A Characterization of Pulsatile Flow by Positron Emission Particle Tracking

A Thesis Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Nitant Piyushbhai Patel
December 2017

ACKNOWLEDGEMENTS

There are few individuals whom I would like to thank for their support and assistance towards this thesis. First, I would like to thank my mentor, Dr. Arthur Ruggles for his advice, support and guidance throughout my academic career at the University of Tennessee, Knoxville. Likewise, I would like to thank my committee members, Dr. Lawrence Heilbronn and Dr. John Auxier II for their time and valuable feedback. I would also like to acknowledge and thank Matthew Buttrey, Seth Langford, Roque Santos and Cody Wiggins for their contribution towards this work. Finally, I would like to give a special acknowledgement to the US Department of Energy, National Nuclear Security Administration and the Stockpile Stewardship Academic Alliance Program for funding this work.

ABSTRACT

Multi-positron emission particle tracking (M-PEPT) is a Lagrangian measurement technique that can be used to evaluate flow attributes. This study seeks to characterize steady flow in a tube, pulsatile flow in a circular and pinched cross section tube and explore particle detection uncertainties of our M-PEPT measurement. A pump driven flow loop and motorized ball valve are designed to create the pulsatile flow of frequency 2.1 Hz in a 19 mm diameter and 3.2 mm thick Masterkleer PVC tube. Anion exchange resins of 600-800 microns are labeled with fluorine-18 (half-life: 109.8 minutes) and pumped through the flow loop at flow velocity near 1 m/s, Womersley number near 70 and approximate Reynolds number of 20,000. Bulk flow velocity and pressure are measured to define the pulsatile flow features. In addition, particle trajectories are collected using M-PEPT and synchronized to a trigger signal produced by laser-photodiode mounted on the motorized ball valve. Trajectories are divided into 20 equally spaced time gated frames in each pulse cycle. A three-dimensional velocity field is generated at each phase of the pulse cycle by integrating time gated data from several hundred pulse cycles. The average location uncertainty in the M-PEPT measurement is calculated to be near 0.31 mm in the x and y-direction and 0.2 mm in z. The turbulent kinetic energy and diffusion characteristics of a steady flow is also presented.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ATTACHMENTS

CHAPTER ONE
INTRODUCTION

Multi-positron emission particle tracking (M-PEPT) is a Lagrangian flow measurement technique used to resolve 4-dimensional flow characteristics. In M-PEPT, the fluid flow is seeded by tracer particles labeled with a positron emitting radionuclide, such as ^{18}F, ^{22}Na, ^{68}Ga, ^{75}Br and ^{124}I [1]. The emitted positron reacts with neighboring electron, producing positronium with a half-life of 10^{-10} second [2]. Positronium decays by the annihilation reaction, generating two coincident and collinear gamma rays of 511 keV each at approximately 180°. The twin gamma rays are identified by using detection systems similar to the positron emission tomography (PET) scanner or positron camera [3]. The detection sites of collinear gamma rays are used to form coincident lines (CLs). A region where CLs are clustered together corresponds to the particle location. CLs detected by the PET scanner are separated into time intervals, and CLs formed in each interval are used to locate particles. This method was initially developed by a group at the University of Birmingham, led by Dr. David Parker, to track a single particle in various engineering systems [4]. Later, this particle location method is used by Yang et al. [5] to simultaneously track multiple particles of different activity in a series of experiments. Others have implemented various particle location approaches in their studies [6], [7]. A modified line density method coupled with optical feature point identification (FPI) is used herein to track particles without any previous knowledge of the particle population, activity, and position [8], [9]. This method is known as multi-particle PEPT (multi-PEPT). This study investigates turbulent flow features in a steady flow, characterizes pulsatile flow in a circular and pinched cross-sectional tube, and characterizes capability of the multi-PEPT algorithm for flow turbulence measurement.

Turbulent flow is distinguished by fluctuations in the mean fluid flow. The fluid flow through pipes, air ducts, atmosphere, ocean currents and even human blood flow are often turbulent. The blood flow through the cardiovascular system, as well as fluid flow through mechanical and hydraulic systems, may change periodically at a set frequency. This type of flow is denoted as pulsatile flow in this thesis. The pulsatile flow is examined by splitting each oscillatory cycle into 20 equal time intervals and the velocity field is reconstructed in each interval. In addition, turbulent kinetic energy (TKE) and diffusion characteristics are explored in the steady flow.

Much research associated with the pulsatile flow is conducted for hemodynamics applications. Focus of the majority of the studies is to detect cardiovascular diseases through examination of abnormal flow patterns.

Numerical and experimental hemodynamics research examines the impact of the non-Newtonian fluid, abnormal turbulence characteristics in large arteries, flow behavior through stenosis and pressure gradients associated with the flow. Some studies have analytically solved the pulsatile flow in circular and elliptical cross section by simplifying and solving the Navier Stokes equations [10], [11]. A few high-resolution experimental studies have been conducted for pulsatile flow. Trip et al. [12] designed and performed planer Particle Image Velocimetry (PIV) study to understand the transition region between laminar and turbulent flow to show potential flow features in the presence of a cardiovascular disease. Another PIV study interrogated fluid behavior pre- and post-stenosis throughout the pulse cycle, to examine flow patterns during plaque built up [13]. Benam *et al*. [14] created a flow system to understand pressure gradients related to arterial wall stiffness and increased viscosities to demonstrate aging effects on the brachial artery.

In this work, steady flow in a Masterkleer Poly-vinyl chloride (PVC) tube of 19 mm diameter is measured using PEPT. A three-dimensional velocity profile along with TKE and diffusion characteristics are presented. The pulsatile flow attributes are measured in a circular and pinched cross section tube at a pulse frequency of 2.1 Hz, flow velocity of 1 m/s and Reynolds number near 20000. In addition, the periodic pressure characteristics are measured at the tube inlet and exit, and pressure phase angles and pressure wave velocity are calculated. This work examines the measurement capability of PEPT in a transient flow and advances our mission towards resolving turbulent flow features.

CHAPTER TWO

THEORY

Fluid consist of states of matter which continuously deform under an applied force. Flow is a movement of the fluid subject to applied stresses and boundary conditions. The fluid flow can be analyzed in two frames, Eulerian and Lagrangian. In a Eulerian frame, the fluid flow is measured at a specific coordinate location fixed to the structures constraining the flow field. While, in Lagrangian, the fluid flow is measured by following individual fluid elements throughout the flow field. The Eulerian point of view is most frequently used to assess velocity in engineering systems due to ease of conducting a measurement at a fixed location. However, it is difficult to resolve many flow behaviors using a Eulerian frame. Hence, Lagrangian flow measurement is necessary to conduct in-depth fluid flow study in the frame native to the fluid elements. Many techniques such as Particle Tracking Velocimetry (PTV), High-Speed Videography (HSV) and PEPT are used to evaluate the flow field in terms of Lagrangian frame. However, PTV and HSV, amongst others, require optical access for the measurement. PEPT can measure flow in opaque media at resolution similar to some of the optical techniques.

Laminar vs. Turbulent

In the past, many researchers have studied fluid flow in a pipe due to its wide variety of applications. One of the classical fluid flow phenomena was observed and explained by Osborne Reynolds [11]. He conducted a dye experiment to understand fluid behavior at increasing velocities. Upon injection of the dye at low velocities, he observed the dye tracks in a straight line, parallel to the flow. On the other hand, at high velocities, chaotic fluctuations, diffusion, and mixing of the dye tracks were observed. These two flow features were broken into two regimes known as laminar and turbulent flow. Through many experiments, he noticed that these flow attributes were not only a function of velocity (v) but also function of density (ρ), hydraulic diameter (h_d) and dynamic viscosity (μ) of the fluid in the non-dimensional form known as Reynolds number.

$$Re = \frac{\rho v h_d}{\mu} = \frac{Inertial\ Force}{Viscous\ Force} \tag{1}$$

The following limits are established to differentiate between each of the flow regimes.

Laminar Flow: Re < 2300

Transition Flow: 2300 < Re < 4000
Turbulent Flow: Re > 4000

For pulsatile flow, the Reynolds number is augmented with the Womersley number [11].

$$\alpha = L\left(\frac{\omega\rho}{\mu}\right)^{1/2} = \frac{Inertial\ Force}{Viscous\ Force} \tag{2}$$

Where L is the diameter and ω is the angular velocity of the pulsation.

In this work, the turbulent flow field is measured for the steady and pulsatile flow. The turbulent flow is defined by irregular fluctuations in the fluid flow. Reynolds observed these characteristics in his experiments and he decomposed a parameter of interest into two components: time averaged mean and fluctuating component as seen in equation 3. Figure 1 shows the velocity measurement collected with an orifice plate for a steady flow through the PVC tube. The data have a frequency of 11-14 Hz present. These 11 to 14 Hz frequencies were not presented at the flexible tube inlet or exit pressure locations. This might be a result of resonance in the tube line of 157 cm length connecting the differential pressure transducer and orifice plate.

$$v(t) = \bar{v} + v'(t) \tag{3}$$

Where:

$v(t)$: Velocity
$\bar{v}$: Mean velocity
$v'(t)$: Velocity fluctuations

The Reynolds decomposition can be used to find multiple characteristics of turbulent flow. One of the features discussed in this thesis is the concept of turbulent kinetic energy. Turbulent kinetic energy is kinetic energy associated with the flow turbulence in the Lagrangian frame and it is defined by equation 4. This is used to understand turbulence features in steady and pulsatile flow. The derivation from mean flow kinetic energy to turbulent kinetic energy is presented below. Herein the $2\bar{u}u'$ term is not considered in the turbulent kinetic energy calculation, consistent with a time averaged TKE value where time averaged values for time varying components of velocity are zero.

$$KE = \frac{1}{2}\,(u^2 + v^2 + w^2)$$

$$KE = \frac{1}{2}\,(\,(\bar{u} + u')^2 + (\bar{v} + v')^2 + (\bar{w} + w')^2$$

$$KE = \frac{1}{2}\,((\overline{u^2} + 2\bar{u}u' + u'^2) + (\overline{v^2} + 2\bar{v}v' + v'^2) + (\overline{w^2} + 2\bar{w}w' + w'^2)$$

$$TKE = \frac{1}{2}\,(\overline{u'^2} + \overline{v'^2} + \overline{w'^2}) \qquad (4)$$

