This is for a 64-pixel by 64-pixel interrogation window. Note the large peak in the plot. The cross correlation contour is offset from zero, indicating there is a level of background noise in the IWs.

Listing 2.2: Gaussian Peak Finding in MATLAB

```
x = i + (lnR(i-1,j)-lnR(i+1,j))/(2lnR(i-1,j)-4lnR(i,j)+2lnR(i+1,j))

y = j + (lnR(i,j-1)-lnR(i,j+1))/(2lnR(i,j-1)-4lnR(i,j)+2lnR(i,j+1))
```

Finally, the velocity vector field of the image pair can be determined. Each pair of IWs yields one planar displacement vector. These are converted to velocity vectors by dividing by the time between the image pairs. This process is repeated for every incoming image pair as it is obtained. A schematic of the entire PIV calculation process used is presented in Figure 2.7.

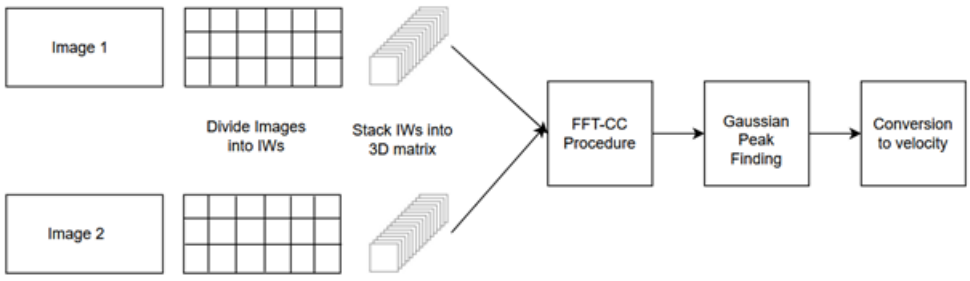


Figure 2.7: Overview of the calculations required to obtain the velocity vectors.

2.8 Real Time PIV

Traditionally, due to the computation expense, PIV calculations were done offline. However, due to advances in computational power, it is now possible to conduct PIV calculations as data is collected [21]. This is known as Real-Time PIV (RT-PIV)

In 1997, Carr et al. [28] used a digital PIV technique with interrogation areas of 32 pixels by 32 pixels. Images were 768 pixels by 480 pixels. The camera could run at 30 Hz, and their system calculated vectors using the cross-correlation method at rates of up to 15 Hz. The obtained vector fields were 47 by 29 entries in size. They used a Field Programmable Gate Array (FPGA) to process the data.

In 2002 McKenna et al. [34] evaluated the performance of several techniques including the FFT method. They showed computation times for an FFT algorithm, a Dynamic-FFT algorithm (algorithm with initial correlation pass, followed by a second pass using individually offset subimages), a Direct spatial domain algorithm (a subimage is correlated to every possible location in a search region), a Hybrid algorithm (a Dynamic FFT correlation is performed, followed by a direct spatial domain algorithm with limited offset), and a PTV algorithm (done by performing a coarse dynamic FFT DPIV pass over the entire image domain, and then matching particles between images). They used 640 pixels by 640 pixels images. They found that their FFT method could process 1369 vectors in 0.71 s with an IW of size 16 pixels by 16 pixels, and 361 vectors in 0.81 s with an IW of size 32 pixels by 32 pixels. Their FFT method produced faster results than their other three methods for both examined IW sizes. A 400 MHz PC running Linux was used.

In 2004 Schiwietz and Westermann [35] presented the first system that implemented FFT on GPUs. Their images were of size 1024 pixels by 1024 pixels and they used interrogation windows of size 16 pixels by 16 pixels, which delivered results at 10 fps.

In 2009, Muñoz et al. [36] used an FPGA implementing the direct cross correlation algorithm. They were able to process one IW (called Particle Image Pattern A) of

size 32 pixels by 32 pixels in 550 μs, or 1250 PIV vectors at 60 Hz.

In 2009, Kriezer et al. [37] presented a high speed particle tracking using FPGA. Their experiment had 1280 pixels by 1024 pixels images at 500 fps and IWs of 32 pixels by 32 pixels with 50% overlap and FFT cross correlation approach. Only the processed data was transferred to the computer on a high bandwidth network.

In 2009, Champagnat et al. [38] presented an algorithm that processed five 1376 pixels by 1040 pixels image pairs per second on a NVIDIA Tesla C1060 GPU. They used the iterative gradient based cross-correlation optimization as an alternative to the FFT-based method. This algorithm's computation time averaged 0.2s with similar accuracy to the FFT-based CC method used by commercial software, while being 50 times faster. This was later improved to 40 image pairs per second [39].

In 2012 Kobatake et al. [40] created a real-time micro-PIV system running at 2000 fps using images of 512 pixels by 512 pixels. They utilized a gradient based method to calculate velocity vectors, instead of the more computationally demanding cross-correlation method. A drawback of this method is that gradient based methods have a low maximum flow velocity this method works for compared to cross-correlation methods. They found that the time average max speed of 230 mm/s for their VFS-OF (variable-frame-straddling optical-flow) algorithm.

In 2014 Gauthier and Aider [41] performed closed loop feed forward control on flow over a backwards facing step. The images were size 2048 pixels by 1088 pixels, and they used the algorithm derived by Champagnat et al. on a Gforce GTX 580 graphics card. Vector fields were computed at speeds up to 224 Hz.

In 2023 McCormick [42] utilized a RT-PIV system to deliver vector fields at 7.35 Hz. They obtained 2368 pixels by 320 pixels images were transmitted via a frame grabber into Simulink Real-Time. A single pass cross-correlation algorithm using 64 pixels by 64 pixels was used to obtain a velocity vector field of size 37×5.

The results of this review are summarized Table 2.1.

Table 2.1: Past RT-PIV Works Reviewed

Ref	Hardware	Algorithm	IW size	Speed	Image Size
[28]	FPGA	Cross-correlation	32 × 32	15 Hz	768 × 480
[34]	400 MHz PC	FFT method	16 × 16	1928 vec/s	640 × 640
[34]	400 MHz PC	FFT method	32 × 32	446 vec/s	640 × 640
[34]	400 MHz PC	Dynamic-FFT	16 × 16	957 vec/s	640 × 640
[34]	400 MHz PC	Dynamic-FFT	32 × 32	221 vec/s	640 × 640
[34]	400 MHz PC	Direct spatial domain	16 × 16	141 vec/s	640 × 640
[34]	400 MHz PC	Direct spatial domain	32 × 32	10 vec/s	640 × 640
[34]	400 MHz PC	Hybrid	16 × 16	431 vec/s	640 × 640
[34]	400 MHz PC	Hybrid	32 × 32	110 vec/s	640 × 640
[34]	400 MHz PC	PTV	16 × 16	1164 vec/s	640 × 640
[34]	400 MHz PC	PTV	32 × 32	1104 vec/s	640 × 640
[35]	GPU	FFT	16 × 16	10 Hz	1024 × 1024
[36]	FPGA	Direct Cross-correlation	32 × 32	60 Hz	unknown
[37]	FPGA	FFT Cross-correlation	32 × 32	500 Hz	1280 × 1024
[38]	GPU	Iterative gradient cross-correlation	Various	40 Hz	1376 × 1040
[41]	GPU	Iterative gradient cross-correlation	r=10 pix	224 Hz	2048 × 1088
[42]	Frame Grabber	Cross-correlation	64 × 64	7.35 Hz	2368 × 320

2.9 Actuators and Flow Control

Although there are many possible methods to apply flow control, this section will only focus on the methods utilized in this experiment. A voice coil actuator will be utilized in the wall-normal direction to offset velocity fluctuations in the direction normal to the flow in order to reduce turbulent drag as discussed in Section 2.3.

A voice-coil actuator is a type of transducer than maps an electrical signal (i.e., voltage) to a physical quantity (i.e., displacement). This is the type of actuator utilized in this experiment. The use of a voice-coil actuator as a moving surface is one possible implementation for active flow control. Other types of actuators used for flow control include ailerons, trailing edge devices and vortex generators. Actuators can also be fluidic, such as pulsed blows or jets [43]. The actuator in this experiment is mounted in a way to minimize disturbances to the flow when it is not in use. More information about the actuator can be found in Section 3.5. The actuator is mounted flush with the surface of the flow channel to ensure the overall geometry of the flow channel remains unchanged. The surface of the actuator serves as a deformable portion of the wall bounding the flow. The actuator utilized in this experiment can move symmetrically such that the maximum positive wall-normal displacement and the maximum negative wall-normal displacement are the same distance from the wind tunnel floor. By moving the actuator outward, (e.g, below the wind tunnel floor) can induce an outward flow component. Likewise, moving the actuator inward, (e.g. above the wind tunnel floor) can induce an inward flow component [16]. This can be used to alter the flow characteristics in order to reduce turbulent drag.

In 1994, Choi et al. [44] explored concepts for active turbulence control in order to reduce drag. They examined velocities imposed in the three different planes of the flow. Their numerical simulation showed drag reduction at low a Reynolds number. They were able to predict 20-30% drag reduction when using the normal or span-wise wall velocity for control. Their flow control simulations were conducted with the

intent to guide future experiments.

Also in 1994, Laadhari et al. [45] showed experimentally that sinusoidally oscillating a flat plate between 2-10 Hz with an amplitude of 2.5 cm in the span-wise direction decreased turbulence. Their experiment was conducted in a a low-speed blower tunnel and found the relative reduction in u' of 45%, 34% in v', and 16% in w'.

In 2000, numerical simulation by Endo et al. [46] showed a 12% drag reduction when using feedback control by local wall deformation.

In 2003, Rathnasingham and Breuer experimented with using a span-wise array of synthetic jet actuators to reduce stream-wise velocity fluctuations by 30% [47]. They used shear stress sensors (flush-mounted hot wires oriented parallel to the main flow) as inputs to their control scheme.

In 2023 McCormick [42] used RT-PIV in conjunction with an active deformable surface of a streamwise array of sixteen actuation points to implement reactive control. Using v-control to decrease vortex shedding, it was found it could be decreased by up to 69%.

Also in 2023, Gibeau [16] used wall mounted pressure sensors to target very large scale motions in fluid flow. An actuator was used in both a laminar boundary layer and a turbulent boundary layer to produce streamwise velocity fluctuations up to one third freestream velocity.

The complexity and chaotic nature of turbulent flow makes it difficult to predict motions within the flow. This has complicated the studies of active flow control for turbulent flow.

2.10 Proportional Control

The proportional control law [48] is the mathematical formula that governs how the controller responds to the input. The proportional control law is a common, simple controller where the control signal is linearly proportional to the system error [48].

Here, C_{out} is the output of the controller (in this case, the output voltage of the microcontroller); K is the gain value; e is the tracking error, the difference between the current measured valued and the reference value (in this case, the difference between the current measured streamwise velocity input vector and the overall mean velocity vector); and $c0$ is the minimum system output value (in this experiment, the minimum voltage out).

$$C_{out} = K \times e + c0 \tag{2.14}$$

Chapter 3

Experimental Setup

The experiment was set up in the University of Alberta Wind tunnel, with two PIV setups. The first PIV setup was an RT-PIV setup upstream of the actuator. The second was an offline PIV setup downstream of the actuator. The speed of the flow is set using the speed of the wind tunnel fan and a flowmeter, and the flow is seeded with fog particles for PIV. Once this setup is ready, the real-time PIV (RT-PIV) and flow control process can begin. MATLAB was used to initiate the RT-PIV process. This upstream PIV took place 7.7 m downstream from the trip wire location and 0.04 m upstream from the leading edge of the actuator, as shown in Figure 3.1. MATLAB first connected to the Basler camera to set the camera parameters. After the camera initialization has completed and the camera is ready to collect images, MATLAB then sends a serial signal to the first Teensy board to start running.

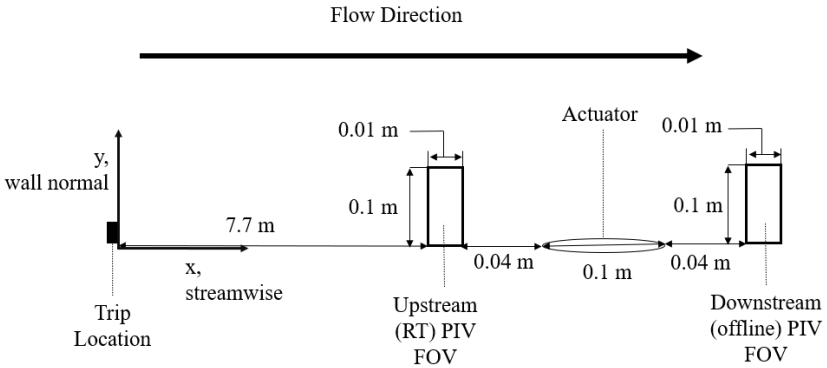


Figure 3.1: Schematic of experiment processes showing distance between components and coordinate system used. Image not to scale.

An overview of the RT-PIV (real-time particle image velocimetry) process used is given below. Figure 3.2 shows the components of the RT-PIV system. The technical details of each component are provided in the next sections.

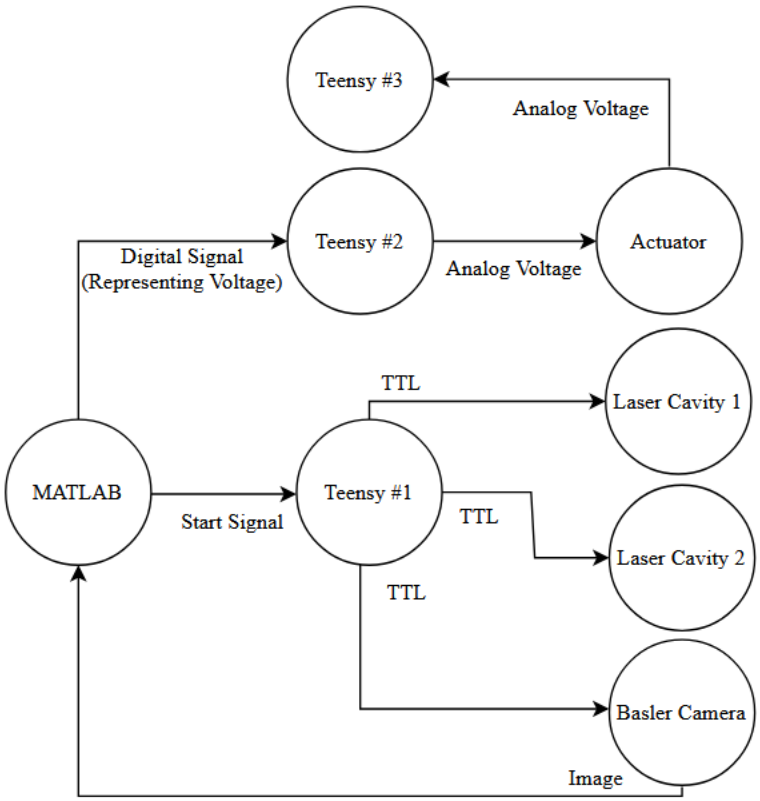


Figure 3.2: Schematic of experiment processes showing direction of information transferred

When the first Teensy board receives the serial signal from MATLAB, it triggers the lasers and the camera at a designated time. As each image is collected, it is sent to MATLAB via the camera's USB. A second image is collected to form the image pair required for two-frame PIV. MATLAB performs the required PIV calculations on the image pair and determines the streamwise and wall-normal particle velocity vector fields. The Teensy board continues to send the laser and camera trigger signals until it receives a signal from MATLAB to stop.

Once a vector field is calculated by MATLAB, the vector located at the region of interest (ROI) near the floor of the wind tunnel is analyzed. If the vector is within the

acceptable range for the flow speed, it will be mapped to a position on the actuator's range of motion. If it is outside that acceptable range, the actuator will hold the previous position for that iteration. Finally, MATLAB will send a digital signal (via a serial number that corresponds to the requested voltage) to the second Teensy. The Teensy will output an analog voltage and the actuator will move to the desired position. The same Teensy also samples the actuator output voltage to verify that the actuator has achieved the correct position and returns these values back to MATLAB. Finally, a second PIV system takes measurements 0.04 m downstream of the trailing edge of the actuator to measure the flow after flow control has been conducted. An identical hardware setup is used as the RT-PIV process, however the images are captured and saved for offline processing. An image of the overall hardware setup in the wind tunnel is shown in Figure 3.3.

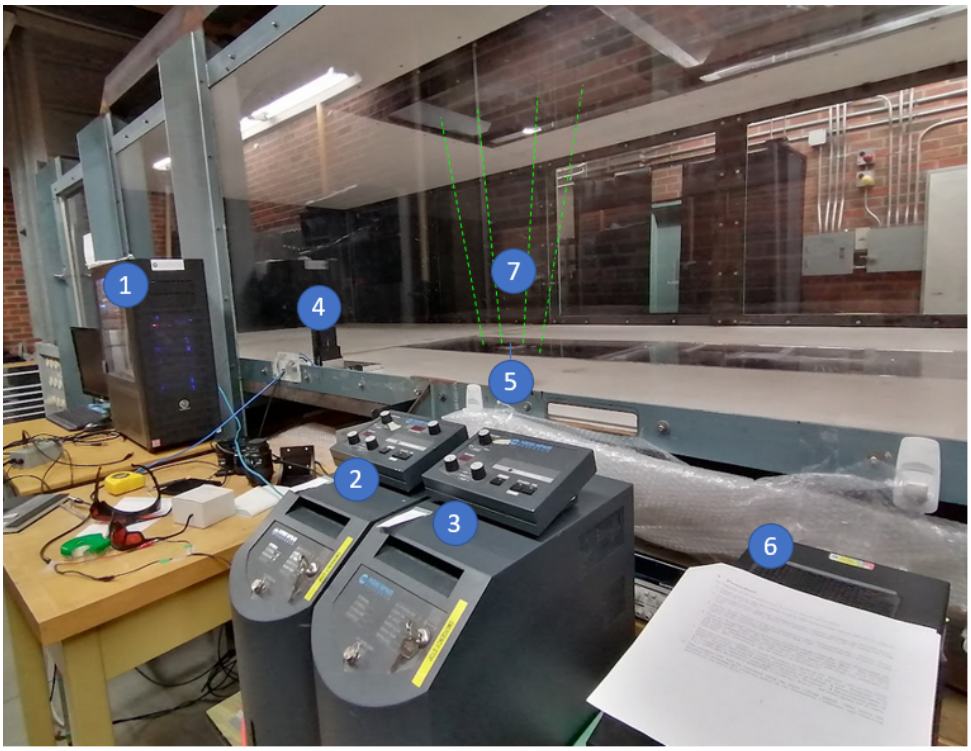


Figure 3.3: Image of system setup showing: 1. RT-PIV computer 2. First Laser Cavity 3. Second Laser Cavity 4. Camera 5. Actuator 6. Actuator Computer 7. Location of Laser Sheet (Approximate laser sheet and actuator shadow shown in green)

3.1 Hardware and Software

3.1.1 MATLAB

The RT-PIV code was written and ran in MATLAB R2022a. MATLAB was selected as it could interface with the camera, the GPU, the microcontrollers, as well as run control laws. MATLAB also ideal for running calculations on matrices, making it useful for running PIV calculations on many IWs (Interrogation Windows). MATLAB runs with minimal computational delay and by utilizing the GPU, was able to keep up with the processing speed required for real time flow control. Installation instructions for all software used in this setup are included in Appendix B.