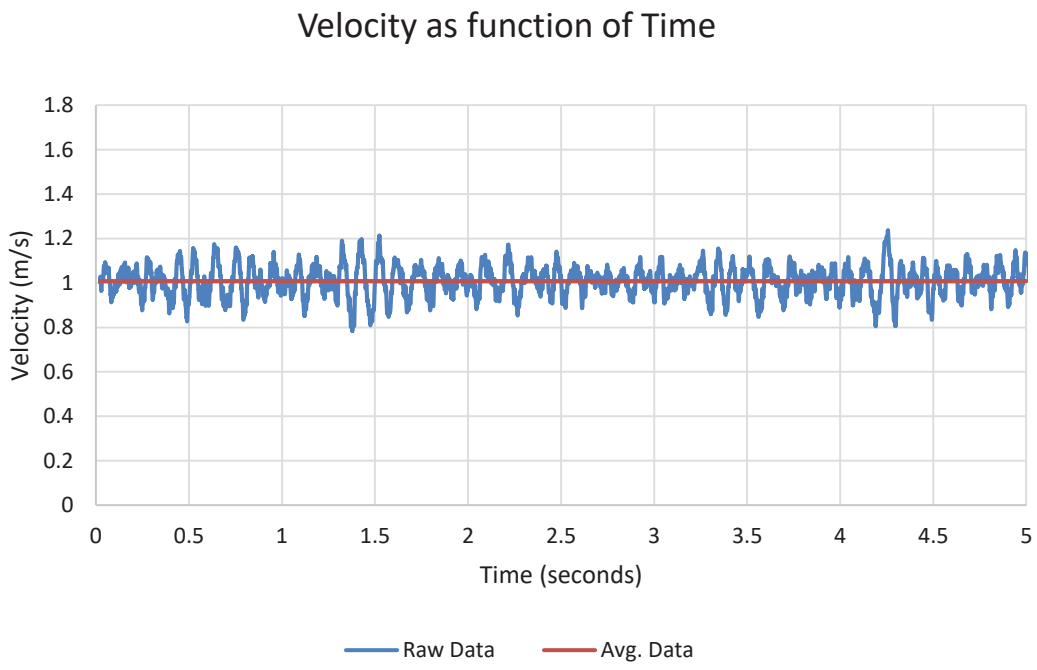


Figure 1. Illustration of a turbulent velocity features in a steady flow

CHAPTER THREE

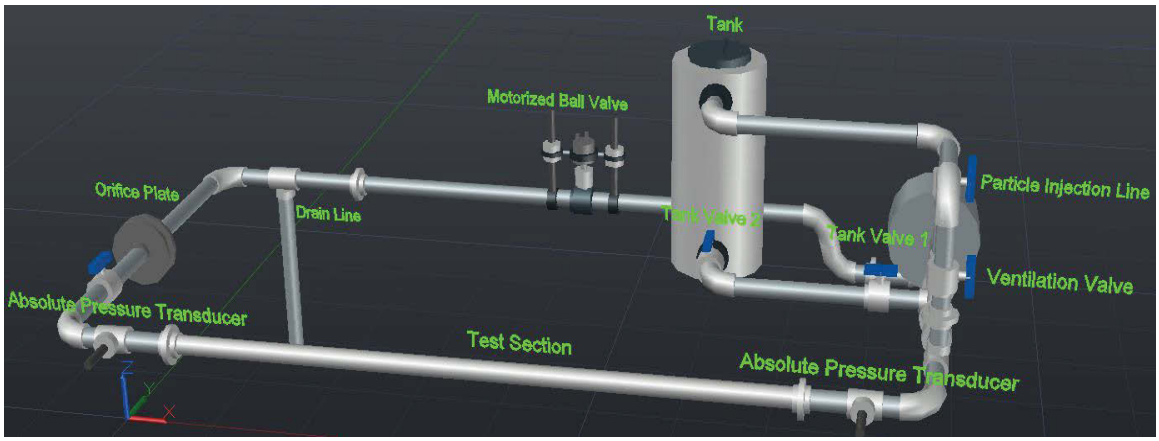
EXPERIMENT

A pump driven flow loop built from a schedule 40 PVC tube of 19 mm diameter is used to conduct series of experiments. Figure 2 shows flow loop with main components labeled. Table 1 shows the list of instrumentations used for the experiments. In addition to pressure instruments, a custom motorized ball valve composed of the gear motor, acetal clamp-on shaft coupling and ball valve (*manufacturer: GF Piping System model number: 161375018*) is constructed and installed to generate pulses in the system. A laser-photodiode sensor is used with a reflective tape on the valve drive to create a trigger signal, which is sent to the LabVIEW data acquisition and to the Inveon PET scanner. The test section used for the experiments is a Masterkleer PVC tube of 19 mm diameter and 3.2 mm thick walls. An AutoCAD drawing of the flow loop inside and outside the PET scanner is seen in Figure 2. The main components such as recirculation valve, vent line, and particle injection line are used for controlling the flow rate, eliminating air bubbles and injecting particles. For each experiment, the M-PEPT measurement is collected 30 diameters downstream of the test section inlet. The particles are tracked in a 127 mm long field of view using the Siemens Inveon PET Scanner. The PET scanner used herein consists of 25,600 lutetium oxyorthosilicate (LSO) detection crystals evenly spread across a scanning bore of 161 mm diameter and an axial length of 127 mm [15]. The detection crystals are arranged in 16 detection blocks with 400 detection crystals per block. Each block is connected to a 64 channel position sensitive photomultiplier tube. The peak efficiency of the scanner is near 6% at the center of the field of view (FOV) [15]. The current scanner provides time stamps in the recorded CL every 200 μs.

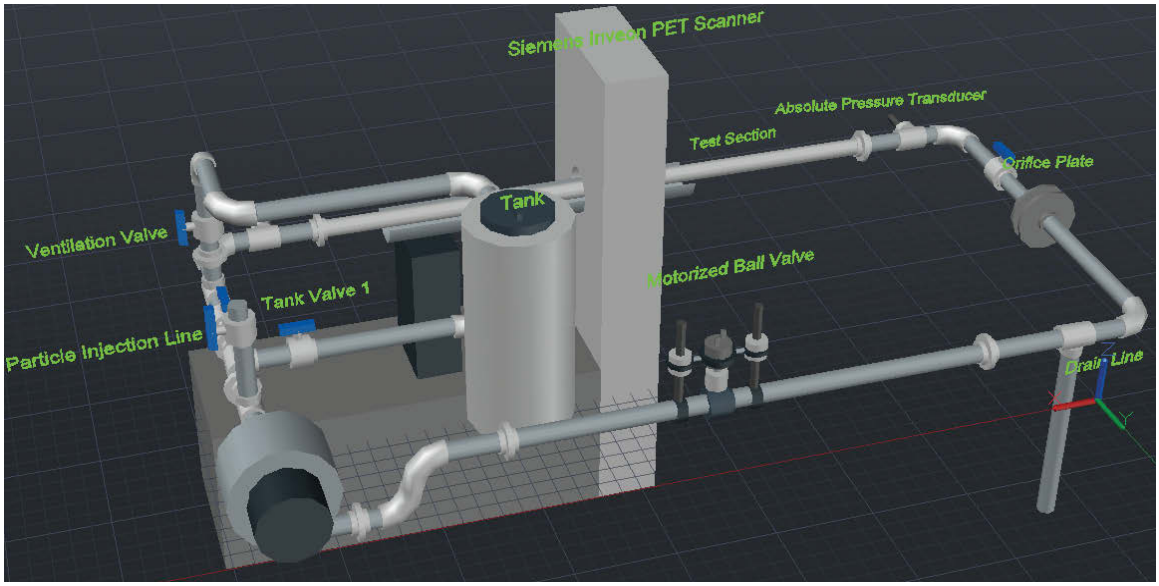
Absolute Pressure Transducers

Two absolute pressure transducers are used to assess the pressure drop across the 116 cm test section. Figure 3 shows the pressure transducer used herein. In the absolute pressure transducer, pressure from the fluid is applied to the end of the pressure transducer housing an electrically conductive diaphragm. The pressure deforms the diaphragm causing a change in capacitance which is converted into a voltage signal. The measured voltage signal is converted to pressure signal by linearly fitting the calibration points provided by the manufacturer, as seen in Figure 4 and Figure 5. The first transducer (444462) is located near the inlet of the test section, and the second transducer (427575) is located near the outlet, per Figure 2. The frequency response of these transducers is 1 kHz.

Figure 2. AutoCAD drawing and image of the flow loop A). Flow loop with components labelled B). Flow loop inside the PET scanner. C). Image of the experimental setup

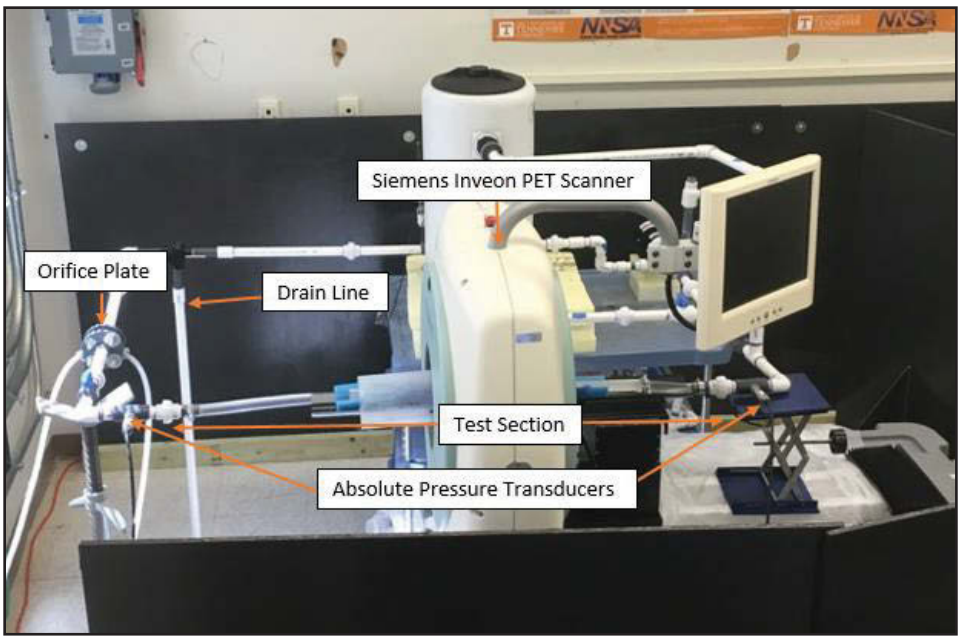


(A)



(B)

Figure 2. Continued



(C)

Figure 2. Continued

Table 1. Instrumentation specification

Instrumentation	Quantity	Manufacturer	Model Number
Absolute Pressure Transducer	2	Omega	PX409-150A5V
Data Acquisition Module	1	National Instruments	9215
Data Acquisition Software	2	National Instruments	LabVIEW 2014 and LabVIEW 2015
Differential Pressure Transducer	1	Omega	PX409-005DWU5V
Laser-Photodiode	1	Monarch Instrument	SPSR-115/230
Particle Size Analyzer	1	Malvern	Mastersizer 3000
Sample Dispersion Unit	1	Malvern	Hydro LV
Orifice Plate	1	Dwyer	PE-B-3
PET scanner	1	Siemens	Inveon Dedicated PET

Figure 3. Visual representation of the absolute pressure transducer

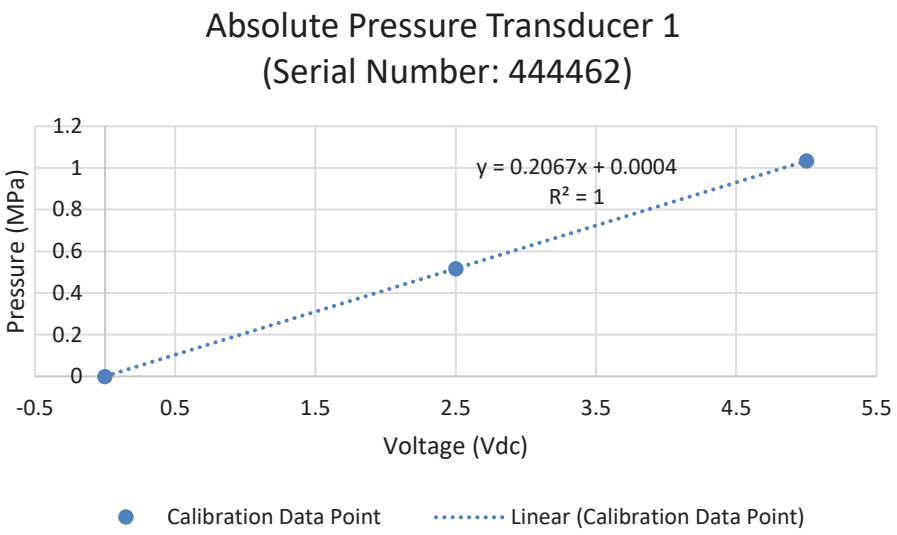


Figure 4. Calibration curve of the absolute pressure transducer located near the inlet edge of the test section

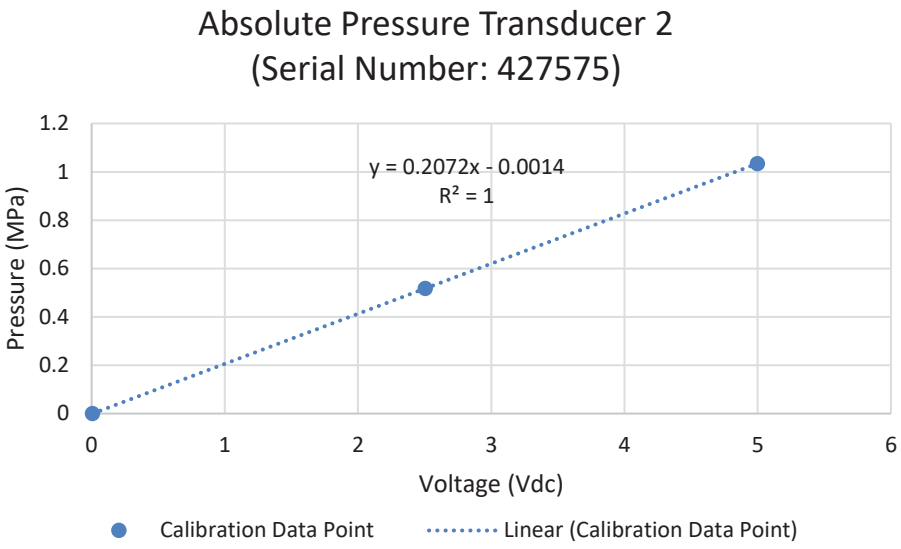


Figure 5. Calibration curve of the absolute pressure transducer located near the outlet edge of the test section

Differential Pressure Transducer

The differential pressure transducer is coupled to the orifice plate flow meter to measure the pressure drop across the orifice plate. Figure 6 shows an image of the differential pressure transducer. The operational method is similar to the absolute pressure transducer, but for this instrument, there are two ports for the measurement. The two pressures are applied to the diaphragm on opposite sides. The measured voltage signal is converted to the differential pressure signal by linearly correlating the calibration data provided by the manufacturer. Figure 7 shows the calibration curve of the differential pressure transducer in units of kPaD (kilopascal differential).

Orifice Plate Flow Meter

The orifice plate flow meter is seen in Figure 8. The orifice plate constricts the cross-sectional area of the moving fluid causing pressure drop across the orifice plate. The pressure differential across the plate is measured by utilizing the two holes located on the orifice body, one before on the inlet edge and another on the outlet edge of the orifice. Differential pressure is measured using the differential pressure meter presented in the previous section. Differential pressure is correlated to volumetric flow rate using equations 5 and 6 provided by the manufacturer. For this research, an orifice plate with 14.73 mm bore is used to measure a pressure differential yielding max volumetric flow rate near 1.5 kg/s.

$$Q = 44.748 * d^2 * K * Y * Fa * \sqrt{\frac{hw}{P_L}} \quad (5)$$

$$K = 0.5959 + 00312\ \beta^{2.1} - 0.1840\ \beta^8 + 91.71\ \beta^{2.5}\ R_n^{-0.75} * \frac{1}{\sqrt{1-\beta^4}} \quad (6)$$

Where:

Q = flow rate (GPM)
d = diameter of the orifice bore (inches)
K = flow coefficient
Y = expansion factor (water = 1)
Fa = thermal expansion factor (water = 1)
hw = differential pressure (inches water column)
P_L = density at line conditions (lbs./ft^3)
β = diameter ratio of the bore and pipe

The orifice plate is only used to measure the reference flow rate at steady flow

Figure 6. Image of a differential pressure transducer

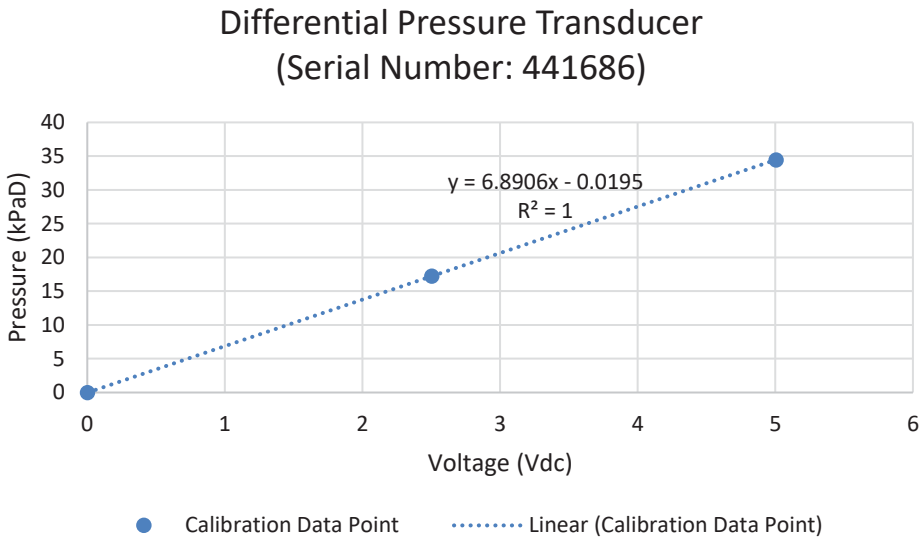


Figure 7. Calibration curve of the differential pressure transducer

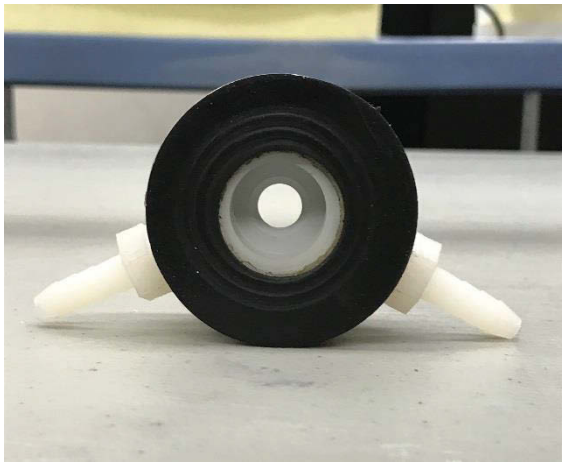


Figure 8. Visual representation of the orifice plate

conditions. Flow rate calibrations are not known for the pulsatile flow.

Laser-Photodiode

The Laser-Photodiode shown in Figure 9 is used to acquire a time stamp at each pulse cycle during the pulsatile flow experiments. A small piece of reflective tape is attached to the clamp-on shaft coupling and laser-photodiode is aligned towards the tape as shown in Figure 10. The laser is reflected back to the photodiode upon hitting the reflective tape and a 5-volt TTL (transistor-transistor logic) signal is generated and sent to the DAQ (Data Acquisition) and PET scanner. The TTL signal causes a time mark to be placed in the list mode of the PET scanner output. The TTL signal is utilized to synchronize the PEPT data and LabVIEW collected flow loop pressure data for individual pulse cycles.

Motorized Ball Valve Calibration

The motorized ball valve used herein is an evolved custom design capable of creating very regular pulsatile flow. Figure 10 shows the three generations of motorized ball valves. The data presented in this thesis were collected using the second generation (pulsatile flow in pinched cross-section) and third generation (steady flow and pulsatile flow in circular cross-section) motorized ball valve. The motorized ball valves are constructed by coupling the 12-volt DC-powered gear motor shaft to the ball valve shaft using a clamp-on shaft coupling. In order to power the gear motor, an AC to DC converter (HP 6365A) is used as seen in Figure 11. The converter used was analog, therefore during each experiment, the motorized ball valve was calibrated using a video camera. A video of the ball valve movement is recorded for 1 minute and converted into suitable frames per minute (fpm). The video is then analyzed using the VLC player and pulse cycle frequency is measured on a frame per frame basis. If the valve is moving at desired frequency than the voltage level is kept. If not, then the voltage is readjusted and the process is repeated. In service, valve rotating frequency is also known through examination of TTL trigger signal recorded in the scanner CL data to 200 μs resolution.

LabVIEW and DAQ System

A data acquisition system by national instruments is used to record and assess the measurements from the test section inlet and exit pressure transducers, laser photodiode, and orifice plate differential pressure transducer. A producer/consumer architecture in LabVIEW is used to acquire and process the data. In this architecture, the producer loop receives the data at very high frequency and consumer loop processes and stores the data in the order of acquisition, but at a slower processing rate than the producer loop. Figure 12 below shows the producer/consumer architecture implemented for the