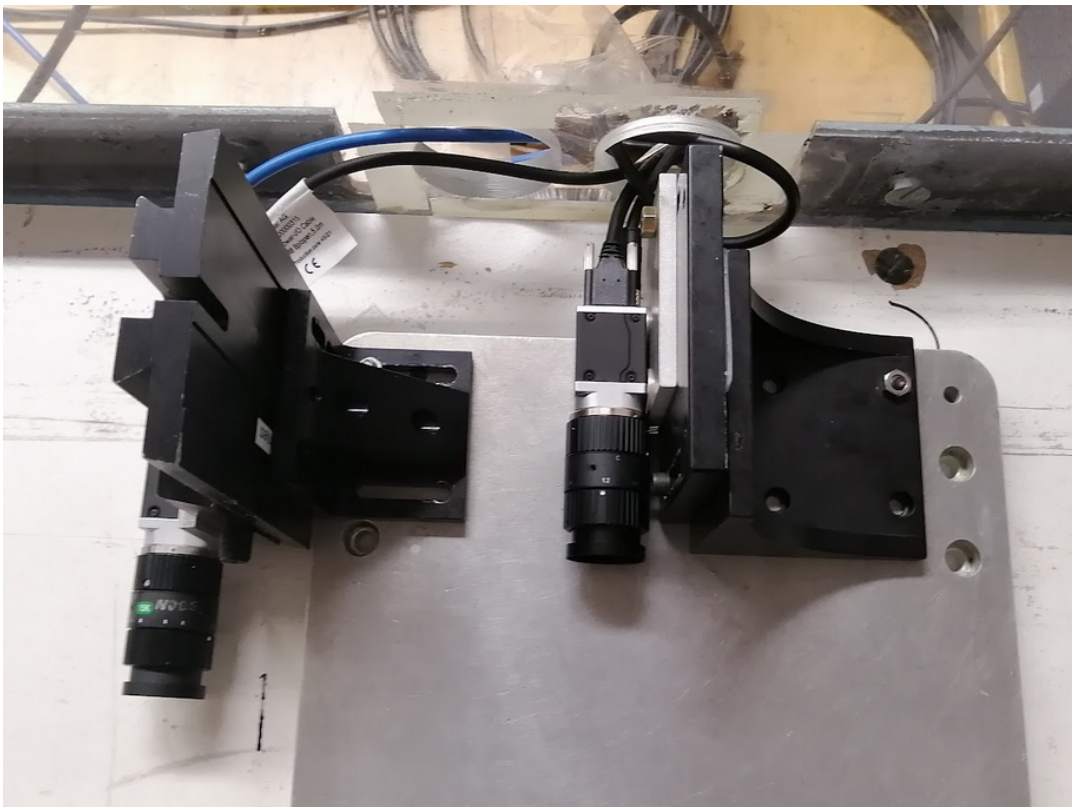
The Parallel Computing Toolbox was installed on MATLAB in order to do GPU computing. The MATLAB function `gpuArray` was used to designate arrays stored in GPU memory for performing parallel calculations.

`imaqtool`, a MATLAB function from MATLAB's Image Acquisition Toolbox, was used to setup and run the camera in MATLAB with the desired camera settings. MATLAB connected to the Basler camera using MATLAB's `imaqtool` and the GenICam support package. The GenICam support package allows MATLAB to utilize the Basler camera drivers from its software package, Pylon, as described in Section 3.1.3.

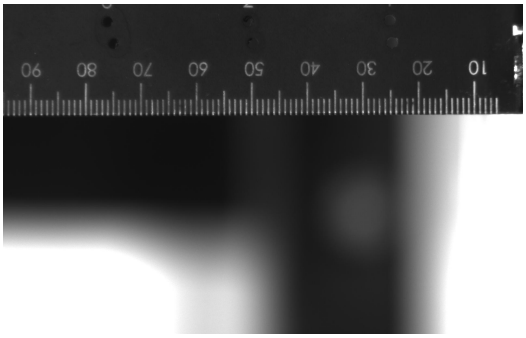
MATLAB also offers support for serial communication to hardware. MATLAB's Instrument Control Toolbox was used to communicate with the Teensy microcontrollers via serial communication.

3.1.2 Cameras

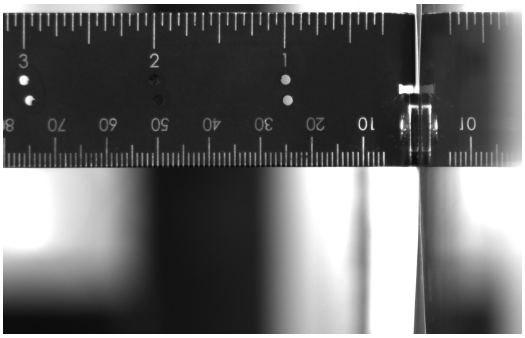
The cameras used for this experiment were Basler a2A1920-160umBAS Basler ace 2 USB 3.0 cameras. One camera was used for upstream data collection and the second camera was used for downstream data collection. Both cameras were mounted and squared to the inside of the wind tunnel to achieve a clear view of the particles illuminated by the laser sheet, as shown in Figure 3.4.



(a)



(b)



(c)

Figure 3.4: a) Basler cameras mounted in the wind tunnel with a 75 mm lens from Computar. The cameras are 0.2 m apart and squared to the laser sheet. b) Upstream camera's view of the (uncropped) region of interest, with a ruler placed at the location of the laser sheet. c) Downstream camera's view of the region of interest, with a ruler placed at the location of the laser sheet.

The Basler camera was selected primarily for its reasonable cost and ability to capture images at a high frame rate. Images are transferred to the computer at USB 3.0 speed, 5 Gbit/s. The manufacturer's given frame rate for this camera is

160 fps [49], however this frame rate can be increased by cropping the image size and changing the resolution. Mono 8 resolution was used for this experiment, the smallest bit depth available on this camera. The camera used in this PIV experiment had a CMOS sensor. The region of interest for this setup was 1920 by 200 pixels, cropped from the camera's full frame of 1936 by 1216 pixels. Using this ROI and setting the exposure time to a minimum, it was found that the camera frame rate could reach 880 fps.

The laser sheet (parallel to the lens surface) was measured to be 1.02 m away from the camera lens. A 75 mm Computar lens was used, at an aperture of f2.8. A ruler was placed at the location of the laser sheet and the size of the region of measurement in the upstream camera frame was determined to be 96 mm, shown in Figure 3.4. The resulting resolution was 50 μm / pixel. The downstream camera had a region of measurement of 100 mm, which results in a resolution of 52 μm / pixel.

3.1.3 Pylon

The Basler Pylon Camera Software Suite was installed to set up the Basler camera drivers. Pylon is an application which allows users to control the settings of the camera. The version used in this experiment was Pylon 7.1.0. The associated software drivers were required in order to use the camera through MATLAB's Image Acquisition Toolbox. The Basler Camera also required MATLAB's Support Package for GenICam Interface. The `imaqtool` settings used are tabulated in Table 3.1 below.

3.1.4 Laser and Laser Optics

The laser used for this experiment is the New Wave Research Gemini 30 Hz PIV Nd:YAG Laser system. The laser was triggered by sending a TTL (Transistor-Transistor Logic – a signal with alternating high and low values) signal from the Teensy to fire the flash lamp. The laser has a wavelength of 532 nm and delay of 180 μs from trigger to fire [50]. The flash lamp was set to external mode and the Q-switch

Table 3.1: List of `imaqtool` settings in MATLAB.

Tab	**Setting**	**Value**
Device Properties	Device Throughput Limit	Max (4149430400 bytes/s)
	Line Selector	Line Selector: Line 3 Line Mode: Output Line Source: Exposure Active
	Trigger Selector	Trigger Selector: Frame Start Trigger Mode: On Trigger Source: Hardware
	Exposure Time	1015 μs
Triggering	Hardware	
Region of Interest	Width 1920	
	Height 200	

was set to internal mode. The laser has a maximum frequency of 30 Hz. The laser was run at 25 Hz, which allowed for timing to be whole microseconds.

The laser was placed under the wind tunnel and a plane mirror was used to direct the beam upwards to the region of interest. Four lenses were used to create the laser sheet, as shown in Figure 3.5. The beam was first passed to a spherical lens with -50 mm focal length to converge the beam from approximately 5 mm diameter [50] to approximately 1 mm diameter. Next, the beam was passed through a spherical lens with a +100 mm focal point to slow the rate of convergence of the laser beam. Finally, the beam was passed through two cylindrical lenses, first with -50 mm focal point and second with -150 mm focal point to spread the beam stream-wise into a sheet. This sheet is parallel to the walls of the wind tunnel and perpendicular to the floor of the wind tunnel. The approximate distances between each of the lenses is also show in Figure 3.5. The laser optics lenses were moved until the sheet was thin and focused near the floor of the wind tunnel.

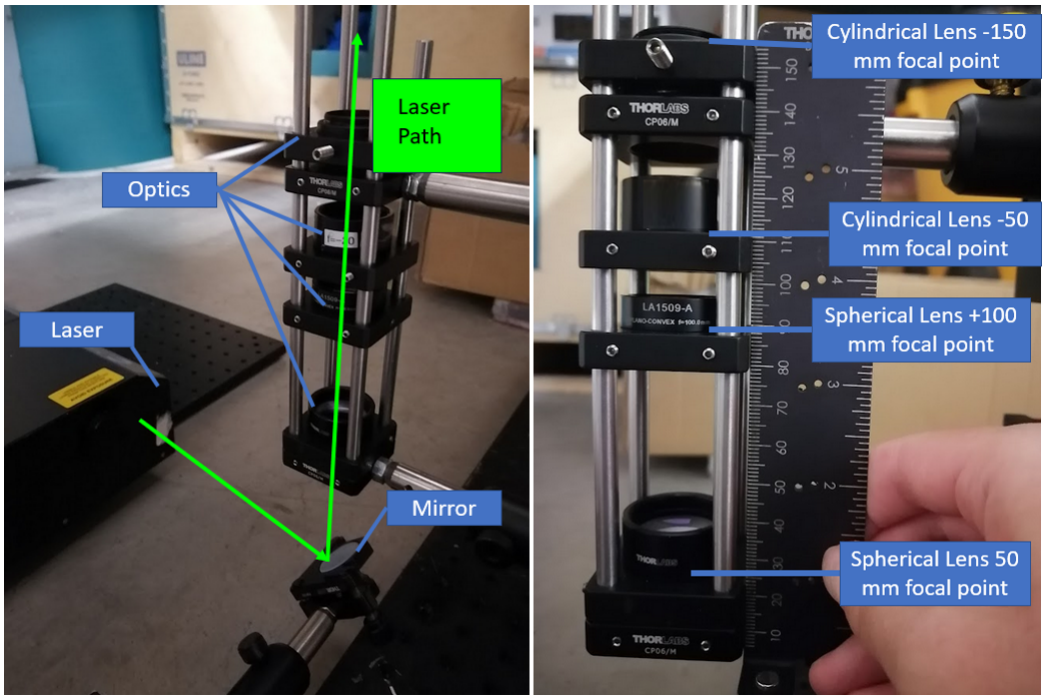


Figure 3.5: Laser setup underneath the wind tunnel.

The laser sheet, shown in Figure 3.6 passes through a clear acrylic panel in the floor of the wind tunnel to illuminate the regions of interest. The actuator cast a shadow through the laser sheet, which can be seen in Figure 3.6. The position of the actuator split the laser sheet into an upstream and downstream region. Although the ideal setup would have not had the actuator shadow, this method was required due to time constrains and lack of additional lasers.

The intensity of the laser beam is determined by two elements: the flashlamp and the attenuator. For this experiment, the flashlamp was set to be fully open and the attenuator was set to a value of 900. The laser intensity is also affected by the laser optics, which determine the size and thickness of the laser sheet. The sheet was made as narrow as possible yet still illuminating the two regions of data collection.

Figure 3.6: The laser sheet in the wind tunnel. The streamwise and wall-normal particle velocity vector fields were calculated for the upstream region from the floor to 96 mm above the floor.

3.2 PIV Particles

The wind tunnel was filled with particles for PIV from the Fog Fury 3000 fog machine, shown in Figure 3.7. This fog machine uses water based fog fluid with glycol-water droplets with particle diameter on the order of 1 μm. These particles will follow the air movement in the wind tunnel and show up on the camera when illuminated with the laser [16] .

Figure 3.7: The fog machine, shown in the wind tunnel, used to seed the wind tunnel flow with particles for PIV.

3.3 Microcontroller

The Teensy LC microcontroller, shown in Figure 3.8, was selected as it is a cost effective, accessible, reasonably fast hardware that can easily interface between MATLAB and other hardware, and is capable of both digital and analog outputs. The actuator used in this experiment required an analog input. The Teensy can accept serial communication from MATLAB, and thus was a natural interface between MATLAB and the actuator. The Teensy LC voltage output is 12 bits 0-3.3 V Digital to Analog (D2A) [51]. Other D2A cards were researched, but no other were found that were able to interface with both Linux and MATLAB, as well as provide an analog output.

Both the camera and the lasers were triggered by a TTL signal generated by the

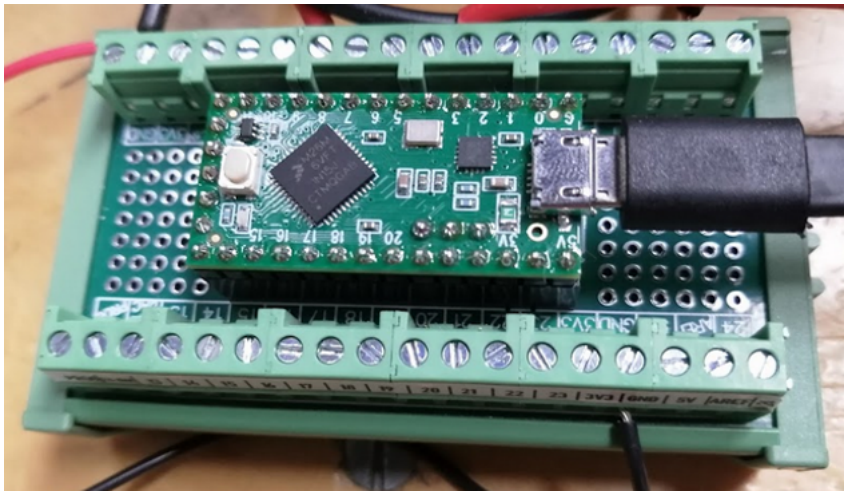


Figure 3.8: The Teeeny LC microcontroller used for generating the camera shutter and laser trigger pulses.

Teensy LC microcontroller. The Teensy output three TTL signals on three of its digital pins. The timing of these signals is shown in Figure 3.9. One signal triggered the Basler camera shutter (shown in blue in the image) through its I/O line and the other two triggered the two laser cavities (laser 1 shown in green and laser 2 shown in red). One laser pulse occurred during each frame the camera took. The timing between the camera trigger, first laser pulse and second laser pulse was controlled using the same Teensy board. The Teensy was programmed using the Teensydruino package and connected to MATLAB via serial communication. All triggers occurred on the rising edge of the TTL signal. The location of the falling edge had no effect on any of the hardware. The timing of each event is also tabulated in Appendix A for further clarity.

An oscilloscope was used to measure the delay between the camera trigger input signal and shutter output signal, which was measured as 20 μs and the minimum inter-frame timing (the time the camera needs between the shutter closing of one frame and the shutter opening for the next frame) of the camera was measured to be 95 μs at these camera settings. These measurements were taken when the camera was running at 50 fps using a hardware trigger (the Teensy microcontroller). Using

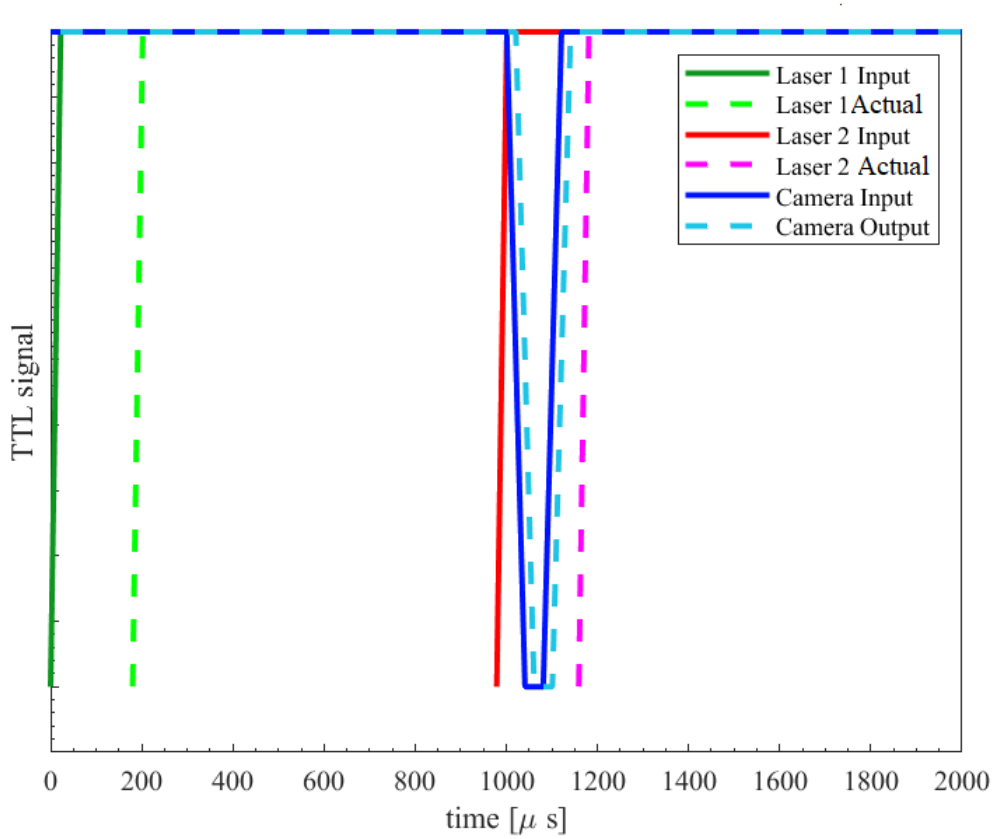


Figure 3.9: The time of the triggers and responses of the camera (shown in blue), the first laser pulse (shown in green), and the second laser pulse (shown in red). The camera input (solid blue) and output (dashed blue) were both measured using an oscilloscope. The lasers fire 180 μs after the trigger is sent, as shown on the dashed lines. Shown to scale.

a different frame speed results in different timing delays.