Figure 9. Image of a laser photodiode

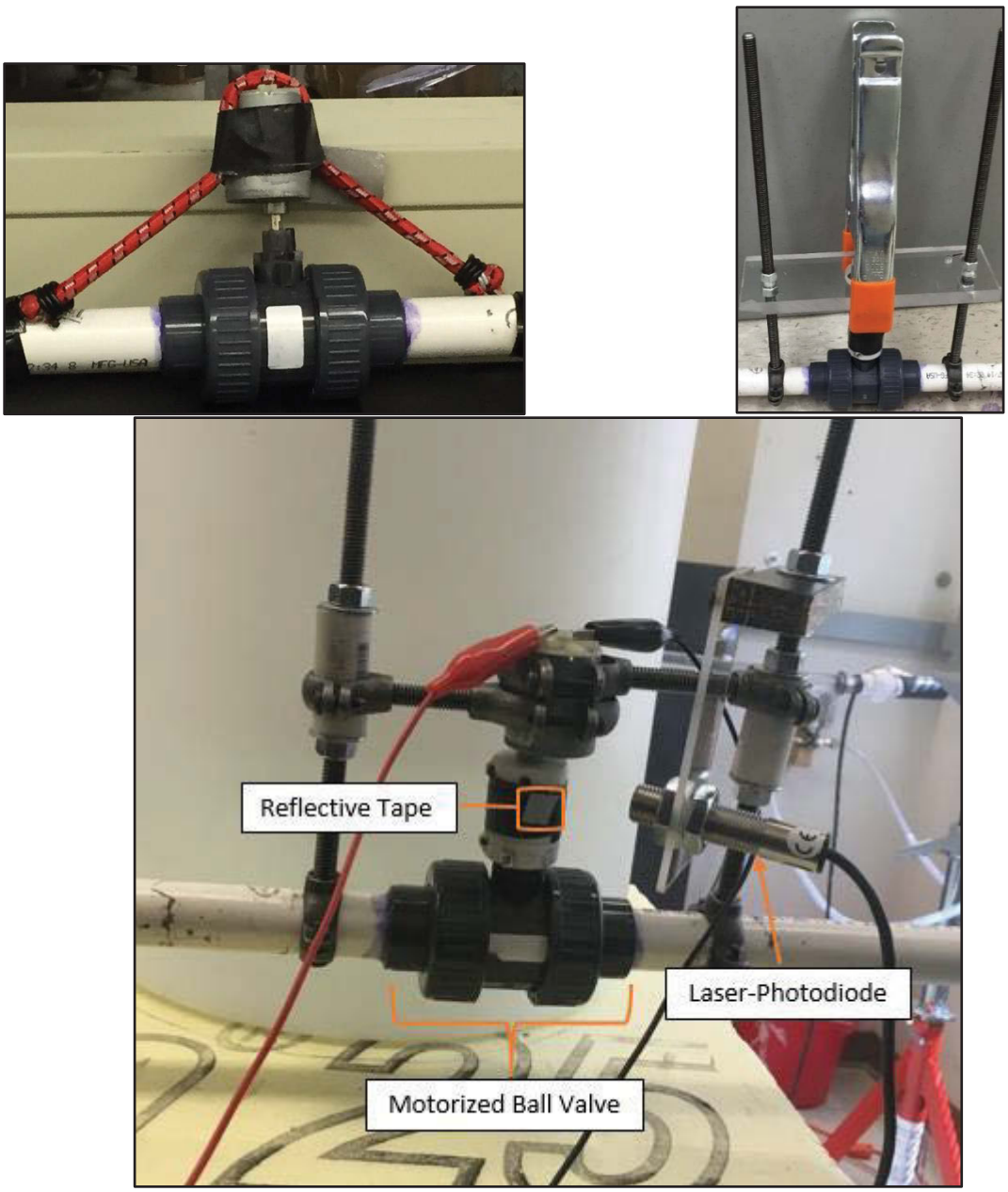


Figure 10. Specially designed motorized ball valves. First generation motorized ball valve (top-left), second generation motorized ball valve (top-right) and third generation motorized ball valve (bottom)

Figure 11. AC to DC power convertor (HP 6253A)

instrumentation suit utilized herein. Figure 12 A shows the producer loop and data supply loop. The producer loop acquires the signal at 3 kHz and stores the excess data into the queue. The data supply loop releases the excess data points which could not be processed by the consumer loop due to slower processing rate. The consumer loop seen in Figure 12 B is utilized to process, store and display the data (seen in Figure 13). A loop seen in Figure 12 C is used to store and convert binary files into text files.

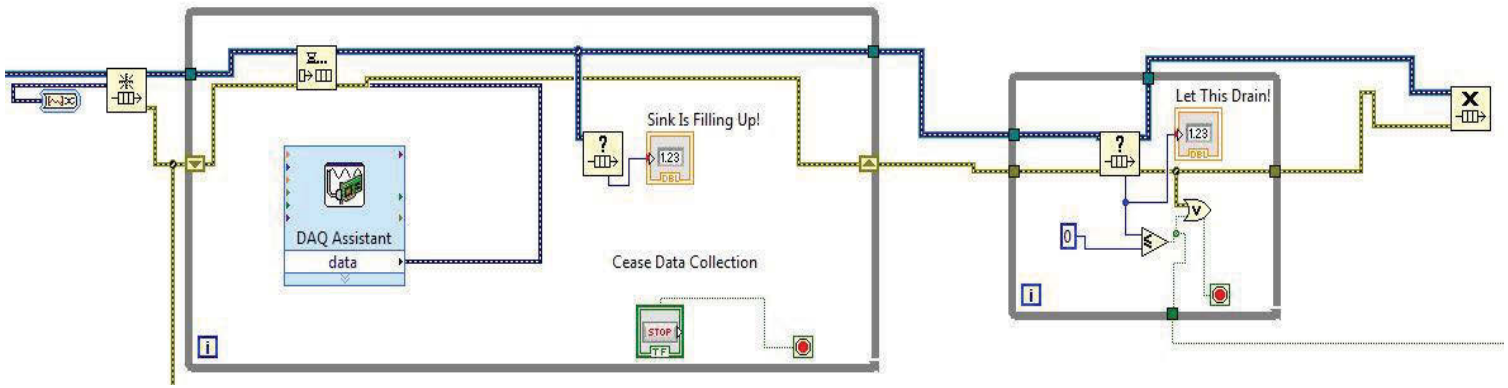
Mastersizer 3000

Mastersizer 3000 is a particle size analyzer used to measure the size distribution of the Amberlyst A26 OH form anion exchange resin beads. A laser diffraction technique is utilized to measure particle sizes ranging between 0.01 µm to 3500 µm. For this measurement, the wet dispersion unit (Hydro LV) is used to suspend the particles for measurement. Figure 14 shows measurement setup with both the Mastersizer 3000 (Top) and Hydro LV (Bottom) attachment. Particles are introduced into the wet dispersion unit and pumped into the Mastersizer 3000 with deionized water at an angular frequency of 3000 rpm. Inside the Mastersizer, two lasers, red and blue with the wavelength of 633 nm and 470 nm, respectively, are targeted towards the measurement cell via precision optics. As the sample circulates through the measuring cell, the particles obscure and scatter the laser light at various angles. These features are captured by the side and back scattering detectors for the 470 nm laser and additional focal plane detector for the red laser. Depending on the particle type, either Mie theory (transparent material) or Fraunhofer (opaque material) scattering technique is used to assess the particle size. For this particular measurement, the fraunhofer scattering is used to measure the distribution of the particles. A sample is measured 10 times in a 12-second duration followed by 48-second delays between the measurements. As a result, the measured distribution for middle 80th percentile, the particle size ranged from 596-789 µm, which slightly differs from the manufacturer quoted range of 560-700 µm. One of the reasons for the difference may be particle swelling. The particle size range quoted by the manufacturer is likely for the dry particles, but once they are wet, they tend to swell and increase in size. Figure 15 below shows the measured particle distribution on a linear scale.

Particle Activation Protocol

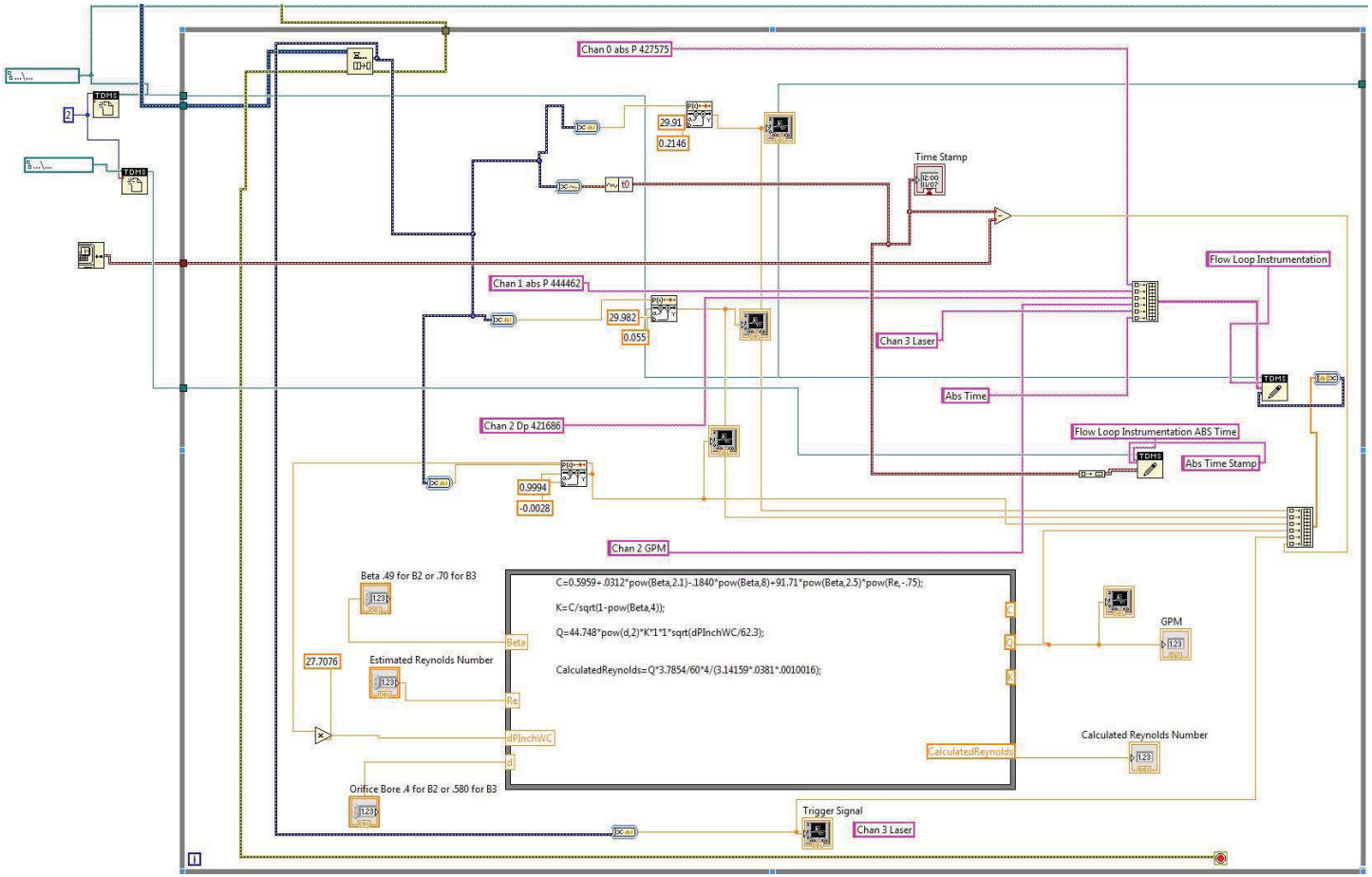
For the PEPT experiment, anion exchange resin beads manufactured by Sigma-Aldrich, shown in Figure 16, are labeled with a^{18}F. The desired number of resin beads are weighted and counted prior to activation, based on past measurement of 46 ± 7 particles giving 8 ± 0.5 µg. Siemens PET-Net center located at the University of Tennessee Medical Center is used to produce 100 µL of ^{18}F ion solution with a specific activity of 1.5 Ci/mL. In order to

Figure 12. Source code utilized for processing and storing instrumentation signals A). Producer Loop B). Consumer Loop C). Data Storage