The camera (shown in blue at the top of the Figure 3.9) trigger rising edge occurs at 0s, meaning the shutter opens at approximately 20 μs. Laser 1 (shown in green in the middle of Figure 3.9) is triggered at a time of 819 μs. The trigger to the laser triggers the laser's flashlamp on. The laser fires 180 μs after the trigger is sent, corresponding to a time of approximately 999 μs. This laser pulse will be captured in the first image of the image pair since the camera shutter remains open until 1020 μs (opened at 20 μs with an exposure time of 1000 μs). The second laser (shown in red at the bottom of Figure 3.9) is triggered 160 μs after the first laser. This is the Δt

for the PIV. The camera is triggered for the second image at 1130 μs and the shutter opens before 1150 μs. The second laser fires at 1159 μs (180 μs after the laser pulse is triggered at 979 μs). The camera shutter remains open until 2150 μs, after which it will close and reopen to capture the next pulse from Laser 1.

A second Teensy LC board was used to send signals to the actuator. This Teensy board was also programmed using the Teensydruino package. The board was programmed so that its analog output pin would produce the voltage required to hold the actuator to its midpoint. It would wait for a serial signal from MATLAB based on MATLAB's PIV calculations, and then convert the signal from MATLAB to a resulting voltage output on its analog pin. This new voltage would move the actuator to the desired position.

The time taken to send the signal to the actuator can be approximated by sending a continuous HIGH-LOW oscillating signal from MATLAB and measuring the period of the oscillations using an oscilloscope. It was determined that the delay between the signal from MATLAB and the Teensy's output is approximately 250 μs.

A third Teensy board was also connected to read the voltage output by the actuator. On an analog input pin, it measured the voltages for each test and reported them as an 8-bit number in order to compare the actuator input to the actuator output.

The Teensy was used for timing since a single microcontroller could be dedicated to timing with zero interrupts. A separate Teensy board was selected for each task such that tasks would not interfere with each other and to ensure proper timing. The Teensy board has good timing precision but like every electronic device is not perfect. A jitter analysis to determine the timing drift is presented in Chapter 4.

3.4 Arduino IDE

The Arduino IDE with the Teensyduino add-on package was used to program the Teensy boards. The versions used were Arduino IDE (1.8.19) with Teensyduino (Version 1.57). The Arduino Uno does not provide a true analog output and thus could

not be used to drive the actuator.

3.5 Actuator

The actuator was used as a controllable deformable surface on the wall of the fluid flow. The actuator used in this experiment is the voice coil actuator LAS16-23-000A-P01-DASH, shown in Figure 3.10. This actuator was designed and built by Bradley Gibeau [16]. It is controlled by the software MotionLab2, which sets the parameters for its position controller. The actuator has been tuned [16] using MotionLab2. MotionLab2 accepts a range and an offset to map the position range of the actuator to a corresponding input voltage. MotionLab2 thus acts as a servo control for the deformable surface. The input voltage linearly maps the actuator to move to a corresponding position, which has a range of 6 mm, centered at 3.04 mm.

For this experiment, MATLAB will send a signal to the second Teensy, which will output a voltage proportional to the selected velocity vector determined by the PIV code in MATLAB. The actuator parameters were chosen such that a -0.2 V input corresponded to the minimum displacement (0.04 mm), and a 3.5 V input corresponded to the maximum displacement (6.04 mm). This range is slightly larger than the Teensy's analog output, which has a range of 0.0 V to 3.3 V. The larger range was selected as a factor of safety for the actuator. The corresponding MotionLab2 software inputs used were 32.43 mm range and -15.85 mm offset.

A second PIV system is located immediately downstream from the actuator. This system is a second Basler Ace 2 camera, also with a 75 mm Computar lens. The camera is connected to a separate computer and images are collected and stored for offline PIV. This camera is triggered by the same Teensy signal as the RT-PIV camera, so it is synchronized with the same laser pulses as the RT-PIV system.

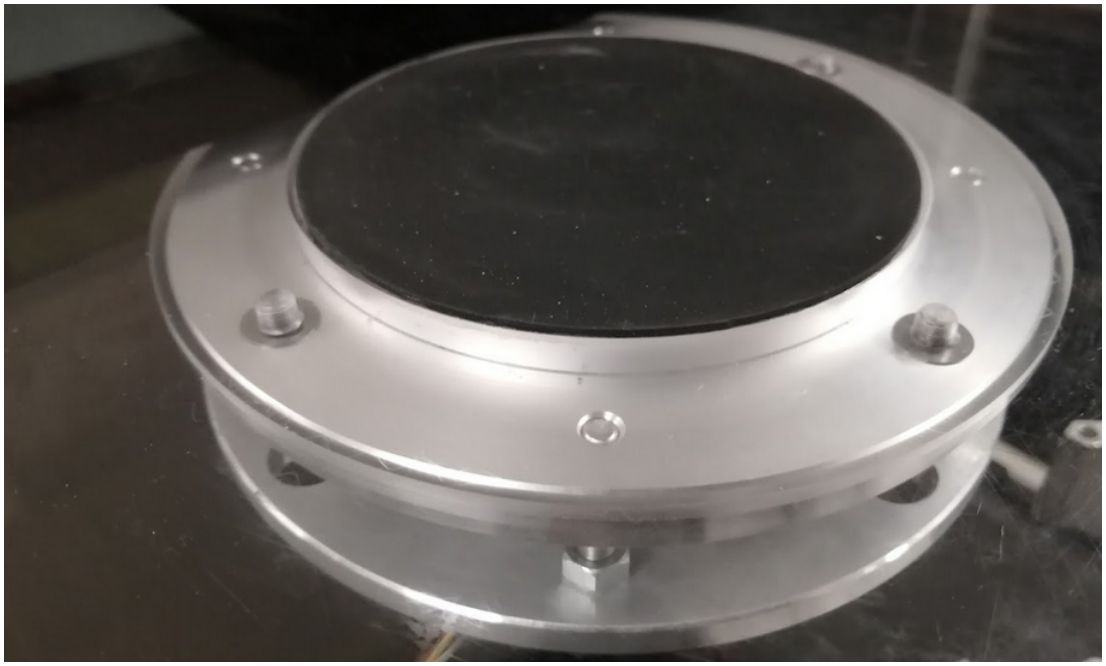


Figure 3.10: The actuator, shown in its neutral position (3.04 mm), mounted flush to the acrylic floor of the wind tunnel.

3.6 Computer and GPU Specifications

This RT-PIV system was built and tested on a Linux OS: Ubuntu 20.04.5. Linux was chosen as a faster OS than Windows. Linux is also an open source OS. The GPU used for processing was the NVDIA GeForce GTX 1080 Ti. The Nvidia Driver and CUDA toolbox was installed to enable MATLAB to run calculations on the GPU. For this experiment, the Driver Version 515.86.01 and the CUDA version 11.7 were used. GPU was used instead of FPGA (Field Programmable Gate Arrays- a type of programmable logic device [52]) since GPUs are ideal for performing parallel calculations [53], such as computing PIV calculations simultaneously on every interrogation window of an image. Software installation instructions are shown in Appendix B. The computer had an Intel i7-8700k processor.

Figure 3.11: View Inside the University of Alberta Wind Tunnel.

3.7 Wind Tunnel

The Wind Tunnel is a two-story facility located in the Mechanical Engineering Building at the University of Alberta, and is shown in Figure 3.11. The wind tunnel can generate sufficient flow speed to obtain a turbulent flow of air through itself. Optical access required for PIV is permitted through acrylic panels in the test section. The wind tunnel was run with a free stream flow speed of 4 m/s. The inside of the test section of the wind tunnel is 2.45 m $\times$ 1.18 m $\times$ 11 m. In this experiment, L, the streamwise distance from the trip location to the upstream PIV data collection region, was measured to be 7.7 m.

The turbulent boundary layer on the floor of the wind tunnel is generated by using a trip wire. This is a series of random protrusions across the spanwise length of the floor of the wind tunnel. The trip wire used in this experiment is shown in Figure 3.12. It was designed and built by Bradley Gibeau [16]. The trip wire is formed by gluing a series of rocks to sandpaper to create a rough and irregular surface. The sandpaper is 7.5 cm in the streamwise direction and covers the entire span of the wind tunnel (2.45 m).

Figure 3.12: The trip wire used in this experiment to commence the start of the turbulent boundary layer on the floor of the wind tunnel.

The Reynolds number at the upstream PIV location in the wind tunnel operating at 20 °C, can be calculated using Equation 2.1 as follows in Equation 3.1 [1]. Since Re is greater than 10^6, the flow at the experiment location can be considered to be fully turbulent.

$$Re = \rho u L/\mu = 2.0 \times 10^6 > 10^5 \tag{3.1}$$

$$\begin{aligned} \rho_{air} &\quad 1.204 \text{ kg/m}^3 \\ u &\quad 4 \text{ m/s} \\ L &\quad 7.7 \text{ m} \\ \mu_{air} &\quad 1.825 \times 10^{-5} \text{ kg/ms} \end{aligned}$$

The thickness of the boundary layer at the region of interest in the wind tunnel [1] as shown in Equation 2.2 (7.7 m downstream from the trip wire) can now be calculated by Equation 3.2

$$\delta = x(0.16)/(Re)^{1/7} = 0.155m \tag{3.2}$$

$$\begin{aligned} x &\quad 7.7 \text{ m} \\ Re &\quad 2.0 \times 10^6 \end{aligned}$$

Therefore, the investigations in the wind tunnel must take place within 155 mm from the floor in order to be within the boundary layer.

3.8 Flow Measurement

The speed of the wind tunnel flow is controlled by the fan. The resulting air flow speed is measured using a flowmeter to ensure accuracy. The speed of the wind tunnel was set to 4 m/s.

3.9 MATLAB Algorithm Explanation

The MATLAB PIV algorithm uses the FFT-CC method, as explained in Chapter 2, and a Gaussian subpixel fitting method to determine the flow velocity vectors. Each image is divided into 64-pixel by 64-pixel subsections called Interrogation Windows (IW). An example of this is shown in Figure 3.13. A square IW with side lengths as a power of 2 was chosen to minimize the calculation time for the FFT. The selected size of the interrogation window used for RT-PIV calculations in this experiment was 64 pixels by 64 pixels.

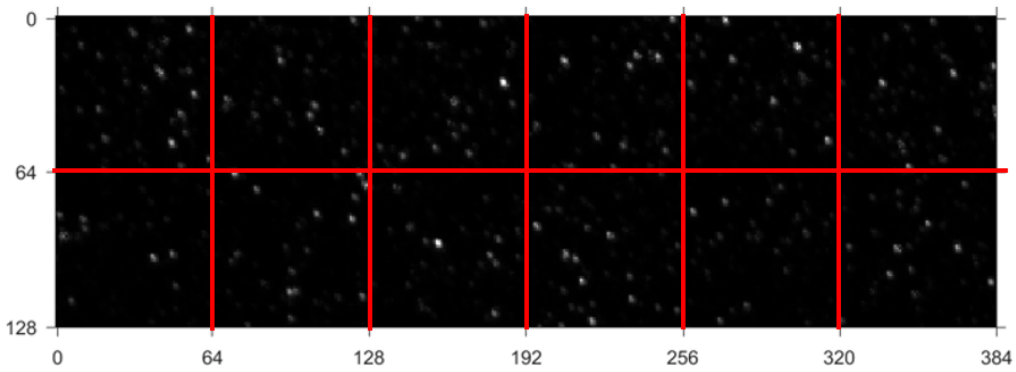


Figure 3.13: Example of dividing a 384 pixel by 128 pixel image into 64 pixel by 64 pixel interrogation windows.

The interrogation windows are stacked, as demonstrated in Figure 3.14, to create a 3-D array in order for all calculations to be completed on each window pair simultaneously. This increases the overall calculation speed when using the GPU since GPUs are ideal for completing many simultaneous calculations.

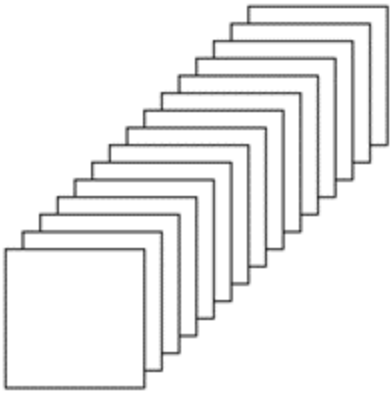


Figure 3.14: The IWs are reshaped into a 3D-array.

On the second image in each pair, IW locations are shifted by a number of pixels in the streamwise direction to account for the particle motion. The freestream flow of this experiment was 4 m/s. However, within the boundary layer, particle flow is slower. For flow at 3 m/s and a Δt of 160 μs, expected particle displacement was 10 pixels per frame. This particle speed will be verified in Chapter 4 to ensure a 10 pixel shift in the streamwise direction is appropriate for these experiment settings. After the IWs are shifted, the cross-correlation algorithm proceeds as normal. The 10-pixel shift is accounted for after cross correlation and Gaussian sub-pixel estimation so that particle velocity estimation remains accurate. This IW pixel shift is used so that a greater number of particles remain within the same IWs in the image pair.

All IWs from a single image are "stacked" to create a 3D array that is sent to the GPU for calculation. Once calculations are complete, the velocity vectors are retrieved from the GPU, unstacked and rearranged into the corresponding location within the image to create a vector field.

The collected PIV images are 1920 pixels $\times$ 200 pixels, resulting in a 30 $\times$ 3 vector field (3 vectors in the streamwise direction and 30 vectors in the wall normal direction). In the experiment, two different vectors were compared for flow control. The first vector selected for input into the control law was the furthest upstream vector that is the third up from the wind tunnel floor. This vector comes from the

IW spanning from about 6.5 mm to about 9.5 mm, or centered around 8 mm from the floor of the wind tunnel. For the next test iteration, the furthest upstream vector that is the fourth up from the wind tunnel floor was selected to be input into the control law. This vector comes from the IW spanning from about 9.5 mm to about 12.5 mm, or centered around 11 mm from the floor of the wind tunnel.

The selected velocity vector is tracked over the series of image pairs. An outlier rejection method is used, rejecting any measurements more than 5 pixels difference between the current calculated displacement and the mean particle displacement. Rejected velocities are replaced with the velocity vector from the previous iteration, so that the actuator will hold the previous position.

Even after outlier rejection, the measured velocity vector field in a turbulent flow using PIV can be very noisy. It is desirable to remove the random noise from this signal and act only on the actual velocities [54]. The vectors of interest were thus filtered to reduce the noise content of the signal. A lowpass filter was selected to remove the high frequency noise from the system. A first order lowpass Butterworth filter was selected due to its flat frequency response in the passband and minimal signal delay. A first order Butterworth filter has a rolloff of -6 dB per octave [54]. A Butterworth filter is a recursive filter [54] meaning it takes a weighted average of the current unfiltered value as well as previous filtered values. The Butterworth discrete filter in MATLAB is desirable in this application for its low latency and low number of coefficients [55], which are important factors for high speed implementation. The filter coefficients were determined in MATLAB using the built-in function `butter`. The implementation of the filter on the velocity is shown in Listing 3.1.

Listing 3.1: Filtering the input velocity

```
%Frequencies
fc = 5; % filter cutoff frequency
fs = 25; %sampling frequency

%Filter Coefficients
[b,a] = butter(1,fc/(fs/2)) ;

% U_in = [current_unfilter_u, previous_unfilter_u, previous_filter_U]

%Filter
filt_U = b(1)*U_in(1) + b(2)*U_in(2) - a(2)*U_in(3);
```

For this experiment, a sampling frequency f_s of 25 Hz was used. Two different lowpass filter cutoff frequencies were compared, f_c of 5 Hz and f_c of 10 Hz. The filter cutoff frequencies were chosen as sufficiently less than half the sampling frequency to avoid introducing increased noise as the filter cutoff frequency approaches half the sampling frequency. Additionally, they were also selected as sufficiently high enough to not significantly decrease the resulting signal bandwidth. A sample of filter effect on velocity at 5 Hz is shown in Figure 3.15.

A proportional control law is applied to the deviational velocity, as previously shown in Equation 2.14. The gain, K, can be defined as the ratio of voltage to pixel shift as shown in Equation 3.3. Let V_{max} and V_{min} represent the maximum and minimum voltage inputs to the actuator (as set in MotionLab2). Likewise, let p_{max} and p_{min} represent the maximum and minimum allowed bounds for estimated particle shift from the RT-PIV algorithm, calculated as $\pm$ 5 pixels from the current mean particle displacement. The calculations were done using estimated particle shift in pixels per image pair, rather than converting to velocity in m/s, to save calculations within the RT-PIV and proportional control algorithm.

$$K = [V_{max} - V_{min}]/[p_{max} - p_{min}] \tag{3.3}$$

For performing active flow control in this system, the reference is the mean streamwise velocity upstream of the actuator. The goal is for the current upstream measured streamwise velocity vector of interest (here the vectors examined are either centered at 8 mm or centered at 11 mm from the wind tunnel floor) to be the same as the

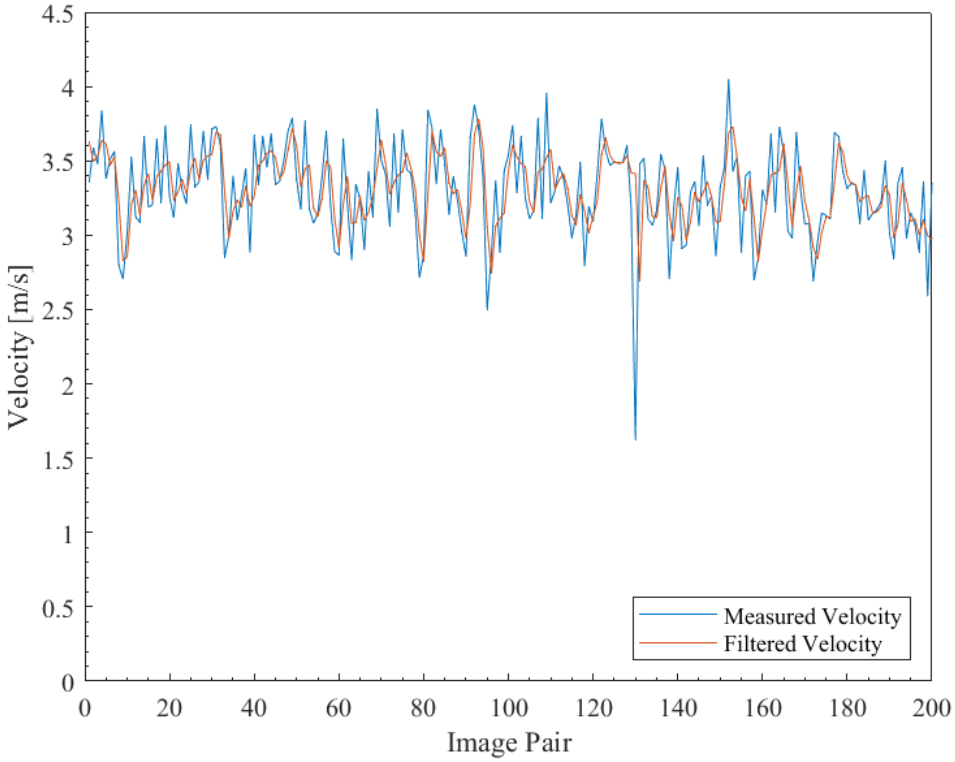


Figure 3.15: The effect of the Butterworth filter with 5 Hz cutoff frequency on the velocity for 200 samples.

mean streamwise velocity. This is so the fluctuating portion of the streamwise velocity downstream of the actuator will decrease to zero. If the difference is non-zero, the proportional controller will signal the actuator to move to a corresponding position to decrease the difference. This is illustrated in Figure 3.16.

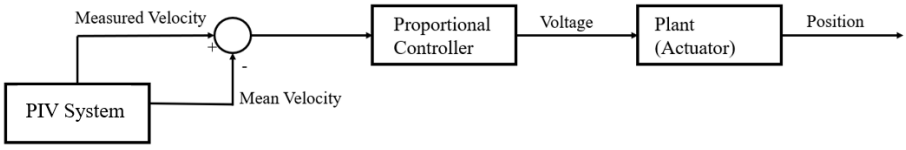


Figure 3.16: The proportional control diagram for this experiment.

The instantaneous error, e, can be defined in Equation 3.4 knowing the current filtered estimated particle shift from the RT-PIV algorithm, p_{in}.

$$e = p_{in} - p_{min} \tag{3.4}$$

The full proportional control law from Equation 2.14 in this experiment is shown in Equation 3.5. V_{out} is the controller output voltage.

$$V_{out} = [V_{max} - V_{min}]/[p_{max} - p_{min}] \times [p_{in} - p_{min}] + V_{min} \tag{3.5}$$

Image collection is conducted in the area immediately after the actuator (using an offline PIV system) to evaluate the effect of the actuation control. Both upstream and downstream images are collected at the same time since the same microcontroller signal triggers both cameras and both cameras use the same laser pulse timing.

Chapter 4

Experimental Results

4.1 Processing Time

The time taken for the RT-PIV and active flow control calculations must be less or equal to the time taken by the particles captured by the RT-PIV camera to flow past the actuator. The steps in the active flow control process can be defined as follows:

1. Collection of Image 1
2. Image 1 is transferred to the RT-PIV computer
3. Collection of Image 2
4. Transfer of Image 2 to the RT-PIV computer
5. Running the MATLAB RT-PIV code
6. Running the MATLAB linear control function
7. Sending signal to the actuator
8. Actuator moving

The time taken to collect each image is determined by the exposure time of the camera. For this experiment, the exposure time was set to 1015 μs. The Δt, the time between the two paired laser pulses, was set to 160 μs. The sensor readout time was determined using Basler's software Pylon [56]. With the parameters of an 8-bit

image, a 1015 μs exposure time, and a ROI of 1920 pixels by 200 pixels, the sensor readout time was 5178 μs.

The theoretical minimum amount of time for the images to transfer from the camera to the MATLAB workspace can be calculated. The images transfer up to USB 3.0 speed, 5 GB/s [56], and the 8-bit images had a size of 1920 pixels by 200 pixels. This yields a minimum transfer time of 77 μs before delays, as shown in Equation 4.1. The MATLAB function `getdata` is used to move images into the workspace. The benefit of using the `getdata` function is that no frames are dropped due to its use of image buffer [57]. This ensures all image pairs remain paired throughout the data collection (i.e. dropping a single frame changes which images are paired).

$$1920 \times 200 \times 8 \text{ bits } /(5\text{GB/s})/(8 \times 10^9 \text{ bits/GB}) = 77 \ \mu\text{s} \tag{4.1}$$

The time that the MATLAB RT-PIV code takes to run can be determined by utilizing the built-in MATLAB functions `tic` and `toc`. Using these timing functions on MATLAB, the PIV code was run 1000 times. The average time for the RT-PIV code was found to be 1.20 ms, with a standard deviation of 0.07 ms. This means the 95% confidence interval for the timing of the RT-PIV system is 1.06 ms – 1.34 ms. This can be compared to the value for the previous code iteration created by a previous student [42]. When run on the same computer used in this current experiment, the time of the previous code was 23.4 ms, with a standard deviation of 3.5 ms. The 95% confidence interval for that was 16.4 ms – 30.4 ms. The previous RT-PIV system [42] was ran at 7.35 Hz. That iteration used Simulink for code execution and communication with the Speedgoat. The RT-PIV code can be further sped up in the future by using a faster GPU.

The approximate time for the delay between MATLAB sending a serial signal to the Teensy and the Teensy's output is 250 μs, as shown in Section

4.2 Microcontroller

.

Total time delay can be determined experimentally by using an oscilloscope. Since the Basler camera is capable of outputting a TTL voltage signal indicating when the shutter is opened and closed, the oscilloscope plot can indicate the time delay between the camera shutter close and the actuator moving or the time delay between the Teensy signal and the actuator reaching its commanded displacement. For this time test, the MATLAB RT-PIV code was run as normally run except the control signal to the actuator was replaced with an alternating high-to-low signal. A digitized rendition of the oscilloscope output is shown in Figure 4.1. The original oscilloscope output is included in Appendix C.

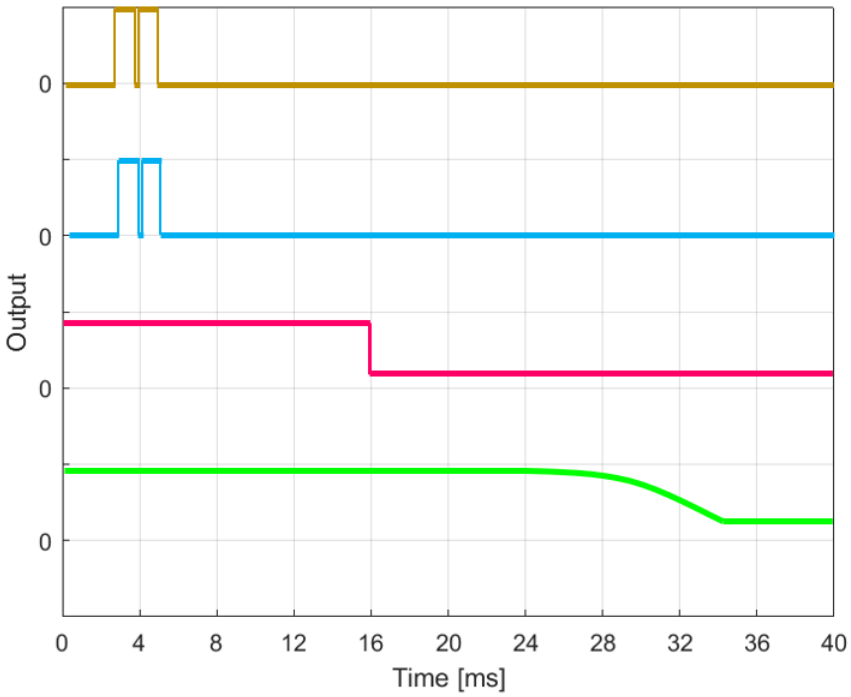


Figure 4.1: Oscilloscope sample output showing the camera input in yellow, camera output in blue, Teensy output/actuator input in pink, and actuator output in green. It can be seen the actuator starts to move at approximately 20 ms after the first of the two images in the pair is collected, and reaches its endpoint approximately 30 ms after the first of the two images is captured.

It can be seen from Figure 4.1 that the most time intensive factor for this setup is the motion of the actuator. As seen from Figure 4.1, the actuator does not start moving until 20 ms after the camera shutter closes, or about 8 ms after it has received the voltage signal from the Teensy microcontroller. The actuator has been tuned [16] to minimize overshoot, however this results in a larger rise time. Excessive oscillations are undesirable in this application since it could increase the velocity fluctuations in the boundary layer.

Although the actuator's motion timescale is long compared to the other components in the RT-PIV system, it can be shown that the actuator reaches its desired position before the particles of interest travel downstream past the actuator.

For this experiment, the free-stream wind tunnel speed was set to 4 m/s. However, within the TBL near the floor, the flow speed will have a gradient. This gradient can be visualized by plotting the streamwise speed of each IW (Interrogation Window) to form a plot of the boundary layer. To visualize the flow boundary layer, 1000 image pairs were collected from the upstream ROI, and the PIV vectors were calculated using the RT-PIV code. Since each image was divided into 3×30 interrogation windows (each of size 64 pixels by 64 pixels) by the RT-PIV algorithm, each image pair yields a 3×30 vector field. Each interrogation window yields one streamwise and one wall normal magnitude of displacement. This vector field has three vector components in the streamwise direction and thirty vector components in the wall normal direction.

The matrix of streamwise vector magnitude was averaged over the 1000 vectors fields as well as averaged over the three vectors in the streamwise direction. This yielded a thirty-element vector that shows the average streamwise velocity of the particles at each vector location in the wall normal direction. Since each IW spans 64 pixels, each IW has a side length of 3.2 mm. The boundary layer velocity profile is shown in Figure 4.2. Note that the laser sheet was much fainter further from the wind tunnel floor due to the laser light diverging (this is shown in Figure 4.8). The

vectors farthest from the wind tunnel floor were removed due to poor quality (this can be seen in Figure 4.9). The plot of the boundary layer shown in Figure 4.2 looks as expected; particle velocity is low near the wall and increases as distance from the wall increases.

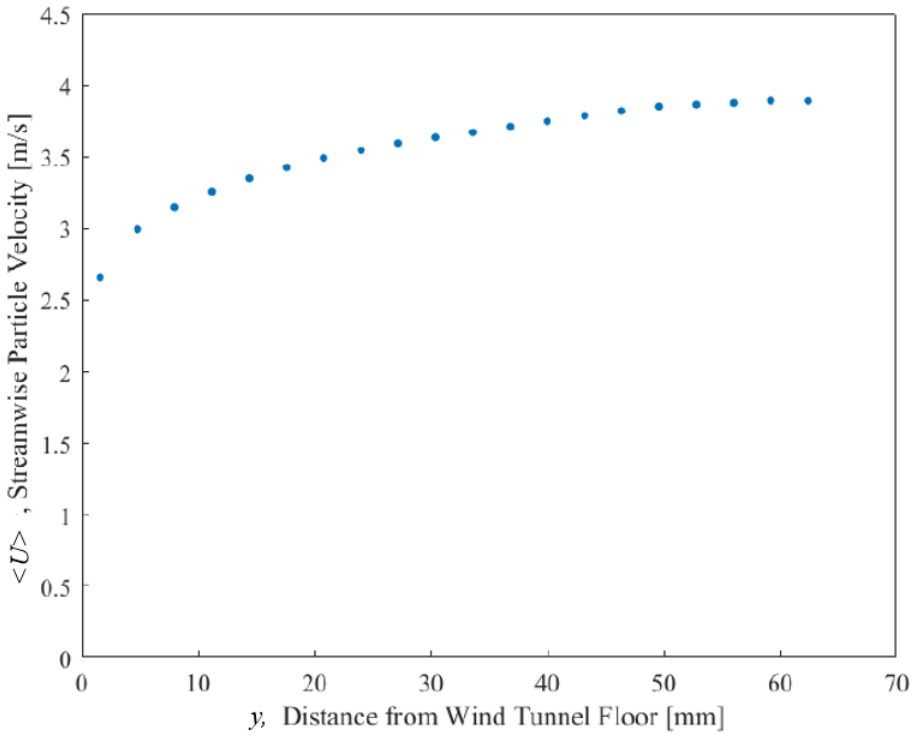


Figure 4.2: Figure showing the boundary layer, the average streamwise flow velocity at each location.

The two vectors of interest for RT-PIV are the 3rd and 4th vectors from the wall. The approximate location of the center of each vector's IW can be determined by placing a ruler at the location of the laser sheet and overlaying the location of the interrogation windows. Vector 3 is calculated from the IW spanning from about 9.5 mm to about 12.5 mm from the wind tunnel floor, or centered around 11 mm from the wind tunnel floor. Vector 4 is calculated from the IW spanning from about 6.5 mm to about 9.5 mm from the wind tunnel floor, or centered around 8 mm from the wind tunnel floor. These vectors were chosen for their proximity to the wall, since the Reynolds Stress $< uu >$ is highest near the wall. These vector locations also

gave more consistent results than the two vectors closer to the floor. The thickness of the boundary layer was calculated in Chapter 3 to be 0.155 m thick at the region of interest, and thus these vectors lay within the boundary layer. These correspond to measured streamwise velocities of approximately 3.26 m/s and 3.15 m/s respectively. The distance the particles have travelled in this time before the actuator moves can be calculated as shown in Equations 4.2 and 4.3 (using the larger velocity to determine the longest distance the particles have travelled). The previous time values that the actuator is moving between 20-30 ms after the first image is captured (actuator starts moving around 20 ms and reaches its final position around 30 ms) are used for these calculations.

$$(3.26 \text{ m/s})(20 \text{ ms}) = 65.1 \text{ mm} \tag{4.2}$$

$$(3.26 \text{ m/s})(30 \text{ ms}) = 97.8 \text{ mm} \tag{4.3}$$

The deformable surface of the actuator starts 40 mm downstream from the RT-PIV location of interest and the end of the deformable surface of the actuator is 140 mm downstream from the RT-PIV location of interest. Control is applied to the midpoint of the actuator (90 mm downstream of the ROI). This means that the actuator will have moved and reached its final position before the particles have travelled past the deformable surface of the actuator.

To ensure that no frames are dropped and that the RT-PIV system is working on the most recent image pair, the time stamps for the image acquisition and the time stamp for when the control signal is sent to the actuator were collected in MATLAB. The difference in time between when the signal was sent to the actuator and when the second image was taken is plotted in Figure 4.3. It can be seen from Figure 4.3 that the delay between the two events — the time the image is taken and the time that MATLAB sends a signal to the actuator — remains nearly constant, about 5130 μs.

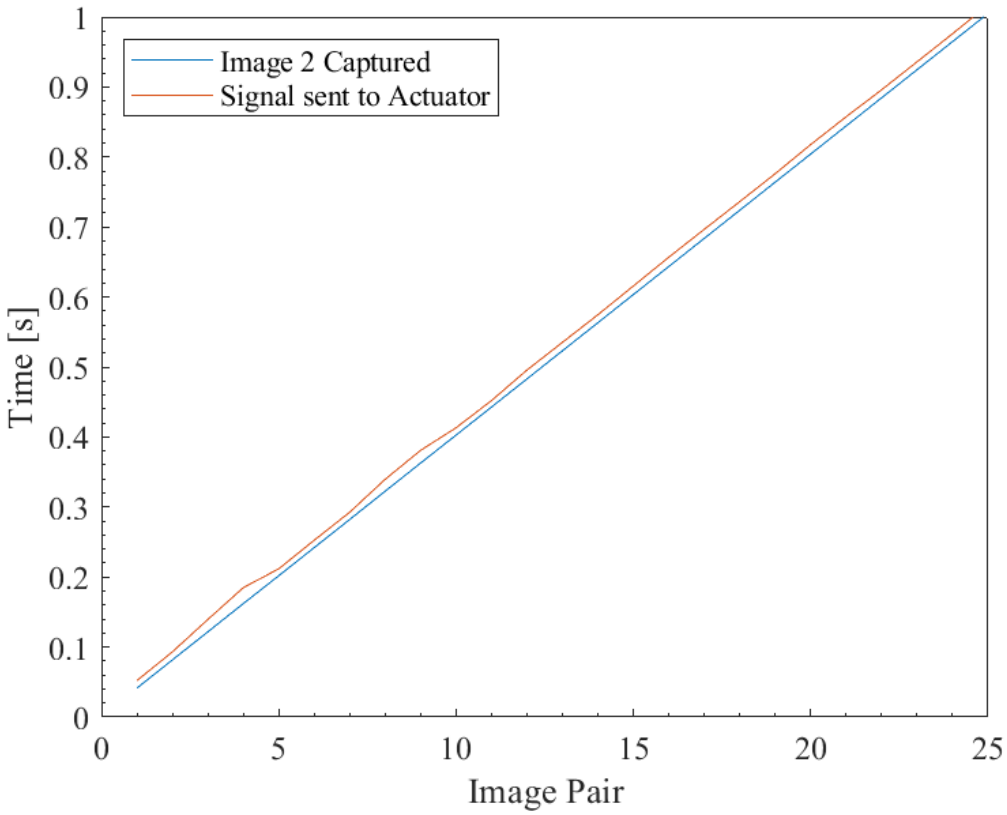


Figure 4.3: The timestamp (from the second image in the RT-PIV process) as well as the timestamp of when the signal is sent to the actuator. One second of time stamps are provided for visual graph clarity, however the time between the two lines remains the same throughout the test.

The required times for each event in the RT-PIV process are listed in Table 4.1, as well as how the time was determined.

Table 4.1: Times for Events in the RT-PIV process

Event	Time	Measured by:
Camera exposure start delay	20 μs	Oscilloscope
Image 2 exposure time	1015 μs	Camera Settings
Camera Sensor readout time	5178 μs	Pylon
Camera USB transmission	77 μs	Nominal
MATLAB RT-PIV code	1200 μs	`tic toc`
Remaining MATLAB code	3930 μs	`tic toc`
Signal sent to actuator	250 μs	Oscilloscope
Actuator transport delay	18 ms	Oscilloscope
Total	**30 ms**	Oscilloscope

4.3 Comparison of PIV Results with Commercial Software

In order to validate the results of the RT-PIV calculations, the calculated vector fields were compared to those calculated offline post-experiment by the commercial PIV software package DaVis. Note that DaVis is capable of more accurate results using more advanced settings (for example, using Image Pre-processing, Multi-Pass, Post Processing, etc), however, for the first comparison the settings in DaVis were chosen to best match the RT-PIV algorithm in MATLAB in order to ensure a fair comparison.

List of settings used for PIV in DaVis:

- Vector calculation – double frames
- No Preprocessing
- Cross-Correlation
- Single Pass

- Square Window Size 64 pixels × 64 pixels with zero overlap
- Offset: 10 pixels in the streamwise direction
- No vector postprocessing

The streamwise velocity results for the two reference locations (8 mm and 11 mm from the wind tunnel floor) were calculated using both the RT-PIV code in MATLAB and DaVis. The results from both reference locations are compared across 200 samples as plotted in Figure 4.4 and Figure 4.5.