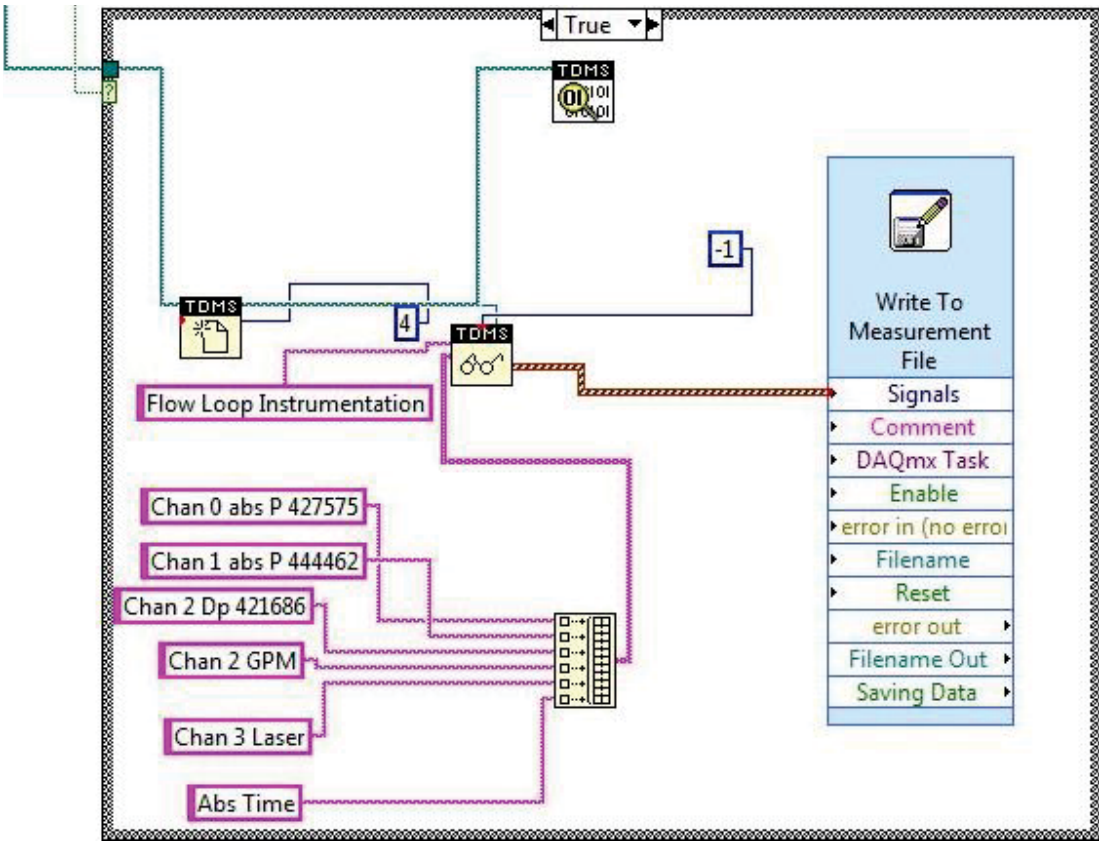
(A)

Figure 12. Continued



(B)

Figure 12. Continued



(C)

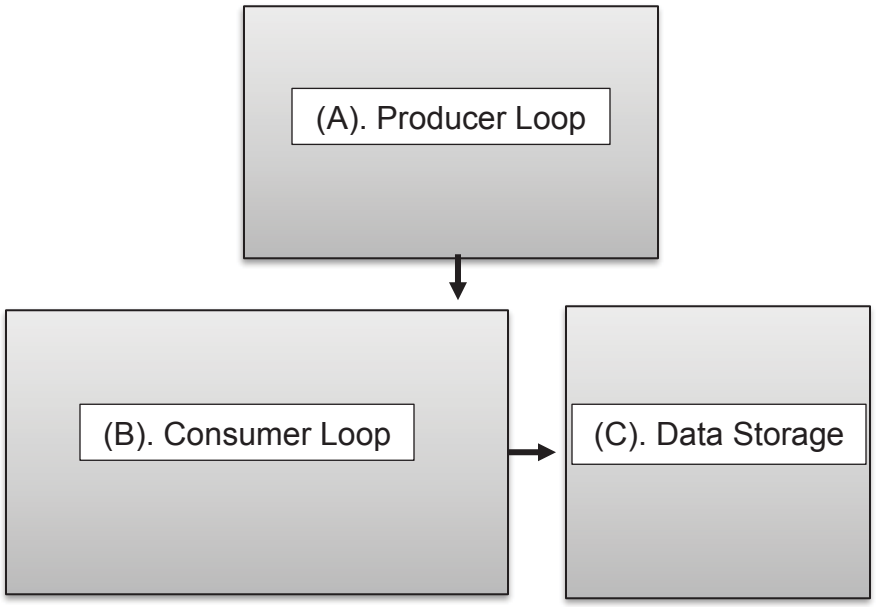


Figure 12. Continued

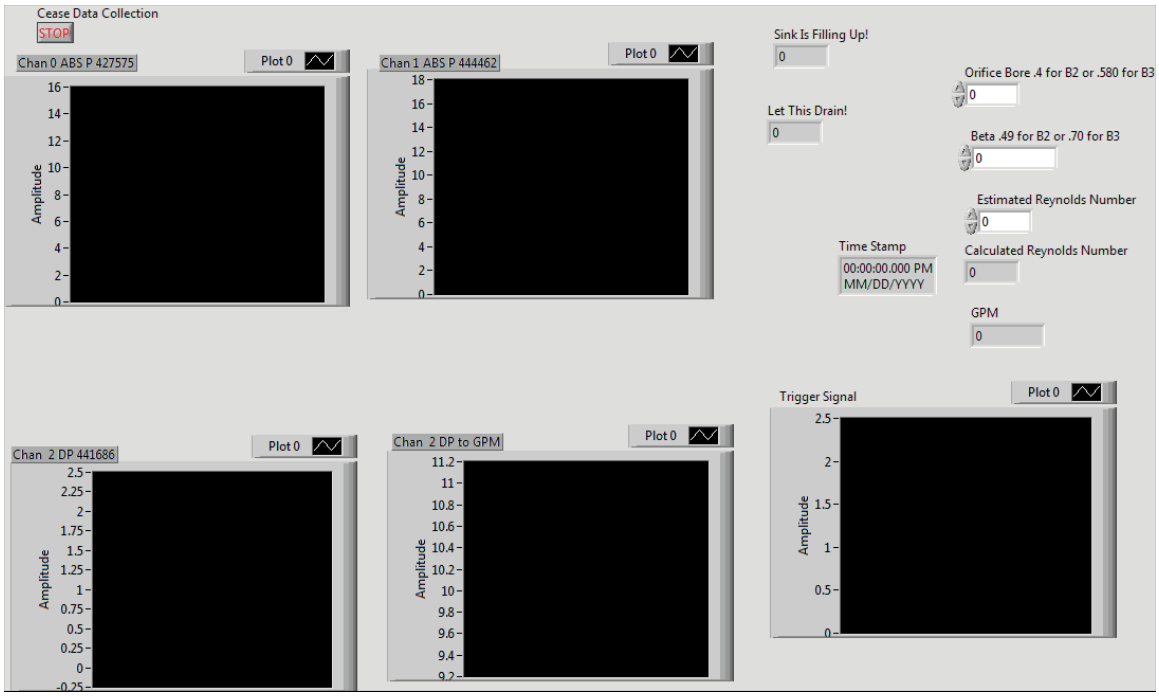


Figure 13. Operator view of the LabVIEW code

Figure 14. Visual representation of the Mastersizer 3000 and Hydro LV

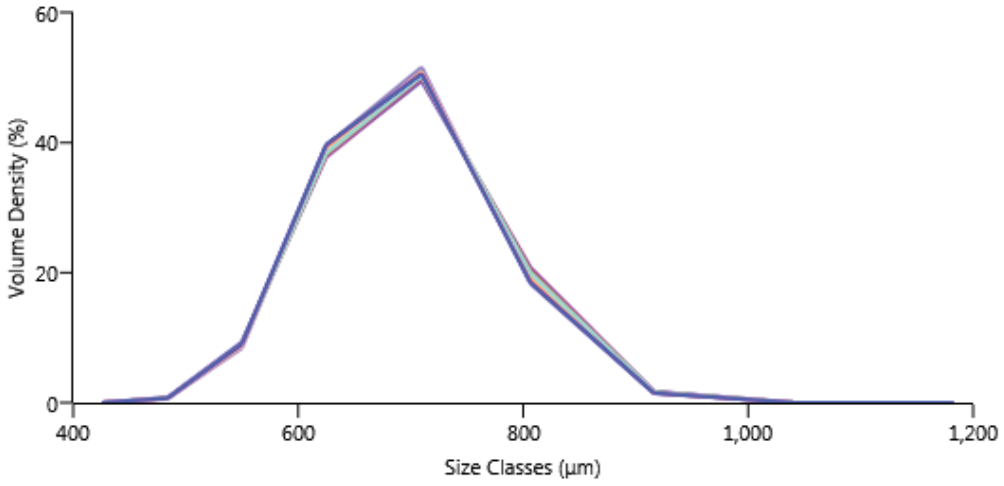


Figure 15. Measured particle size distribution of the anion exchange resin beads

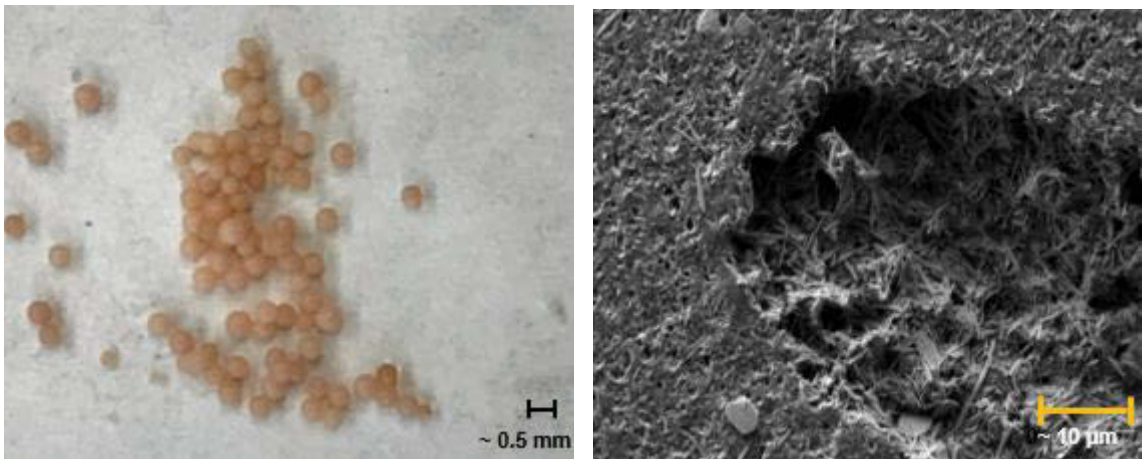


Figure 16. Amberlyst A26 anion exchange resins. Physical appearance of the anion exchange resin beads (left). SEM of the recirculated surface (right)

achieve this, ^{18}O enriched water is bombarded with 11 MeV proton beam produced from the Siemens cyclotron. Once the process is complete, the solution is transported in a syringe in a shielded shipping cask to the activation lab and the particle activation protocol is started. A centrifuge vial with a desired number of particles is placed in a syringe shield behind the shielding block. The syringe filled with ^{18}F is delivered from the cyclotron and added to the centrifuge vial. The wetted resin beads are left to soak for 20 minutes with manual agitation every 5 minutes. Next, the particles are rinsed 3 times with deionized water before placement in the counting well for the activity measurement. Upon completion, the particles are transported to the laboratory with the flow loop and Inveon PET scanner in a lead shielded box via radiation safety personnel. Before placement in the flow loop, the particles are mixed with a few drops of glycerin and deionized water.