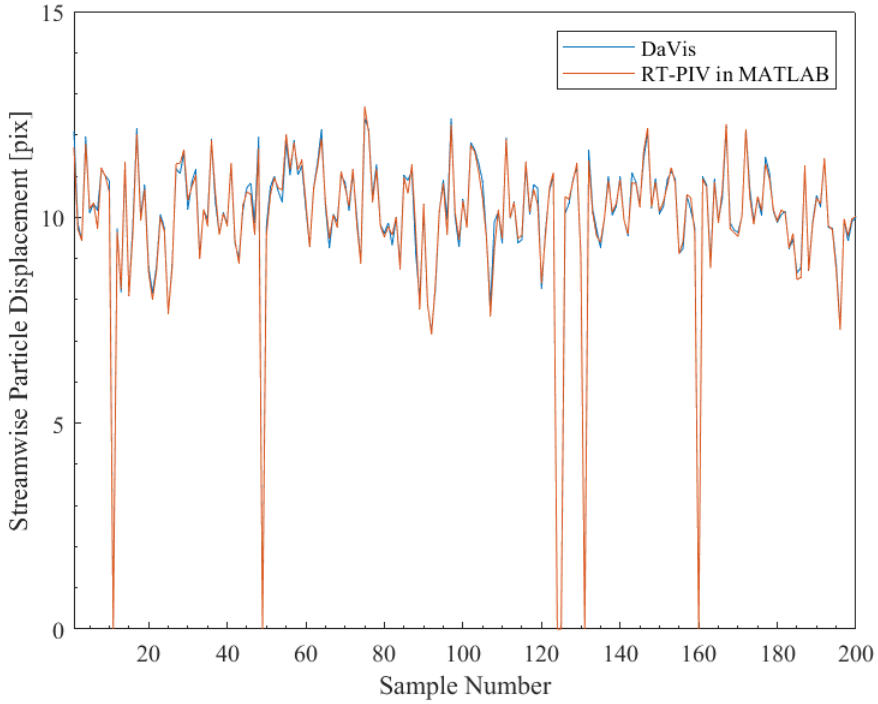


Figure 4.4: Comparison of calculated particle displacement at vector location 3 (IW centered at 11 mm from wind tunnel floor) for 200 image pairs using both DaVis and the RT-PIV code in MATLAB.

The two data sets show close similarity between the MATLAB RT-PIV code implementation and DaVis, apart from the few outliers in either method. The similarity can be better assessed by removing the outliers (those values less than 5 pixels or greater than 15 pixels) and plotting the DaVis and RT-PIV results against each

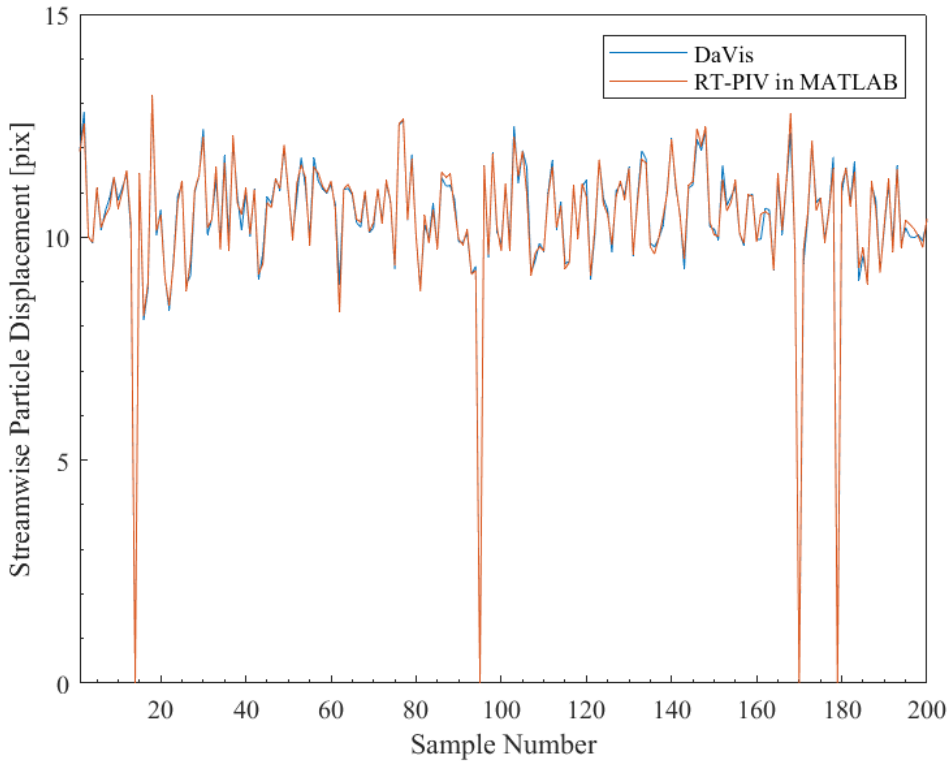


Figure 4.5: Comparison of calculated particle displacement at vector location 4 (IW centred at 8 mm from wind tunnel floor)for 200 image pairs using both DaVis and the RT-PIV code in MATLAB.

other. Removing the outliers is acceptable since the RT-PIV algorithm specifically ignores outliers (detected by the control code as any data point showing more than 4 pixels displacement above or below the mean displacement), and thus they will be ignored by the control law.

A paired sample t-test can be used here to check the mean difference between the pixel displacements calculated by the RT-PIV MATLAB algorithm and those calculated by DaVis. The built-in MATLAB function `ttest` was used. This test determines if a linear equation with zero offset will be an appropriate fit for the data. The t-test showed the null hypothesis of the pairwise difference between the two data vectors has a mean equal to zero (at the 1% significance level) was not rejected, with a p-value of 0.474. The p-value is the probability of observing a test statistic given

that the null hypothesis is true. Since this p-value is larger than 0.1, the difference between the two data vectors is not statistically different from zero [58]. This test showed a 99% confidence interval of -0.0290 to 0.0164. Here the confidence interval states the range for the true population mean. The value of the test statistic was -0.7161 with 1001 degrees of freedom and an estimated difference in standard deviation of 0.2787. This standard deviation gives a 95% confidence level that the MATLAB RT-PIV algorithm matches DaVis up to a displacement between -0.6 pixels to 0.6 pixels.

A linear regression calculation was preformed to calculate the relation between the particle's displacement calculated by DaVis and the particle's displacement calculated by the RT-PIV algorithm in MATLAB. Figure 4.6 shows the fitted regression line, and that the relation between the DaVis and RT-PIV particle displacements is linear.

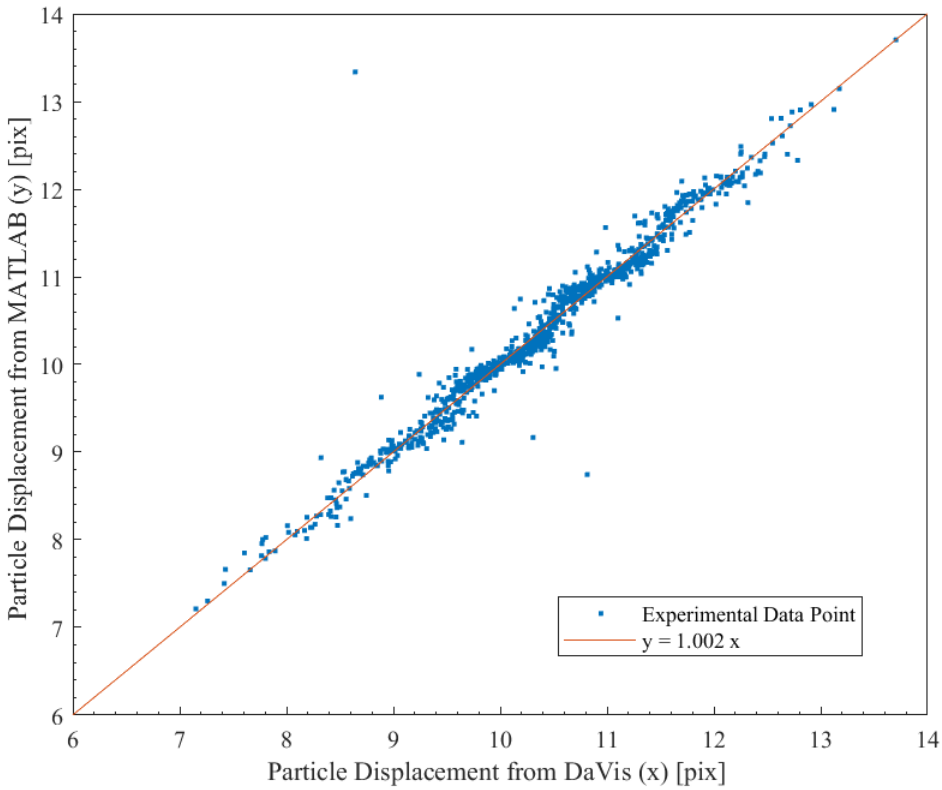


Figure 4.6: Linear regression fit of calculated particle displacement for 200 image pairs at each ROI using both DaVis and the RT-PIV code in MATLAB.

The closeness of the fit can be quantified by the coefficient of determination R^2. The coefficient of determination is the ratio between the variation explained by regression (residual sum of squares) to the total variation (total sum of squares). It is calculated using Equation 4.4. A value of 1 indicates a perfect correlation and as the value decreases from one, the correlation weakens.

Designate X as the values given by DaVis, Y as the values given by the RT-PIV algorithm, and B as the linear coefficient found from the linear regression. Similarly, designating $\overline{Y}$ as the mean of the values given by the RT-PIV algorithm, R^2 can be calculated as follows:

$$R^2 = 1\text{-}[\sum_{n=1}^{n}(Y\text{-}BX)^2]/[\sum_{n=1}^{n}(Y\text{-}\overline{Y})^2] \tag{4.4}$$

The calculated value for R^2 was 0.9804. This indicates that the data is highly correlated, and thus the RT-PIV algorithm provides an acceptable replacement to DaVis for calculating particle displacements between PIV frames. The measured streamwise particle displacement (or even velocity vector) could also be compared to DaVis' results using more advanced settings (for example, using Image Pre-processing, Multi-Pass, Post Processing, etc), however, this would not be a fair comparison since DaVis can employ powerful offline processing techniques whereas the MATLAB code is specifically designed for use in real time.

While the R^2 value gives a value for correlation between the two data sets, the Root Mean Square Error $RMSE$ can quantify the difference between the two data sets. The MATLAB function `rmse` was used to find the $RMSE$ value between the two vector sets. It was found that the $RMSE$ was 0.279.

4.3.1 Streamwise Pixel Offset for PIV calculations

In order to decrease the number of outliers produced by the RT-PIV algorithm, the interrogation windows can be shifted in the direction of the expected particle motion. This is helpful to decrease the likelihood that a particle has moved out of range of

the window. The expected particle displacement, in pixels, is calculated. Equation 4.5 shows the expected particle displacement in the free stream, where the particles are travelling at 4 m/s, and Equation 4.6 shows the expected particle displacement at the region of interest, where particles are travelling at 3.26 m/s.

$$\delta_{pix} = 4\text{m/s} \times 160\mu \text{ s} \times 1920\text{pix}/[96\text{mm}] = 12.8 \text{ pix freestream} \tag{4.5}$$

$$\delta_{pix} = 3.26\text{m/s} \times 160\mu \text{ s} \times 1920\text{pix}/[96\text{mm}] = 10.4 \text{ pix ROI} \tag{4.6}$$

Based on these calculations, the RT-PIV code in MATLAB uses a 10-pixel shift in the streamwise direction between corresponding IWs in each image pair to account for the motion of the particles. This shift in IWs is only to increase the number of particles that remain within the corresponding IWs, i.e. to decrease the possibility of a bad correlation. The shift is accounted for in velocity calculations so that the pixel shift of the IW does not change the final value for velocity.

4.4 Error Analysis

Several sources of error need to be considered in this analysis. The three main sources in this experiment are error in timing, error in particle displacement, and error in actuator motion.

4.4.1 Error in timing

Errors in timing of the laser pulse or camera shutter occur due to small drifts in the timing of the Teensy LC microcontroller. The jitter of the Teensy can be measured using an oscilloscope. The timing of the laser pulse is the most critical, as the camera shutter remains open and nothing will be seen on the image until the laser is pulsed to illuminate the ROI. The approximate variation in the trigger signals of the laser pulses was measured using an oscilloscope as approximately 3 μs. This is small compared to the Δt (time between two paired laser pulses) value of 160 μs (approximately 2%

error).

The effect of the timing drift on the calculated particle velocity can be approximated in Equation 4.7. It is undesirable to have a large drift in the timing, since if the laser pulses occur too close together, the particles will have had less time to travel, and the RT-PIV code will measure less displacement between frames and report that smaller number to the control code. Likewise, if the timing of the laser pulses drift and are further apart in time, the particles will have had more time to travel, and the RT-PIV code will measure more displacement between frames.

$$3\ \mu\text{s} \times 3.26\ \text{m/s} \times 1920\ \text{pix}/(96\ \text{mm}) = 0.2\ \text{pix} \tag{4.7}$$

$$0.2\ \text{pix}/160\ \mu\text{s} = 0.02\ \text{m/s} \tag{4.8}$$

Since the drift in timing is small compared to the scale of the time measurements, the overall effect on the values determined by the control code is also small.

4.4.2 Error in Particle Displacement Calculation

The frames collected from the Basler cameras were 8-bit grayscale images, meaning each pixel in each recorded image had a value between 0-255. An "ideal" PIV image at these settings would have background pixels of 0 and particle pixels with a high saturation. However, it is also important that the image is not oversaturated with particles at an intensity level of 255. If many pixels of the image are fully saturated, it becomes difficult for the algorithms to distinguish particles from each other, which may result in the wrong particles getting correlated. A less ideal PIV image could have background noise above 0 or low pixel intensity. In this experiment, it was noted that the cross correlation coefficient plots were offset from zero, indicating a level of background noise. Decreasing the background noise would improve the correlation matrix.

The maximum intensity of the pixels in the upstream ROI was plotted for 1000 sample images as shown in Figure 4.7.

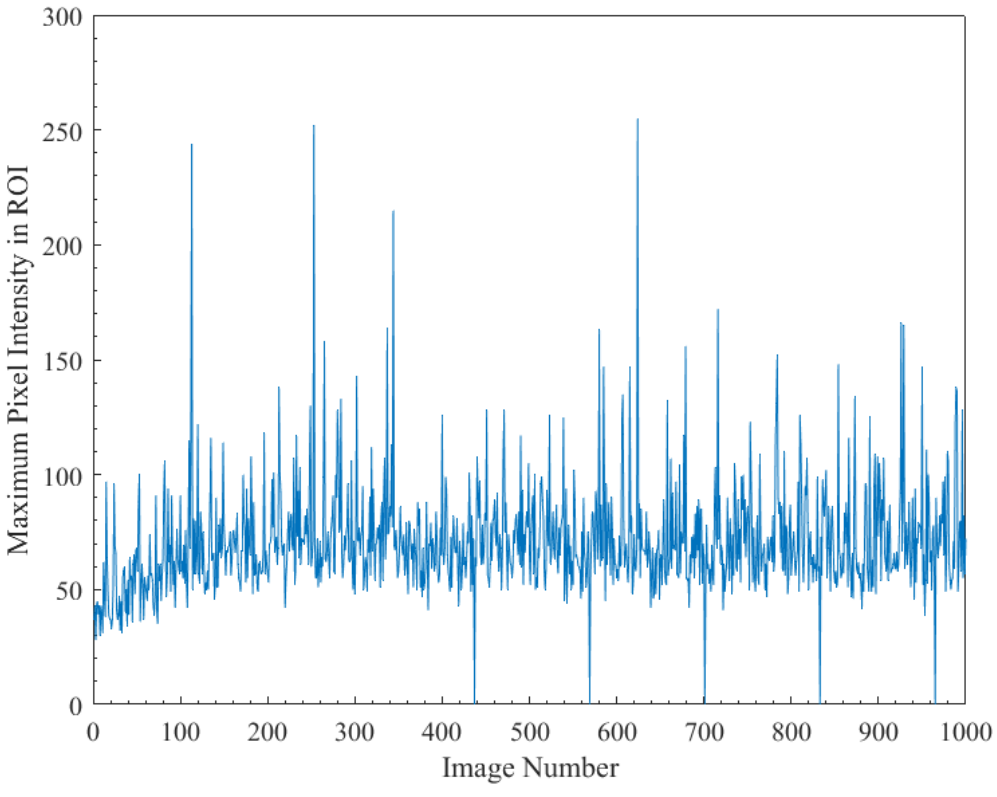


Figure 4.7: The maximum pixel value in the ROI of the upstream data collection for 1000 images

The mean value of maximum pixel intensity in the ROI for these 1000 images is 71. A higher maximum value would enable a better correlation. This could be achieved by increasing the brightness of the laser sheet, however, due to the current experimental setup, the laser was already at its maximum output power and the sheet could not be spread any thinner while still illuminating both the upstream and downstream ROIs. Future experiments could benefit from using a more powerful laser or even separate lasers for upstream and downstream imaging areas.

Another possible source of error in the calculated particle displacement is if the RT-PIV algorithm correlates particles between the paired images that are not the same particle. This was somewhat mitigated by the outlier rejection in the control code,

where if the control code received a velocity value that was beyond the designated acceptable bound of ±4 pixel deviation from the mean velocity, the actuator would hold its previous position rather than moving. As discussed in the previous paragraph, increasing the maximum particle intensity and allowing for more varied brightness within the ROI would also help decrease errors of this sort. A further improvement could be to modify the RT-PIV algorithm to do a multi-pass calculation or do a single pass with IWs offset (for example, dividing the image into a second group of IWs at a different location than the first group of IWs before sending all IWs to the RT-PIV algorithm). However, this solution would come with increased computational cost. This could be offset by employing a more powerful GPU.

Comparing the calculated RT-PIV displacement to the calculated DaVis displacement, the RT-PIV calculated displacement was within 0.6 pix of the DaVis value at a 95% confidence level. Because this displacement variation is small, the RT-PIV algorithm is considered to be trustworthy. The error can be calculated in m/s as well.

$$0.6 \text{ pix} \times \frac{50\mu\text{m}}{\text{pix}} \times \frac{1}{160\ \mu s} = 0.19\text{m/s} \tag{4.9}$$

Combining the errors in timing with the error in particle displacement results in a total error of 0.2 m/s.