Conducting an Experiment

Several preparatory protocols are implemented to ensure the experiment is conducted smoothly and safely. Before each measurement, an experimental procedure along with a data sheet are written. A day before the experiment, all participants conduct a rehearsal to ensure everyone is aware of their duties. The flow loop is assembled and rinsed with deionized water to assure no impurities are present. The flow loop is filled with deionized water at room temperature and circulated around the loop by a little giant pump (4-MDQ-SC). The loop is inspected for leaks and instrumentation is tested. The written procedure is followed and necessary changes are implemented. At the end of the process, necessary equipment for the particle activation is gathered. In addition, the desired number of particles are weighed and counted.

On the day of the experiment, two of the participants travel to Siemens PET-Net center for the particle activation procedure. Meanwhile, a few participants' stay at the experimental laboratory to verify no leaks or air bubbles are present in the flow loop. The instrumentation is also checked once again for proper operation.

Once the particles arrive at the experimental facility, they are mixed with a few drops of glycerin and deionized water. The pump is turned off and particles are injected by gravity through the particle injection line. Once particles are at the bottom of the line, the pump is actuated. The particles circulate throughout the loop until the end of the experiment.

Pulsatile and Steady Flow Experiment

Three series of experiments are conducted using the flow loop seen in Figure 2. The initial experiment is conducted to measure fluid behavior through pinched cross-section. The test section for the experiment is pinched with a spring clamp (manufactured by Wolfcraft), creating a desired pinched cross-section. During this measurement, a few flaws were discovered in the flow loop. The absolute pressure transducers located on each end of the test section were mounted upright above tee adapters. The mounting position resulted in an air column between the pressure transducer sensor and water. This led to inaccurate pressure measurement and dynamic response of the compressible air column. A new mount with no additional adapter was installed and positioned horizontally to eliminate the air volume. In addition, the flow loop was water solid during the initial experiment, including the upright particle injection line. The pressure waves created with each valve closure were not damped between cycles causing chaotic behavior in the pressure signal. In order to resolve this issue, a 9 cm air volume was kept in the particle injection line to increase the compressibility downstream of the test section. Finally, the motorized ball valve showed inconsistency in the period throughout the duration of the experiment. The rotary valve design was improved (version 2 to version 3 per Figure 10) to improve consistency in pulsatile frequency during the experiment.

A few months after the initial experiment, another experiment, with reconfigured flow loop per above discussion is conducted. This experiment is performed to measure the steady flow and pulsatile flow in a circular cross-section tube. Table 2 below shows the experimental conditions for all three experiments. Note that the uncertainty in the activity per radiotracer is based on the uncertainty in the number of radiotracers. There might be additional variation associated with the activity per radiotracer based on the surface area of the individual radiotracer.

Table 2. Experimental conditions and measurement acquisition rate

Parameters	Initial Pulse Flow	Steady Flow	Pulse Flow
Cross-Section	Pinched	Circular	Circular
Measurement Time (minutes)	50	2	65
Selected Energy Window (keV)	425-625	425-625	425-625
Average Flow Rate in a Steady Condition (m/s)	1.1	1	1
Pulse Frequency (Hz)	2.15±0.01	N/A	2.07±0.01
Reynolds Number	20945	18850	18850
Number of Radiotracers	127±20	155±25	155±25
Radioactivity per Radiotracer (μCi/tracer)	190±35.5	170±32.7	170±32.7
CL Timing (Hz)	5000	5000	5000
Instrumentation Acquisition Rate (Hz)	3000	3000	3000

CHAPTER FOUR

DATA ANALYSIS SOFTWARE

Multiple data analysis software packages, written in house, are used to analyze the measured data. The data collected by the scanner are stored in a list mode file. The list mode files encompass information on coincident events, detector number, time stamps (every 200 μs), time marker from the photodiode and some other information such as scanner bed position. Multi-PEPT algorithm is utilized to read in the information from the list mode files, extract CL orientation and timing data, and to use the CL and timing data to detect and track individual resin beads throughout the field of view of the scanner. Particle position information of the individual trajectory is extracted to assess various flow features, such as velocity, acceleration, diffusion and turbulent kinetic energy. Furthermore, the pressure wave velocity of the flow is calculated based on the inlet and outlet pressure data.

Multi-PEPT Algorithm

The multi-PEPT algorithm used herein is based on the line density method of Bickell *et al.* [7] and feature point identification method of Crocker and Grier [16], and Sbalzarini and Koumoutskos [17]. Figure 17 below shows a graphical representation of the multi-PEPT algorithm. A detailed description of the multi-PEPT algorithm is presented by Wiggins *et al.* [8]-[9]. First, a positron emitting trace particle travels from the outside to inside of the FOV of the scanner (Figure 17 A and B). The emitted gamma rays are detected by the PET scanner and linked into CLs at a prescribed time step. The measured CL data from the entire scan is arranged into 1 ms time steps. A virtual grid of 1 mm x 1mm x 1 mm volumes, referred to as voxels, is laid across the field of view of the scanner (Figure 17 C). As the particle travels through the scanner, the collected CLs are tallied based on the number of line crossings in each voxel during each time step (Figure 17 D). These single scalar line crossing tallies in each voxel are smoothed by convolution method with a cubic boxcar kernel of volume 27 mm^3. The filtered image is used to find the location with maximum line crossings at a given time step. A user defined radius around the maximum line crossing location is considered to locate a grid-weighted centroid to determine the particle location (Figure 17 E and F). All the particle positions are located in each 1 ms time step for the duration of the scan. Next, the particle positions in each frame are linked based on the 2-frame nearest neighbors approach. This method is adapted from the four-frame nearest neighbor approach described by Hassan and Canaan [18]. In the 2-frame nearest neighbor approach, the previous particle position, velocity, and acceleration information is used to predict the next position in future frames. Based on this, a user defined search radius is explored

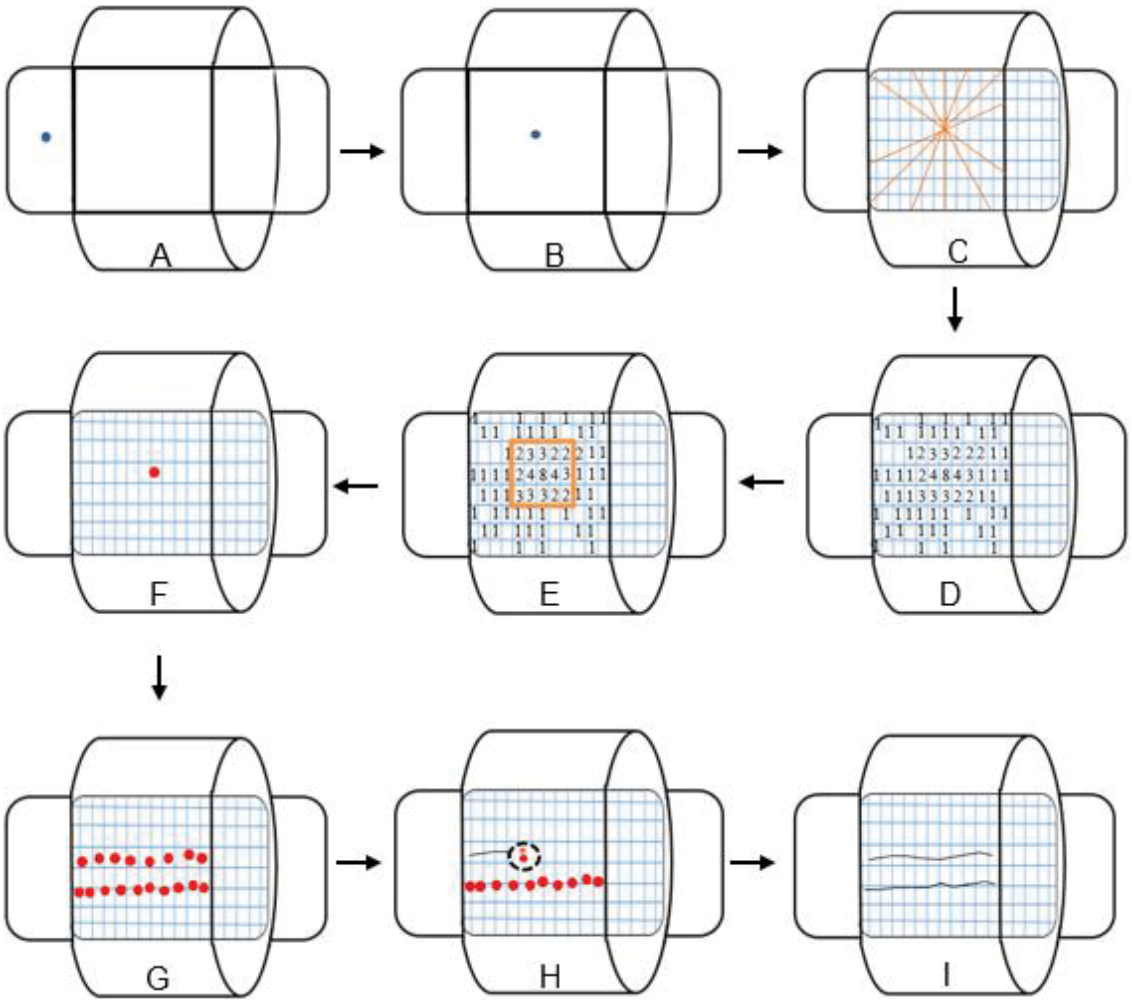


Figure 17. Visual illustration of the multi-PEPT algorithm. Particle annihilation (top row), particle detection and location (middle row) and particle trajectories (bottom)

for the next particle position (Figure 17 H). The particle with the shortest distance to the predicted position is linked into trajectory (Figure 17 I). This processed is based on Kuhn-Munkres method [19]–[21].

Particle Dispersion Analysis

The particle dispersion analysis is conducted to observe the particle behavior from one location to another downstream of the test section. Filtered particle trajectories acquired from the multi-PEPT algorithm are plotted in 3-D using MATLAB 2016. Multiple parameters, such as the region of interest, the minimum length of the trajectory and axial location of analysis are user defined. The trajectory behavior from the initial point to end is analyzed and index of dispersion can be calculated using equation 7. Appendix A contains a MATLAB code used to implement this method and Figure 18 shows a visual representation of this method.

$$D = \frac{\sigma^2}{\mu} \tag{7}$$

Where:

σ: Standard deviation
μ: Mean

Turbulent Kinetic Energy Calculation

MATLAB 2016 is used to calculate the turbulent kinetic energy across multiple radial slices of the tube. Filtered particle trajectories with a particle position every 1 ms are extracted from the multi-PEPT algorithm. Individual trajectories with more than 4 data points are considered for the analysis. First, the particle trajectories are transformed to align the test section with the z axis. This is computed by subtracting individual data points with the average of the sample maximum and minimum in x and y coordinate directions as seen in equation 8 and 9.

$$Mean\ (x,y) = \frac{maximum\ (x,y) + minimum(x,y)}{2} \tag{8}$$

$$\begin{aligned} x' &= x_i - Mean\ (x) \\ y' &= y_i - Mean\ (y) \end{aligned} \tag{9}$$

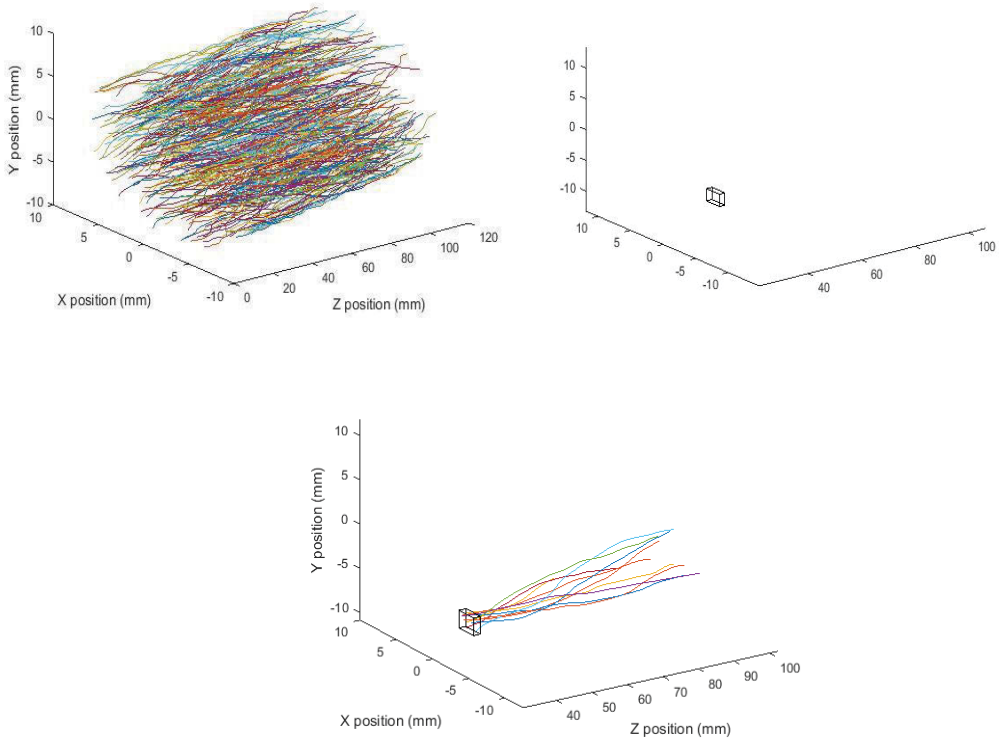


Figure 18. Visual representation of the diffusion analysis. Plot of particle trajectories (top-left), region of interest (top-right) and particle dispersion (bottom)

Where,

x': Adjusted position in x direction
y': Adjusted position in y direction
x_i: X-position at a given time
y_i: Y-position at a given time

The adjusted x and y position along with the z position and time information of an individual trajectory is used to calculate the radius, angle, and velocity (in each coordinate direction). Equation 10-12 below shows the method used to assess each parameter. The velocities represented herein are assessed by backwards divided difference and transformed from Cartesian to cylindrical coordinate system following equations 13-15 below.

$$r_i = \sqrt{x'^2 + y'^2} \tag{10}$$

$$\theta_i = \tan^{-1}\left(\frac{y'}{x'}\right) \tag{11}$$

$$V_{u,v,w} = \frac{P_{x'_{i+1},y'_{i+1},z'_{i+1}} - P_{x'_i,y'_i,z'_i}}{t_{i+1} - t_i} \tag{12}$$

$$V_{r_i} = u_i * cos(\theta_i) + v_i * sin(\theta_i) \tag{13}$$

$$V_{\theta_i} = -u_i * sin(\theta_i) + v_i * cos(\theta_i) \tag{14}$$

$$V_{z_i} = w_i \tag{15}$$

The mean velocity of an individual trajectory is calculated in each coordinate direction based on the equation 16.

$$\bar{V}_{r,\theta,z} = \frac{\sum_{i=1}^{i} V_{r_i,\theta_i,z_i}}{i} \tag{16}$$

Turbulent kinetic energy is based on the velocity deviation from the mean in each coordinate direction as shown in equation 4. Hence, the mean of an individual trajectory is subtracted from the velocity at each instance. Afterwards, the velocity fluctuations in each coordinate direction are binned into a radial slice and a turbulent kinetic energy value is assigned to each bin. Figure 19 below shows

a visual representation of the five concentric slices assessed for the steady flow. Implementation of this method is seen in Appendix A.

Phase Angle and Pressure Wave Velocity

The phase angle and the pressure wave velocity is assessed for the pulsatile flow based on the measured absolute pressure across the test section. First the absolute pressure is smoothed by averaging the data (collected at 3 kHZ) in 50 data point increments. The averaged data is normalized by subtracting each point with minimum average pressure and dividing the corrected pressure by maximum average pressure. The average outlet pressure is slightly shifted up to match the inlet pressure. Normalized inlet and outlet pressures are seen in Figure 20. The time differential (dt) between two signals is calculated over the normalized pressure range of 0.3 to 0.9 in 0.05 increments and phase angle determined by equation 17. Likewise, the pressure wave velocity is assessed by using equation 18. The pulsatile flow in pinched cross-section and a circular cross section is measured at the set frequency of 2.15 Hz and 2.07 Hz respectively, in a 1.16 m long test section. Appendix D shows the MATLAB code used for this assessment.

$$\theta = 360 * f * dt \tag{17}$$

$$Pressure\ Wave\ Velocity = \frac{Length\ of\ the\ Test\ Section}{dt} \tag{18}$$

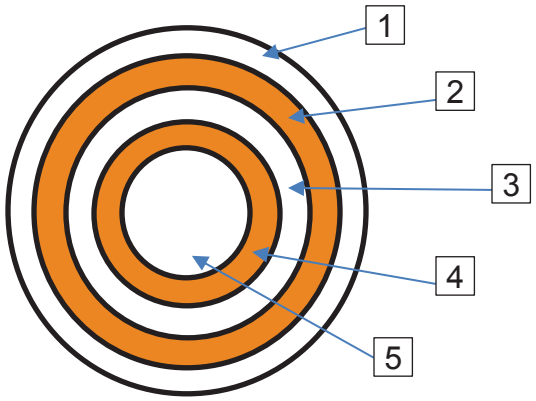


Figure 19. Five equally divided regions of the pipe assed to analyze the average TKE per region

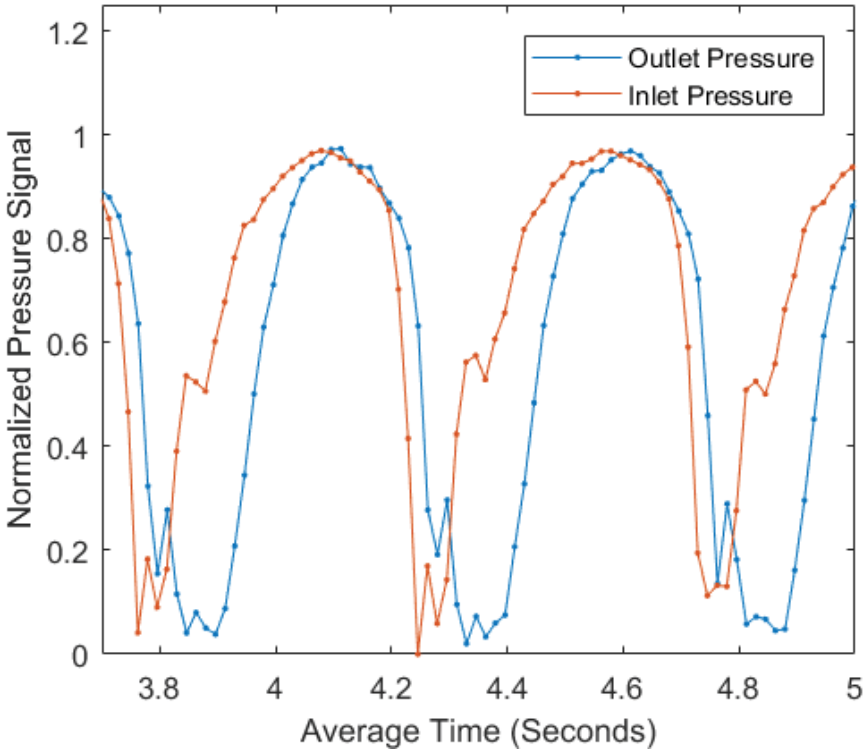


Figure 20. Normalized pressure signal

CHAPTER FIVE
RESULTS AND DISCUSSION

The in-house data analysis software described in previous section is used to analyze the flow features in a steady and pulsatile flow. First, the multi-PEPT algorithm is used to detect and track the particles throughout the field of view of the scanner in 1 ms time increment using 1 mm^3 grid size. As a result, 4828, 422 and 7275 particle trajectories are detected in pulsatile flow (pinched cross-section), steady flow and pulsatile flow (circular cross-section) respectively. The particle dispersion analysis software is a proof of concept tool created to measure particle dispersion downstream of a selected volume in a steady flow. In this work, it is used to show some preliminary results. The turbulent kinetic energy calculation is performed to understand the turbulence characteristics of the fluid flow. Herein, the turbulent kinetic energy is calculated as a function of radial position for steady flow and as a function of time and radial position for the pulsatile flow. Note that the majority of the data presented here are in the Lagrangian frame, unless otherwise specified, therefore individual particles are followed to collect statistics on turbulence.

Steady Flow

The steady flow is measured at 1 m/s in a circular tube of 19 mm diameter. The individual particle trajectories are measured and velocity at each instance is calculated using two methods: convolution with differentiated Gaussian kernels [9] and backward divided difference. Figure 21 shows the measured velocity field in the z- coordinate direction. The velocity is peaking near 1.4 m/s. The particle velocities and radius values are averaged in 20 equally spaced radial slices to generate the velocity profile shown in Figure 22. Figure 22 also shows the measured velocity profile compared an empirically calculated profile based on the $1/7^{th}$ power law.

Individual particle positions acquired from the multi-PEPT algorithm are plotted and a region with heavy particle population is chosen for the diffusivity analysis. The analysis is conducted at coordinate (-1.5,-0.5, 40) with 1 mm grid size in each dimension. A filter is applied to allow the user to only consider the trajectories with a minimum length of 50 mm. The diffusivity of the particles is analyzed from the initial coordinate direction to 90 mm in the z direction. Figure 23 shows the visual representation of the particles passing through a cubic volume. The index of dispersion could not be calculated for this data set due to the low number of particle trajectories (i.e. 8), however, Table 3 shows the dispersion characteristics of the particles downstream of the test section.