4.4.3 Error in Actuator Motion

The actuator was tuned by an earlier student [16]. As seen in Figure 4.1, the actuator rises smoothly to reach its final position with minimal overshoot. Small secondary oscillations in the actuator position were not a concern for this experiment [16] since these are small in magnitude compared to the primary displacement. This was validated by [16] for displacements up to 2 mm and frequencies 20 Hz to 40 Hz, and the present experiments operated within these parameters.

The actuator was located at a fixed position in the wind tunnel, as were the upstream and downstream PIV areas. Vectors were obtained at discrete times (25 Hz),

rather than at every instance of time. This may have created some errors as the actuator acted on the closest known velocity vector rather than the exact velocity vector of the particles immediately above the actuator at that instant.

Future improvements could be made by using a different actuator. An actuator with faster rise time without overshoot would allow the actuator to reach its final position sooner. This would decrease the amount of time required between image capture and actuator movement, and that the RT-PIV system could be run at a higher frequency. Additionally, a higher range of motion from the actuator would enable using larger control gains to improve performance of the system.

4.5 Flow Control Effects: Comparison of Reynolds Stresses With and Without Flow Control

Using the same laser pulse as used by the upstream camera, the downstream camera captured images immediately downstream of the actuator. The shadow of the actuator is visible in the PIV images collected by the downstream camera. A sample PIV image from the downstream camera is shown in Figure 4.8.

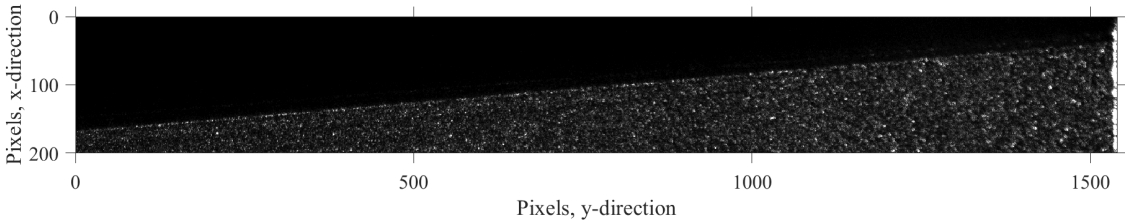


Figure 4.8: A sample image collected by the downstream camera. The floor of the wind tunnel is visible on the right hand edge of the image. The shadow of the actuator is visible diagonally through the image. Flow is in the downwards direction.

DaVis was used to obtain the vector fields for the downstream images, and these vector fields were used to calculate the Reynolds Stresses $< uu >$, $< uv >$, and $< vv >$. A sample time-averaged vector field from the DaVis results is shown in Figure 4.9, and a sample instantaneous vector field is shown in Figure 4.10. The shadow of the actuator can be seen in both Figures 4.9 and 4.10 as a region with

vectors of zero velocity magnitude.

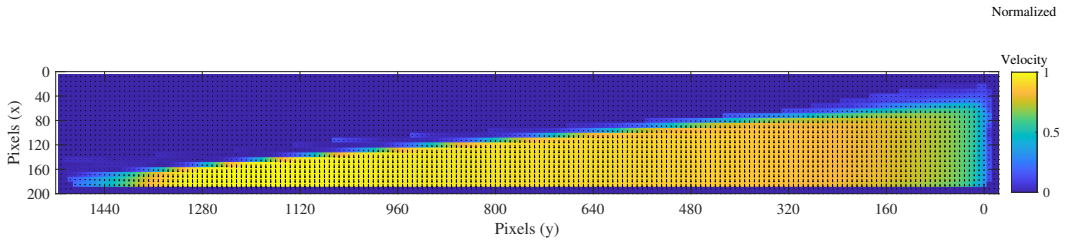


Figure 4.9: A time averaged vector field of the downstream PIV region. Vectors were normalized by dividing by the magnitude of the largest vector in this field. The wall (wind tunnel floor) is located at y = 1530 pixels.

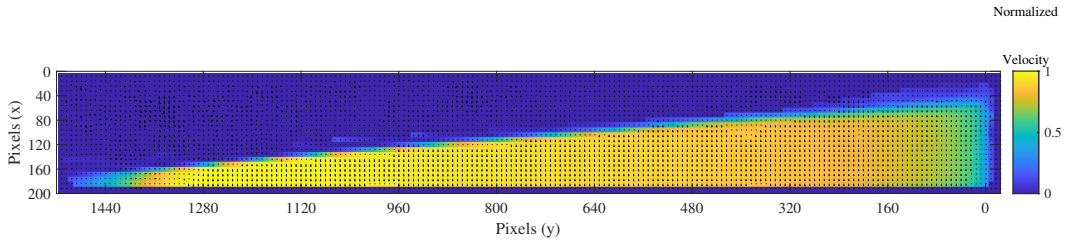


Figure 4.10: An instantaneous vector field of the downstream PIV region. Vectors were normalized by dividing by the magnitude of the largest vector in this field. The wall (wind tunnel floor) is located at y = 1530 pixels.

Although DaVis produces good results, occasionally vectors from IWs near the edge of the shadow of the actuator are outliers. This can be seen in Figure 4.9 where the vectors decrease in magnitude as they get close to the shadow of the actuator. This is likely due to the IWs being partially obscured by the actuator's shadow. Additionally, vectors near the edge of the image can be of poor quality. For this reason, a mask was used on the vector fields to isolate the region of reliable vectors. When collecting data, each test involved collecting 3200 images, or 1600 image pairs. For the first 1000 image pairs, the actuator was stationary at its midpoint flush to the wind tunnel floor. For the next 600 images, the actuator moved in response to inputs from the control code. There were four variations of tests conducted where

filter cutoff frequency was varied (5 Hz or 10 Hz), and where the vector location of interest was chosen (centred at 11 mm or centred at 8 mm). This vector location will be called the Point of Interest (POI). For each of these four variations, the test was conducted five times. A total of 20 data sets were collected in this manner. The vector sets were visually examined to ensure vectors were logical, and poor vector sets were excluded from future calculations.

First, the velocity profiles were compared before and during actuation. During actuation, there is not a noticeable change in the streamwise velocity observed for the trials with $f_c = 5$ Hz and control code vector input at 8 mm, as shown in Figure 4.11.

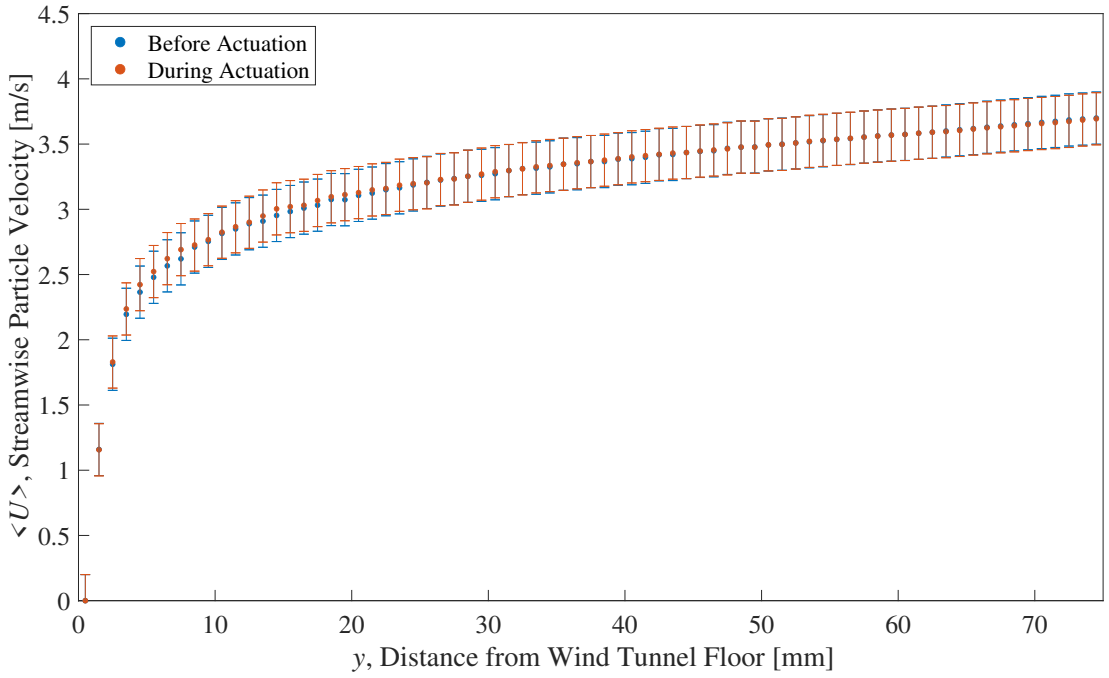


Figure 4.11: The velocity profile before and during actuation for the five trials with $f_c = 5$ Hz and control code vector input at 8 mm.

During actuation, a small increase in streamwise velocity can be observed for the trials with $f_c = 5$ Hz and control code vector input at 11 mm, as shown in Figure 4.12. However, this increase is insignificant compared to the uncertainty associated with the velocity calculations. One trial was excluded from processing due to poor vector quality.

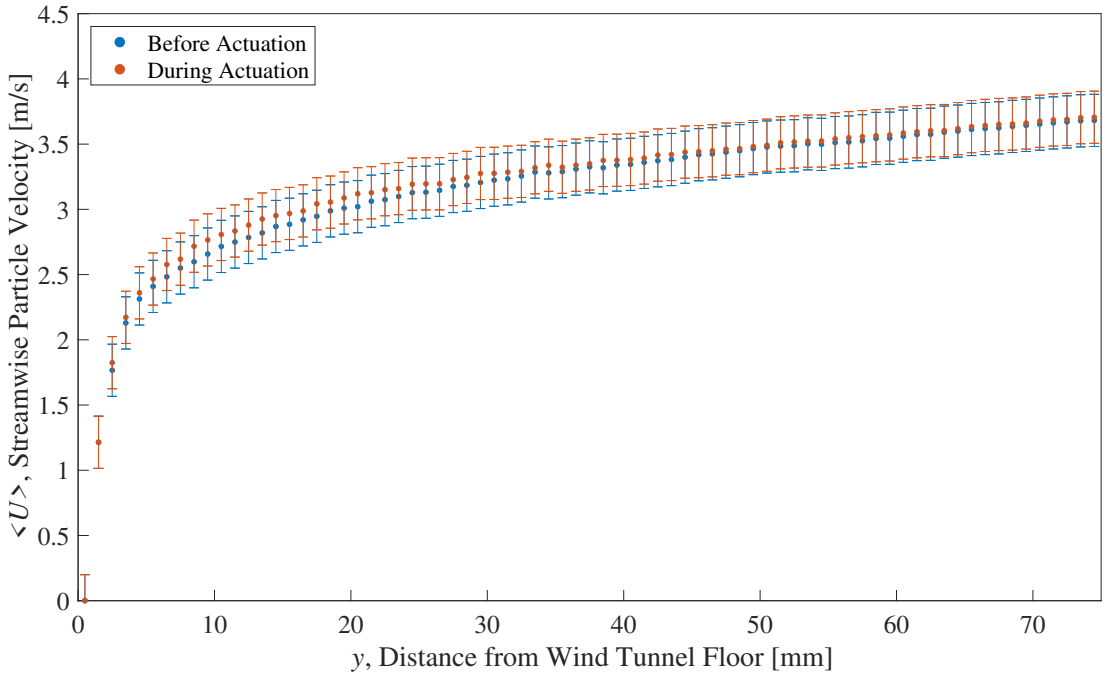


Figure 4.12: The velocity profile before actuation and during for the four trials with $f_c = 5$ Hz and control code vector input at 11 mm.

During actuation,there is not a noticeable change in the streamwise velocity observed for the trials with $f_c = 10$ Hz and control code vector input at 8 mm, as shown in Figure 4.13. One trial was excluded from processing due to poor vector quality.

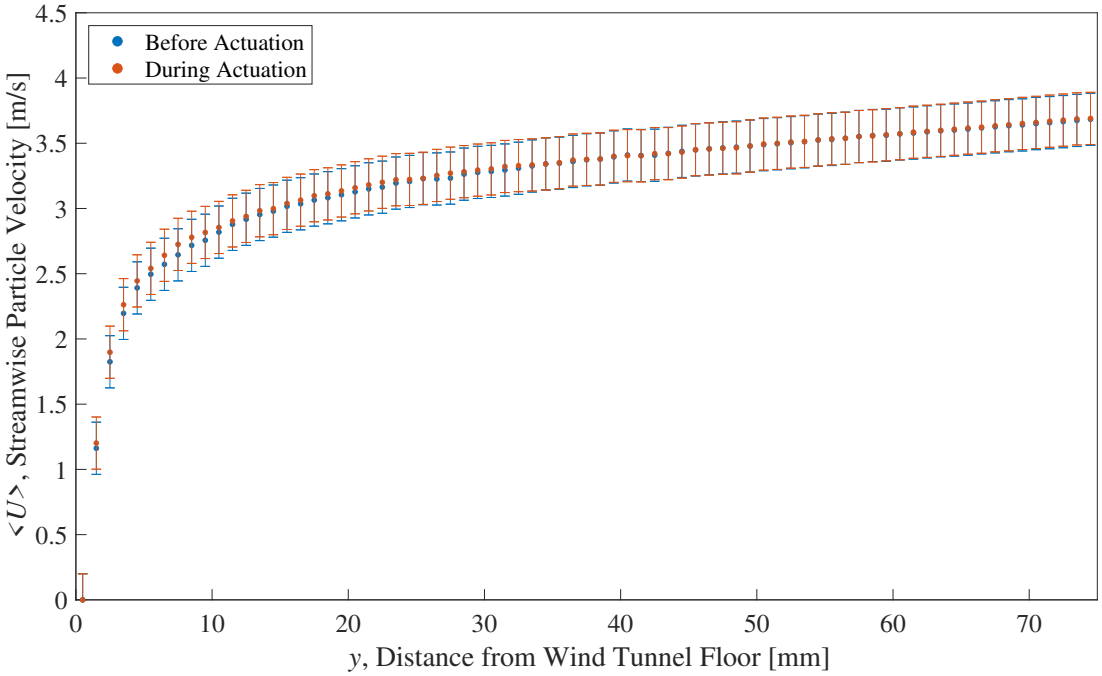


Figure 4.13: The velocity profile before and during actuation for the four trials with $f_c = 10$ Hz and control code vector input at 8 mm.

During actuation, a small increase in streamwise velocity can be observed for the trials with $f_c = 10$ Hz and control code vector input at 11 mm, as shown in Figure 4.13. However, this increase is insignificant compared to the uncertainty associated with the velocity calculations.

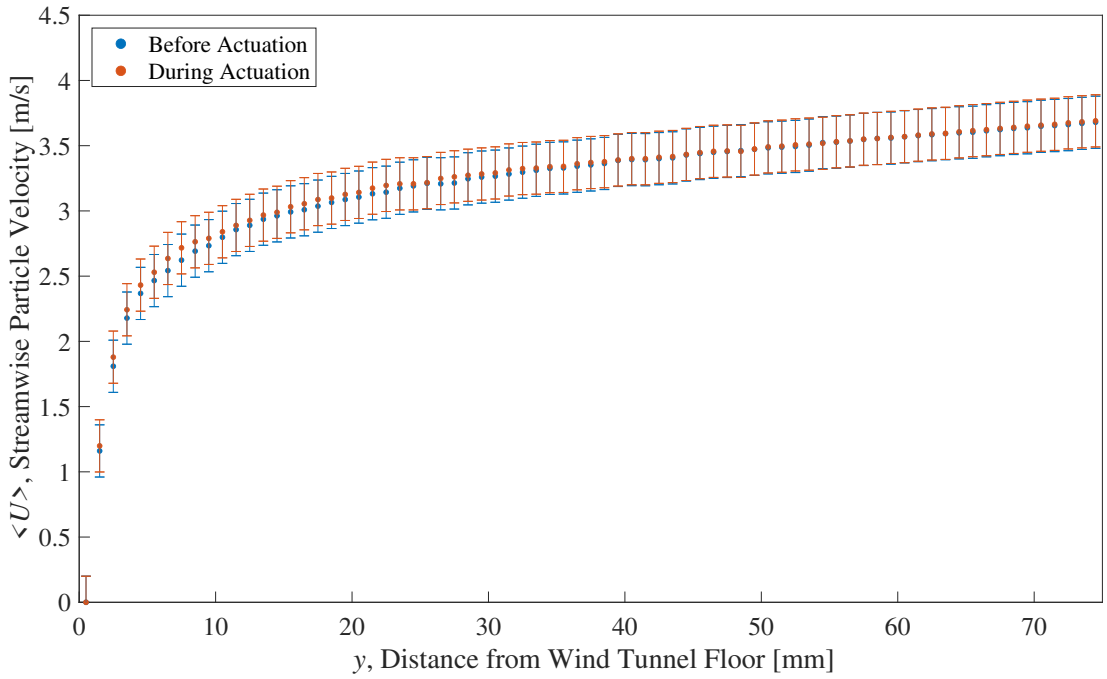


Figure 4.14: The velocity profile before actuation for the five trials with $f_c = 10$ Hz and control code input at 11 mm.

The most noticeable change in streamwise particle velocity when actuation was occurring was the increases observed on the tests where the control code input for RT-PIV was set to 11 mm. However, even in the most noticeable case, the increase in streamwise particle velocity could be explained by the errors associated with velocity calculations. Repeating this experiment with the wind tunnel freestream speed increased could help show if changes to the velocity profile have occured during actuation.

The Reynolds Stresses can be compared with and without flow control to monitor for any differences. The actuator acts on the fluid flow to eliminate the fluctuations in streamwise velocity. When successful, this should result in vectors with magnitudes

that have less deviation from the average velocity, and thus lower Reynolds Stresses.

Figure 4.15 shows that difference between the calculated $< uu >$ Reynolds stresses before and during actuation is very small in this case with $f_c = 5$ Hz and the control code input at 8 mm.

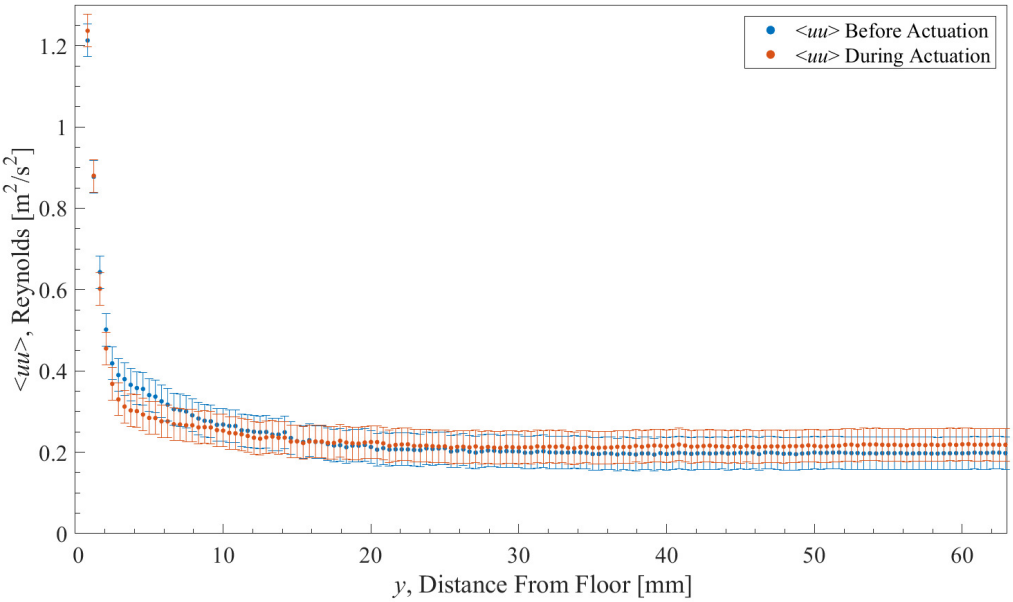


Figure 4.15: The $< uu >$ before actuation and during actuation for the five trials with $f_c = 5$ Hz and the control code input at 8 mm.

Figure 4.16 shows a small improvement during actuation in the calculated $< uu >$ Reynold's stresses with $f_c = 5$ Hz and control code input at 11 mm when compared to before actuation. The mean difference in calculated $< uu >$ Reynold's stresses before and during actuation is 20%.

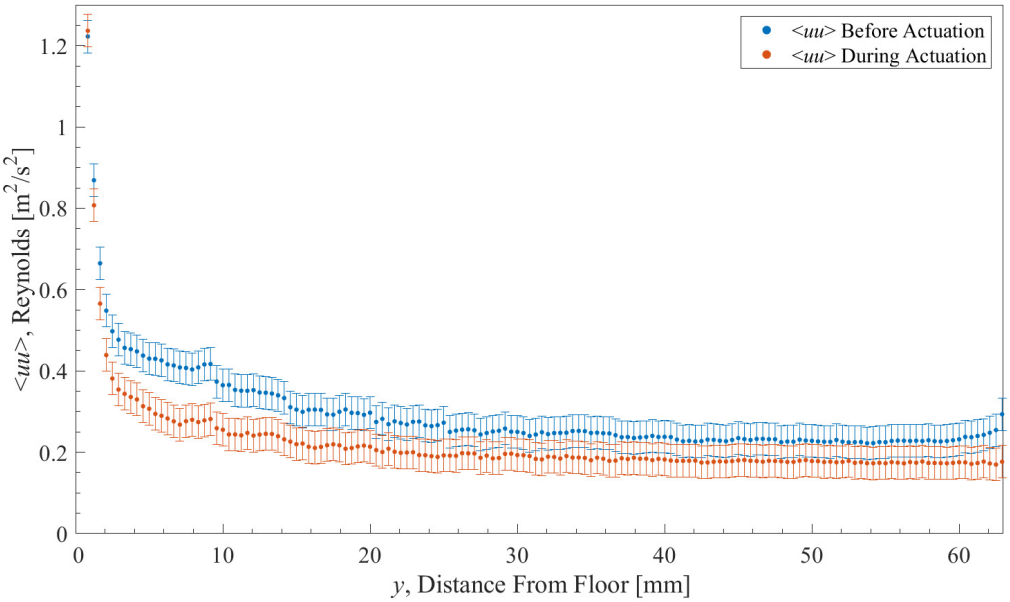


Figure 4.16: The $< uu >$ before actuation and during actuation for the four trials with $f_c = 5$ Hz and POI at 11 mm. One trial was excluded from this average due to poor vector fields.

Figure 4.17 shows that difference between the calculated $< uu >$ Reynolds stresses before and during actuation is very small in this case with $f_c = 10$ Hz and the control code input at 8 mm.

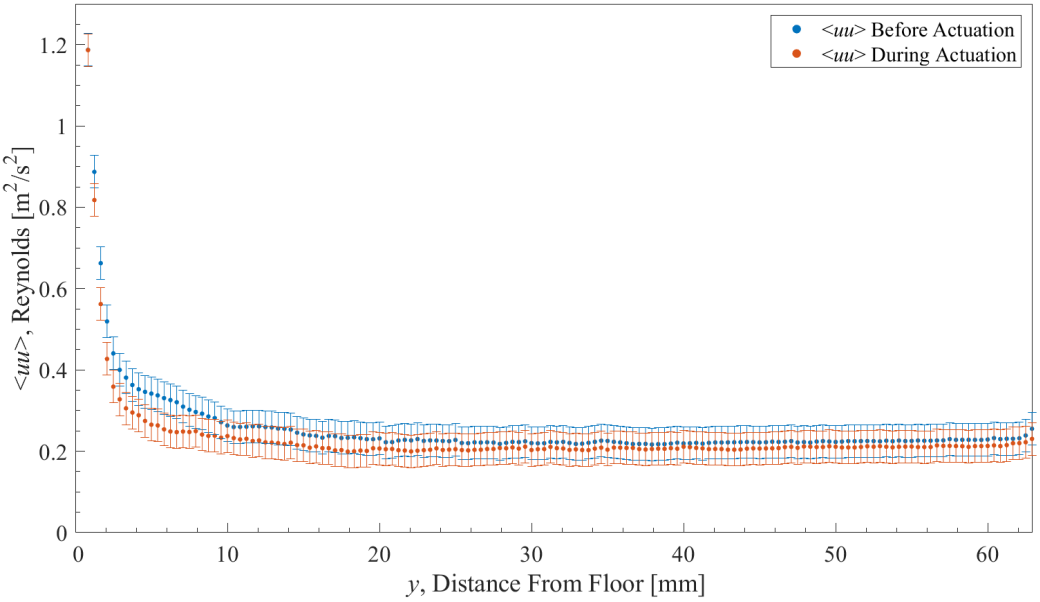


Figure 4.17: The $< uu >$ before actuation and during actuation for the four trials with $f_c = 10$ Hz and the control code input at 8 mm. One trial was excluded from this average due to poor vector fields.

Figure 4.18 shows that difference between the calculated $< uu >$ Reynold's stresses for before and during actuation is very small in this case with $f_c = 10$ Hz and the POI at 11 mm.

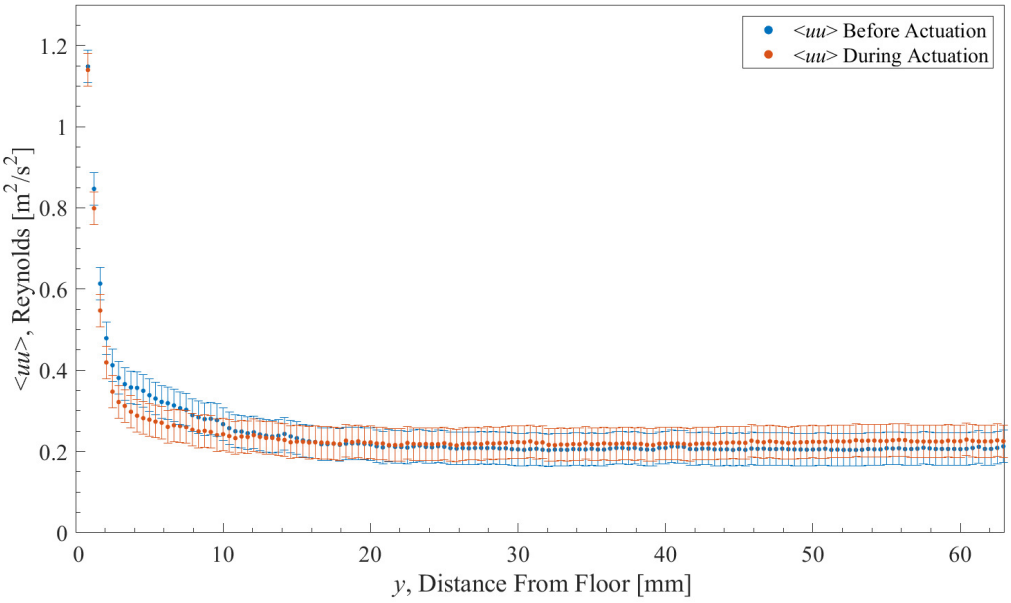


Figure 4.18: The $< uu >$ before actuation and during actuation for the five trials with $f_c = 10$ Hz and the control code input at 11 mm.

For $< uu >$, a decrease in the magnitude of the calculated Reynolds stress would imply a decrease in the variation in streamwise velocity, which would result in a decrease in drag. This is difficult to determine from the data collected. This could be due to several factors, which are discussed in the section below. The results from the test with f_c of 5 Hz with the POI at 11 mm look promising, but more data is needed to verify that the results are sufficiently significant. The differences in the the $< uv >$ and $< vv >$ Reynolds before and during actuation did not show any significant changes due to the small calculated values of v.

The statistical convergence of the Reynolds Stress profiles can be checked by plotting the Reynolds Stress at a fixed location as the number of image pairs increases. The convergence of the Reynolds stresses $< uu >$ is presented in Figure 4.19 It can be seen that the Reynolds Stress converges before 600 image pairs, meaning that the

number of image pairs collected in the experiment (1000 before actuation and 600 during actuation) was suitable to show convergence.

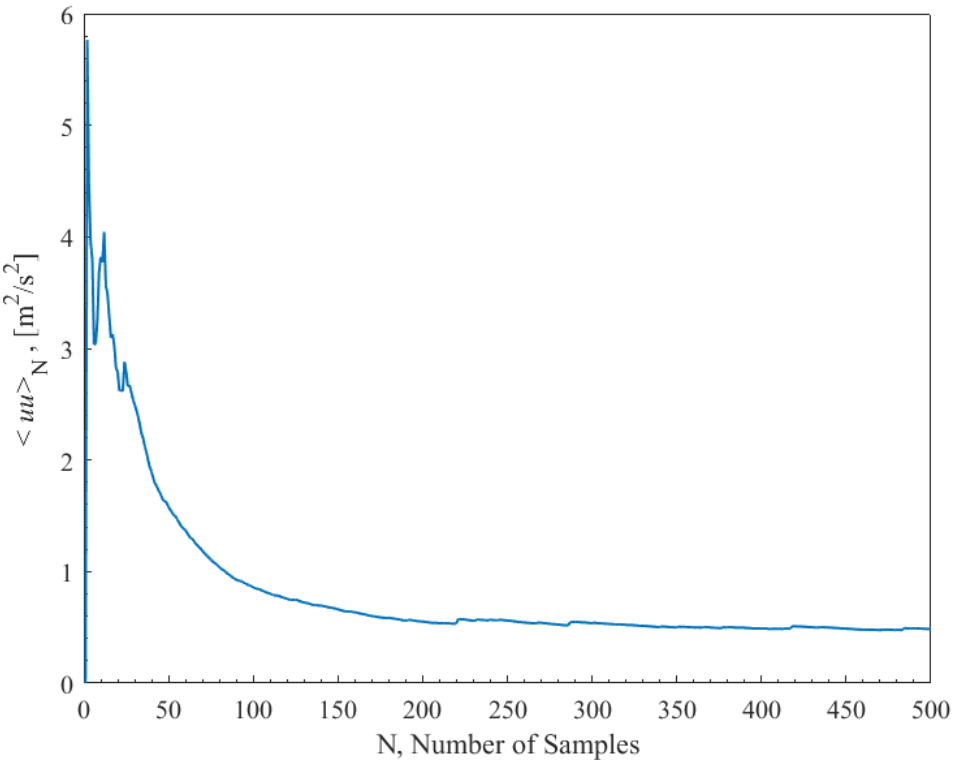


Figure 4.19: Convergence of $< uu >$ at y = 11 mm for 500 image pairs.

Chapter 5

Conclusions, Recommendations, and Future Work

A major goal of AFC is to reduce drag within the flow [7]. Researchers are investigating using AFC to improve efficiency and thus reduce energy consumption. This investigation sought to use RT-PIV in conjunction with a voice-coil actuator to achieve a decrease in Reynolds stresses for a turbulent flow in a wind tunnel. The decrease in Reynolds stresses would imply that the fluctuating portion of the velocity field of the turbulent flow has been decreased, and thus a decrease in drag of the fluid flow has been achieved. This chapter summarizes the results of this investigation and discusses recommendations for future work.

5.1 Summary of results

The objective of this research was to implement a GPU-based RT-PIV system that could be used for AFC for a turbulent flow inside a wind tunnel, and to evaluate the downstream effects of the AFC in terms of the Reynolds Stresses. Two parameters in the experiment were varied, the cutoff frequency (5 Hz and 10 Hz) of the control code's first order Butterworth filter applied to the measured velocity, and the location of the vector being input to the control code (8 mm and 11 mm above the wind tunnel floor). Each of the four resulting permutations were tested five times, where 1000 image pairs were collected at a flow location upstream of the actuator, and another

600 image pairs were collected downstream from the actuator.

The GPU based RT-PIV MATLAB algorithm worked well, delivering calculated particle displacement values within ±0.6 pix of the standard software DaVis. The execution speed of the RT-PIV MATLAB code and actuator, 20 ms from image collection to actuator motion, was acceptable for the rate of flow (4 m/s), since it was determined the particles of interest would not have moved past the actuator before it moved. The actuator used in this experiment was a previously built voice coil actuator, consisting of a deformable membrane flush with the floor to ensure a continuous surface. The actuator was able to move and reach its final position before the particles imaged by the upstream camera passed over the deformable membrane of the voice-coil actuator. The Teensy microcontroller was able to input digital signals from MATLAB and output the desired analog signals to the actuator. The RT-PIV particle displacement calculation results agreed with values calculated by DaVis, a commercial PIV software application. It was shown that the difference in calculated displacements between the RT-PIV code and the DaVis software using the same image inputs and matching parameters was within ±0.6 pix. The velocity vectors calculated from the upstream flow using the RT-PIV code were used as inputs to the active flow control law.

Downstream data was logged by a PIV system and the images were transferred to DaVis to calculate the velocity vector fields. These vectors were used to estimate Reynolds stresses with and without active flow control. In an ideal case, the Reynolds stresses would decrease in magnitude once the actuation began. The experimental results showed inconclusive evidence, however, the results from the test with a corner frequency of 5 Hz and POI at 11 mm from the floor of the wind tunnel showed the most decrease in Reynolds stress $< uu >$.

5.2 Recommendations for Future Work

Collecting more image pairs per trial would help improve the results. More image pairs would yield smoother Reynolds stress plots as well as produce more reliable statistical results. The number of images collected in the trials of this experiment were limited by the operating speed of the system. The New Wave Research Gemini PIV Nd:YAG Laser system used in the experiments has a maximum operating frequency of 30 Hz. The experiments were conducted at 25 Hz to ensure each event timing occurred at a whole number of microseconds. When the wind tunnel is seeded with fog particles, there is a finite amount of time before the fog particles have travelled through the wind tunnel. This time was visually noted to be about 2 minutes when the wind tunnel is operated at 4 m/s. Therefore, running the system at a higher frequency so that more image pairs could be collected would increase the maximum number of images that can be captured before the fog particles disperse, and this would improve the resulting Reynolds stress plots.

5.3 Laser

The laser needs changing to a laser with higher pulse frequency as well as higher power. The benefit of a laser with higher pulse frequency is smoother Reynolds stress profiles, as discussed in Section 5.2. The benefit of higher powered laser include brighter PIV images, better signal-to-noise ratio, and the ability to move the upstream data collection further upstream. Additionally, two separate lasers could be used for upstream and downstream data collection. This would allow for each laser to be focused for its specific ROI and would prevent the laser sheet from being spread too thin. Collecting more data with an improved laser setup would help verify the results by improving the quality of the data collected.

5.4 Speed

Further improvements to system speed could also be made. A faster GPU would further increase the speed of the code. The GPU used for processing in this experiment was an NVIDIA GeForce GTX 1080 Ti. Preliminary experiments conducted with a NVIDIA RTX 3090 GPU showed a significant decrease in processing time compared to the GTX 1080 Ti GPU. The NVIDIA RTX 3090 GPU was capable of obtaining vectors at speeds up to 1000 Hz when using the current RT-PIV MATLAB algorithm. Finally, re-implementing the code in C or a different compiled language could further improve execution speed.

5.5 Actuator

The actuator could be replaced. A new actuator with a larger range of motion could be implemented to see more prominent effects on the resulting Reynolds plots. A faster actuator would improve the bandwidth of the overall system.

5.6 Control Law

A very simple proportional control law was used in this experiment. Different control laws could be examined in future work. A Proportional Derivative (PD) control law could be implemented by taking into account both the current measured velocity within an image pair as well as the change in velocity between image pairs. More complicated control laws could considered as well, although difficulties may arise in implementation. Possibilities include Proportional Integral Derivitive (PID), optimal, adaptive, nonlinear, or Partial Differential Equation (PDE) based control laws. The use of Machine Learning (ML) to optimize flow control could also be examined.

Bibliography

[1] Y. A. Çengel and J. M. Cimbala, *Fluid Mechanics: Fundamentals and Applications*. McGraw Hill, 2017.

[2] X. Zhang, X. Duan, and Y. Muzychka, "Drag reduction by polymers: A brief review of the history, research progress, and prospects," *International Journal of Fluid Mechanics Research*, vol. 48, pp. 1–21, 6 2021.

[3] N. Beets, "Making modern cars more aerodynamic making modern cars more streamlined and aerodynamic," [Online]. Available: https://www.researchgate.net/publication/266057830.

[4] J. J. Spillman, "The use of variable camber to reduce drag, weight and costs of transport aircraft," *The Aeronautical Journal*, vol. 96, no. 951, pp. 1–9, 1992.

[5] S. N. Shah, A Kamel, and Y Zhou, "Drag reduction characteristics in straight and coiled tubing – an experimental study," *Journal of Petroleum Science and Engineering*, vol. 53, no. 3–4, pp. 179–188, 2006.

[6] J. W. Hamstra and D. N. Miller, "Active flow control: Enabling next-generation jet propulsion aerodynamics," in *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2001 NAE Symposium on Frontiers of Engineering*, Washington, DC: The National Academies Press, 2002, pp. 3–10.

[7] A. Seifert, T. Shtendel, and D. Dolgopyat, "From lab to full scale active flow control drag reduction: How to bridge the gap?" *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 147, pp. 262–272, 2015.

[8] A. Shmilovich and Y. Yadlin, "Flow control techniques for transport aircraft," *AIAA Journal*, vol. 49, no. 3, pp. 489–502, 2011.

[9] N. Tabatabaei, R. Vinuesa, R. Orlu, and P. Schlatter, "Techniques for turbulence tripping of boundary layers in RANS simulations," *Flow, Turbulence and Combustion*, vol. 108, pp. 661–682, 2022.

[10] M. F. Lambert, J. P. Vítkovský, A. R. Simpson, and A. Bergant, "A boundary layer growth model for one-dimensional turbulent unsteady pipe friction," in *Proceedings of the 14th Australasian Fluid Mechanics Conference*, Adelaide, Australia, 2001, pp. 929–932.

[11] H. M. Shapiro, "Aircraft measurements in the boundary layer," in *Probing the Atmospheric Boundary Layer*, H. M. Shapiro, Ed., Boston, MA: American Meteorological Society, 1986, 39—55.

[12] R. S. Subramanian, "Elements of Prandtl's boundary layer theory," [Online]. Available: https://www.researchgate.net/publication/252560127.

[13] D. M. Bushnell and J. N. Hefner, Eds., *Viscous Drag Reduction in Boundary Layers* (Progress in Astronautics and Aeronautics). American Institute of Aeronautics and Astronautics, 1990, vol. 123.

[14] S. B. Pope, *Turbulent Flows*. Cambridge University Press, 2000.

[15] M. Orsi, *Dispersion and mixing in turbulence: Data analysis, numerical simulations and modelling*, Master's thesis, Milan, Italy, 2019.

[16] B. J. Gibeau, "Advances in sensing and actuation for turbulent boundary layer control," Ph.D. dissertation, University of Alberta, 2023.

[17] C. E. Brennen, "An internet book on fluid dynamics," [Online]. Available: http://brennen.caltech.edu/FLUIDBOOK/index.htm.

[18] M. Pang, J. Wei, and B. Yu, "Numerical studies on effects of bubbles regular array on the liquid-phase turbulence," *The Canadian Journal of Chemical Engineering*, vol. 88, no. 6, pp. 945–958, 2010.

[19] M. Raffel, C. E. Willert, F. Scarano, C. J. Kähler, S. T. Wereley, and J. Kompenhans, *Particle Image Velocimetry: A Practical Guide*, Third Edition. Cham, Switzerland: Springer, 2018.

[20] J. Westerweel, "Fundamentals of digital particle image velocimetry," *Measurement Science and Technology*, vol. 8, no. 12, pp. 1379–1392, 1997.

[21] C. Dallas, M. Wu, V. Chou, A. Liberzon, and P. E. Sullivan, "Graphical processing unit-accelerated open-source particle image velocimetry software for high performance computing systems," *Journal of Fluids Engineering*, vol. 141, no. 11, p. 111 401, 2019.

[22] R. J. Adrian, "Bibliography of particle image velocimetry using imaging methods: 1917—1995," Department of Theoretical and Applied Mechanics, University of Illinois Urbana-Champaign, Urbana, IL, TAM R 817, 1996.

[23] C. Willert and J. Kompenhans, "PIV analysis of ludwig prandtl's historic flow visualization films," 2010. [Online]. Available: https://arxiv.org/abs/1010.3149.

[24] C. J. D. Pickering and N. A. Halliwell, "Laser speckle photography and particle image velocimetry: Photographic film noise," *Applied Optics, Vol. 23, Issue 17, pp. 2961-2969*, vol. 23, no. 17, pp. 2961–2969, 1984.

[25] C. Willert, "Recounting twenty years of digital piv, its origins and current trends," in *Proceedings of the 8th International Symposium On Particle Image Vecolocimetry*, Melbourne, Australia, 2009.

[26] J. J. Charonko, E. Antoine, and P. P. Vlachos, "Multispectral processing for color particle image velocimetry," *Microfluidics and Nanofluidics*, vol. 17, no. 4, pp. 729–743, 2014.

[27] D. Dabiri, "Cross-correlation digital particle image velocimetry-a review," 2007. [Online]. Available: https://www.aa.washington.edu/sites/aa/files/faculty/dabiri/pubs/piV.Review.Paper.final.pdf.

[28] E. Arik and J. Carr, "Digital particle image velocimetry system for real-time wind tunnel measurements," in *International Congress on Instrumentation in Aerospace Simulation Facilities*, Pacific Grove, California, 1997, pp. 267–277.

[29] D. Sundararajan, "The discrete fourier transform," in *The Discrete Fourier Transform: Theory, Algorithms and Applications*. World Scientific, 2001, 31––60.

[30] T. Olson, "The discrete fourier transform," in *Applied Fourier Analysis: From Signal Processing to Medical Imaging*. New York, NY: Birkhäuser, 2017, pp. 75–120.

[31] R. A. D. Gilbert, "Evaluation of FFT based cross-correlation algorithms for particle image velocimetry," M.A.Sc. thesis, University of Waterloo, 2002.

[32] *2-D fast Fourier transform - MATLAB fft2*. [Online]. Available: https://www.mathworks.com/help/matlab/ref/fft2.html.

[33] H. Yu, M. Leeser, G. Tadmor, and S. Siegel, "Real-time particle image velocimetry for feedback loops using FPGA implementation," *Journal of Aerospace Computing, Information, and Communication*, vol. 3, no. 2, pp. 52–62, 2006.

[34] S. P. McKenna and W. R. McGillis, "Performance of digital image velocimetry processing techniques," *Experiments in Fluids*, vol. 32, pp. 106–115, 2002.

[35] T. Schiwietz and R. Westermann, "GPU-PIV," in *9th International Fall Workshop on Vision, Modeling, and Visualization*, Stanford, CA, 2004, pp. 151–158.

[36] J. M. Iriarte Munoz, D. Dellavale, M. O. Sonnaillon, and F. J. Bonetto, "Real-time particle image velocimetry based on FPGA technology," in *Proceedings of the 5th Southern Conference on Programmable Logic*, Sao Carlos, Brazil, 2009, pp. 147–152.

[37] M. Kreizer, D. Ratner, and A. Liberzon, "Real-time image processing for particle tracking velocimetry," *Experiments in Fluids*, vol. 48, pp. 105–110, 2010.

[38] F. Champagnat, A. Plyer, G. Le Besnerais, B. Leclaire, S. Davoust, and Y. Le Sant, "Fast and accurate PIV computation using highly parallel iterative correlation maximization," *Experiments in Fluids*, vol. 50, pp. 1169–1182, 2011.

[39] F. Champagnat, A. Plyer, G. Le Besnerais, B. Leclaire, and Y. Le Sant, "How to calculate dense PIV vector fields at video rate," in *Proceedings of the 8th International Symposium On Particle Image Vecolocimetry*, Melbourne, Australia, 2009.

[40] M. Kobatake, T. Takaki, and I. Ishii, "A real-time micro-PIV system using frame-straddling high-speed vision," in *IEEE International Conference on Robotics and Automation*, St Paul, MN, 2012, pp. 397–402.

[41] E. Varon, J.-L. Aider, Y. Eulalie, S. Edwige, and P. Gilotte, "Adaptive control of the dynamics of a fully turbulent bimodal wake using real-time PIV," *Experiments in Fluids*, vol. 60, no. 124, 2019.

[42] F. S. McCormick, "Reactive control of velocity fluctuations using an active deformable surface and real-time PIV," M.Sc. thesis, University of Alberta, 2023.

[43] F. Ternoy, J. Dandois, F. David, and M. Pruvost, "Overview of Onera actuators for active flow control," *Aerospace Lab*, vol. 6, AL06–03, 2013.

[44] H. Choi, P. Moin, and J. Kim, "Active turbulence control for drag reduction in wall-bounded flows," *Journal of Fluid Mechanics*, vol. 262, pp. 75–110, 1994.

[45] F. Laadhari, L. Skandaji, and R. Morel, "Turbulence reduction in a boundary layer by a local spanwise oscillating surface," *Physics of Fluids*, vol. 6, no. 10, pp. 3218–3220, 1994.

[46] T. Endo, N. Kasagi, and Y. Suzuki, "Feedback control of wall turbulence with wall deformation," *International Journal of Heat and Fluid Flow*, vol. 21, no. 5, pp. 568–575, 2000.

[47] R. Rathnasingham and K. S. Breuer, "Active control of turbulent boundary layers," *Journal of Fluid Mechanics*, vol. 495, pp. 209–233, 2003.

[48] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, Sixth Edition. Pearson, 2010.

[49] *A2a1920-160umbas — basler product documentation*. [Online]. Available: https://docs.baslerweb.com/a2a1920-160umbas.

[50] *Operator's manual New Wave Research, Inc. Tempest and Gemini PIV Nd:YAG lasers*, New Wave Research, Inc., Fremont, CA, 2000.

[51] *Teensy LC (low cost)*. [Online]. Available: https://www.pjrc.com/teensy/teensyLC.html.

[52] M Amagasaki *et al.*, *Principles and Structures of FPGAs*. Springer Singapore, 2018. DOI: 10.1007/978-981-13-0824-6.

[53] D. H. Jones, A. Powell, C. S. Bouganis, and P. Y. Cheung, "Gpu versus fpga for high productivity computing," *Proceedings - 2010 International Conference on Field Programmable Logic and Applications, FPL 2010*, pp. 119–124, 2010. DOI: 10.1109/FPL.2010.32.

[54] J. Roberts and T. D. Roberts, "Use of the Butterworth low-pass filter for oceanographic data," *Journal of Geophysical Research*, vol. 83, no. C11, pp. 5510–5514, 1978.

[55] N. Juillerat and B. Hirsbrunner, "Low latency audio pitch shifting in the frequency domain," in *2010 International Conference on Audio, Language and Image Processing*, Shanghai, China, 2010, pp. 16–24.

[56] *Sensor readout time — basler product documentation*. [Online]. Available: https://docs.baslerweb.com/sensor-readout-time.

[57] *Bringing Image Data into the MATLAB Workspace - MATLAB and Simulink*. [Online]. Available: https://www.mathworks.com/help/imaq/bringing-image-data-into-the-matlab-workspace.html.

[58] "Statistical quality standards," United States Census Bureau, Statistical Quality Standard E1 - Analyzing Data, 2023.

Appendix A: Timing of Events in the RT-PIV Process

For further clarity, the timing of each event is tabulated in the table below.

Table A.1: Timing of each event in PIV process

Time From Start [μs]	**Time From Prior Event** [μs]	**Event Name**
0	-	Camera Trigger Sent
20	20	Camera Shutter Opens
819	799	Laser 1 Trigger Sent
979	160	Laser 2 Trigger Sent
999	20	Laser 1 On
1020	21	Camera Shutter Closes
1130	110	Camera Shutter Triggered
1150	20	Camera Shutter Opened
1159	9	Laser 2 On

Appendix B: Software Installation Guide

Five software components are required for this system: an operating system (OS), MATLAB, Nvidia Drivers with CUDA Toolkit, Pylon Viewer, and the Arduino IDE with Teensyduino. Software specifications and how to install each program are described in detail as follows:

1) Install an OS. This setup uses a Linux OS. It is possible to use Windows 10, however the image processing speed decreases. The system was built on Ubuntu 20.04.5. All following installation instructions are given for Linux. To install a new OS, a boot disk will need to be created. To create a boot disk, use a blank USB with at least 4 GB, and a computer with an existing OS (which does not have to be Linux).

1a) To Install Ubuntu Linux: A If Windows is used to create the boot disk, Rufus can be used to create the boot disk (https://ubuntu.com/tutorials/create-a-usb-stick-on-windows#1-overview). If Ubuntu is used to create the boot disk, it can be created directly for Ubuntu using Startup Disk Creator (https://ubuntu.com/tutorials/create-a-usb-stick-on-ubuntu#1-overview). To install Ubuntu: create a boot disk (https://releases.ubuntu.com/) and follow installation instructions (https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview) Other operating systems can also be used, see the Ubuntu tutorials for more information (https://ubuntu.com/tutorials)

2) Install Nvidia Driver and CUDA toolbox. The Nvidia Driver is required so

that MATLAB can run on the GPU. For this system, the Driver Version 11.7 and Toolkit Version 11.2 was used. Note: newer versions of Ubuntu may include Nvidia drivers automatically. To check, enter the command `nvidia-smi` in the command terminal. If nvidia is already installed, DO NOT INSTALL ANOTHER VERSION. It is recommended to install Nvidia drivers before installing MATLAB in case install errors occur.

2a) To install on Ubuntu Install the Nvidia Driver for the specific GPU from the Nvidia website: https://www.nvidia.com/Download/index.aspx https://www.cyberciti.biz/faq/ubuntu-linux-install-nvidia-driver-latest-proprietary-driver/ Install the toolbox version that is compatible with the version of MATLAB that will be used. https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=18.04&target_type=deb_local

2b) Check the drivers have been properly installed by entering command gpuDevice in MATLAB command window after MATLAB has been installed.

3) Install MATLAB (including relevant toolboxes). The image processing program is built and ran in MATLAB. The version of MATLAB used for this setup is R2022a. For either Linux OS, first download the Linux release of MATLAB to the Downloads folder (https://www.mathworks.com/downloads/web_downloads).

3a) For Ubuntu: Open the terminal and run the following commands

`cd Downloads`	Changes directory to MATLAB zip
`sudo apt install unzip`	Install an unzip tool
`xhost +SI:local user:  root`	Allow root user access to X Server
`mkdir matlab`	Make a directory for MATLAB
`unzip -qq matlab*.zip -d matlab`	Unzip MATLAB to MATLAB directory
`cd matlab`	Change to MATLAB directory
`sudo ./install`	Install MATLAB

The MathWorks product installer window should pop up. Follow the instructions given onscreen. The required toolboxes are GPU coder, Image Acquisition Toolbox,

Image Processing Toolbox, Computer Vision Toolbox. Ensure the folder selected for installation is one with write access.

3b) Set up Matlab path variables: https://www.mathworks.com/help/gpucoder/gs/setting-up-the-toolchain.html#mw_62a2b85b-1ed1-4c55-a1d2-67d93040cac5

3c) Add required MATLAB Toolboxes. Finally, once MATLAB is installed open the Add-Ons from the Home menu. Select "Get-Add-Ons". Search for and install the "Image Acquisition Toolbox Support Package for GenICam Interface". This is the driver for MATLAB to run the Basler camera. Search for and install the "MATLAB Support Package for Arduino Hardware". This package is needed to run the Teensy, which outputs the TTL step signal to control the laser output time. The Arduino IDE is not required to interface the Teensy with MATLAB; however, it is required to create the programs that the board runs.

4) Install Pylon Viewer (pylon 7.1.0) Pylon Viewer is the proprietary software from Basler that is used to control its cameras. In order for MATLAB to access the required drivers to communicate with the camera, the pylon package must be downloaded and installed 4a) For Ubuntu: download and run the Debian Installer Package from the Basler webpage https://www.baslerweb.com/en/downloads/software-downloads/#os=linuxx8664bit;version=all;type=pylonsoftware;language=all

5) Install the Arduino IDE The Arduino IDE installation package can be downloaded from https://www.arduino.cc/en/software. For this system, Arduino IDE 1.8.19 was used. Ensure that the selected version is compatible with the Teensyduino extension. Compatible versions are listed on https://www.pjrc.com/teensy/td_download.html.

5a) Extract the downloaded arduino-1.8.19-linux64.tar.xz (or similarily name) file. It is important to know the location of the extracted file, since Teensyduino will need this information. In a terminal, navigate to the folder with the extracted Arduino package. Enter the command `sudo chmod +x install.sh` to make the Arduino script executable. Finally, enter the command `./install.sh` to run the install.

5b) The Teensyduino add on for Arduino IDE can be installed from https://www.pjrc.com/teensy/td_download.html. First Download the Linux udev rules (located at https://www.pjrc.com/teensy/00-teensy.rules) and copy the file to /etc/udev/rules.d. In the terminal, enter `sudo cp 00-teensy.rules /etc/udev/rules.d/`. Finally, enter the following commands to run the installer by adding execute permission and then execute it.

```
chmod 755 TeensyduinoInstall.linux64
./TeensyduinoInstall.linux64
```

Appendix C: Oscilloscope Output

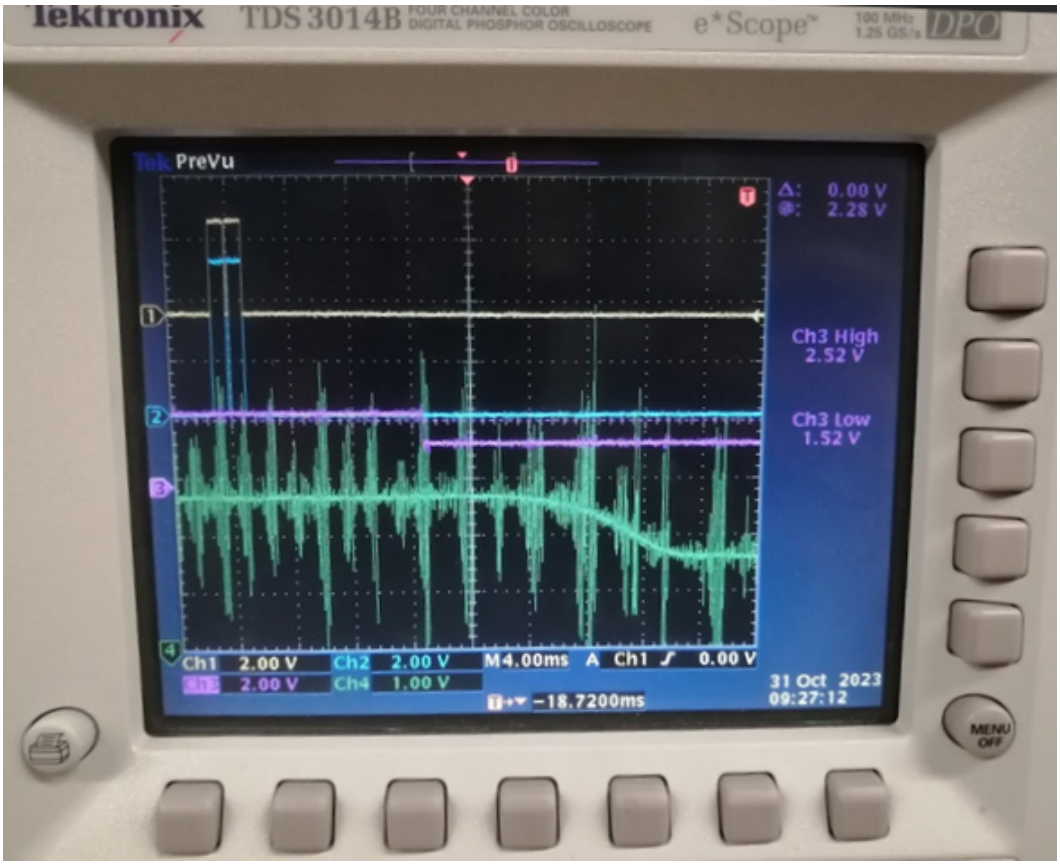


Figure C.1: Oscilloscope sample output showing the camera input in yellow, camera output in blue, Teensy output/actuator input in pink, and actuator output in green. It can be seen the actuator starts to move at approximately 20 ms after the first of the two images in the pair is collected, and reaches its endpoint approximately 30 ms after the first of the two images is captured.