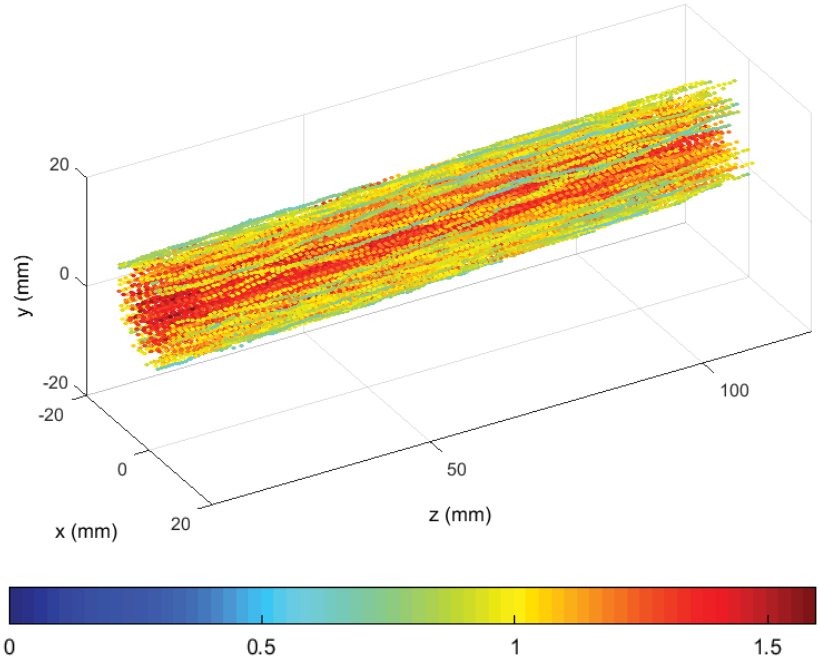


Figure 21. Axial velocity field across the test section in a steady flow

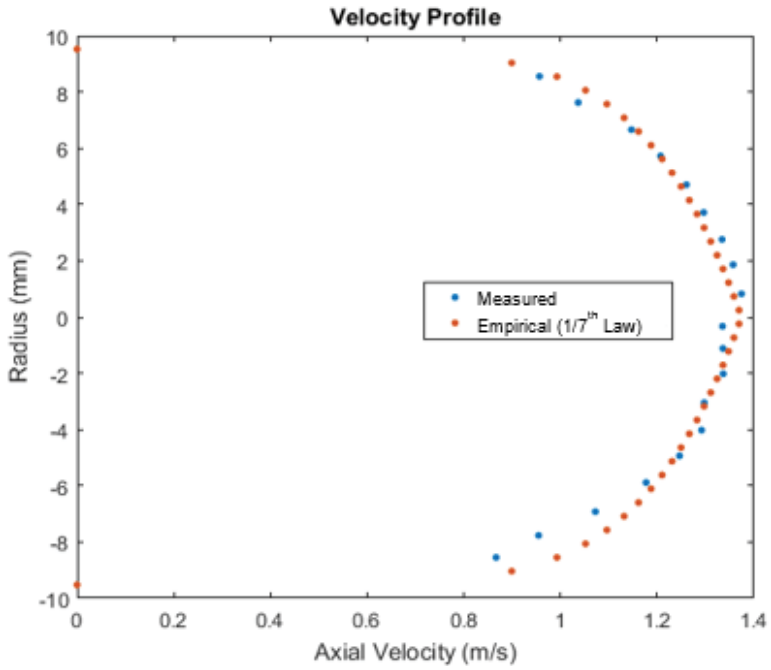


Figure 22. Measured and analytical velocity profile in a steady flow (Re~20000)

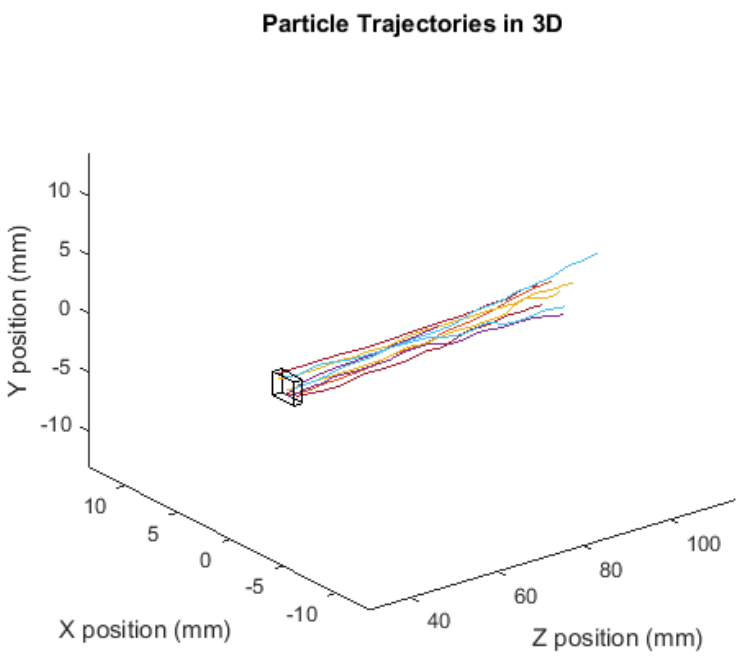


Figure 23. Particle trajectories passing through 6 mm^3 volume

Table 3. Diffusivity in x and y coordinate position

Trajectory	$X_{initial}$ Position (mm)	X_{final} Position (mm)	$Y_{initial}$ Position (mm)	Y_{final} Position (mm)
1	-1.7959	0.6423	-1.1210	-2.2952
2	-0.6562	-0.4952	0.3128	0.6645
3	-1.6381	-1.8972	-0.6075	0.4549
4	-0.5802	0.4274	-0.2530	-0.5294
5	-2.4418	-1.6598	-0.8266	0.0937
6	-1.7670	-1.4693	-0.5392	1.3677
7	-1.5090	1.3241	-1.2368	-1.6067
8	-1.2170	-2.1703	0.1914	-0.1999

Figure 24 shows the turbulent kinetic energy as a function of radius. The turbulent kinetic energy seems to be highest on the edge of the pipe and lowest towards the center of the pipe. In addition, a discontinuous region at the center of the pipe is observed. This might be due to either lack of trajectories in that region or initial test section alignment with major axis for data assessment used during the TKE calculation. Figure 25 shows the fluctuating components of velocity in each coordinate direction.

Pulsatile Flow (pinched cross-section tube)

The pulsatile flow through pinched cross-section tube is created with the second-generation motorized ball valve. The flow characteristics are measured at flow rate 1.1 m/s with the pulse frequency of 2.15 Hz. The individual particle trajectories collected during the length of the scan are synchronized into individual pulse cycles based on the time marks from the laser photodiode. The individual pulse cycle is time gated into 20 equally spaced frames to understand flow attributes at different instances of the pulse cycle. Figure 26 shows 8 of 20 frames, depicting the velocity field in the z direction at various parts of the pulse cycle. The velocity accelerates through two-thirds of the cycle then quickly decelerates and recovers towards the last one-thirds of the cycle. In addition, the velocity of the particles moving around the pinched region is 2-4 times higher than the circular region upstream of the pinched region. A region towards the end of the pinched cross-section exhibits flow recirculation apparent as blue color particle velocity trajectories in Figure 26. The velocity profile, diffusivity and turbulent kinetic energy analyses are not conducted due to complex geometry and time dependent nature of the flow.

Pulsatile Flow (circular cross-section tube)

The pulsatile flow in the circular tube is generated using the third-generation motorized ball valve with improved consistency in pulse frequency. The flow features are measured at a flow rate of 1 m/s with the pulse frequency of 2.07 Hz. The individual particle trajectories are synchronized and divided into 20 equally spaced time frames. Figure 27 represents 8 of 20 frames, illustrating the velocity field in the z direction at various instances of the pulse cycle. Similar to the pulsatile flow in pinched cross-section tube, the flow accelerates, decelerates and recovers during the pulse cycle. A time dependent velocity profile of the pulsatile flow is generated in 20 frames to illustrate flow features throughout the pulse cycle. Appendix C shows the MATLAB code used for this assessment. The velocity profile is averaged at 20 radial locations and 3 of 20 frames are seen in Figure 28. The velocity profile at multiple instances is similar to the steady flow, however, there is a little dip in the profile towards the center of the pipe. This feature is caused by the tube elasticity and misalignment of the tube during the experiment. The diffusivity is not calculated for this flow due to

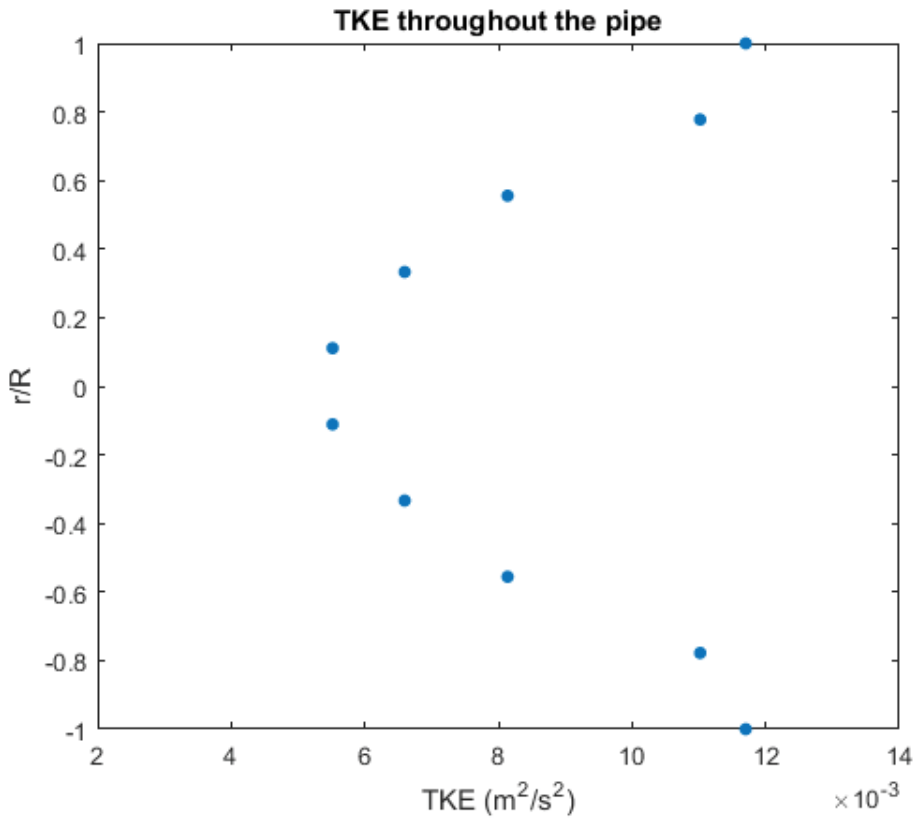


Figure 24. Average measured turbulent kinetic energy at 5 equally spaced concentric regions of the tube

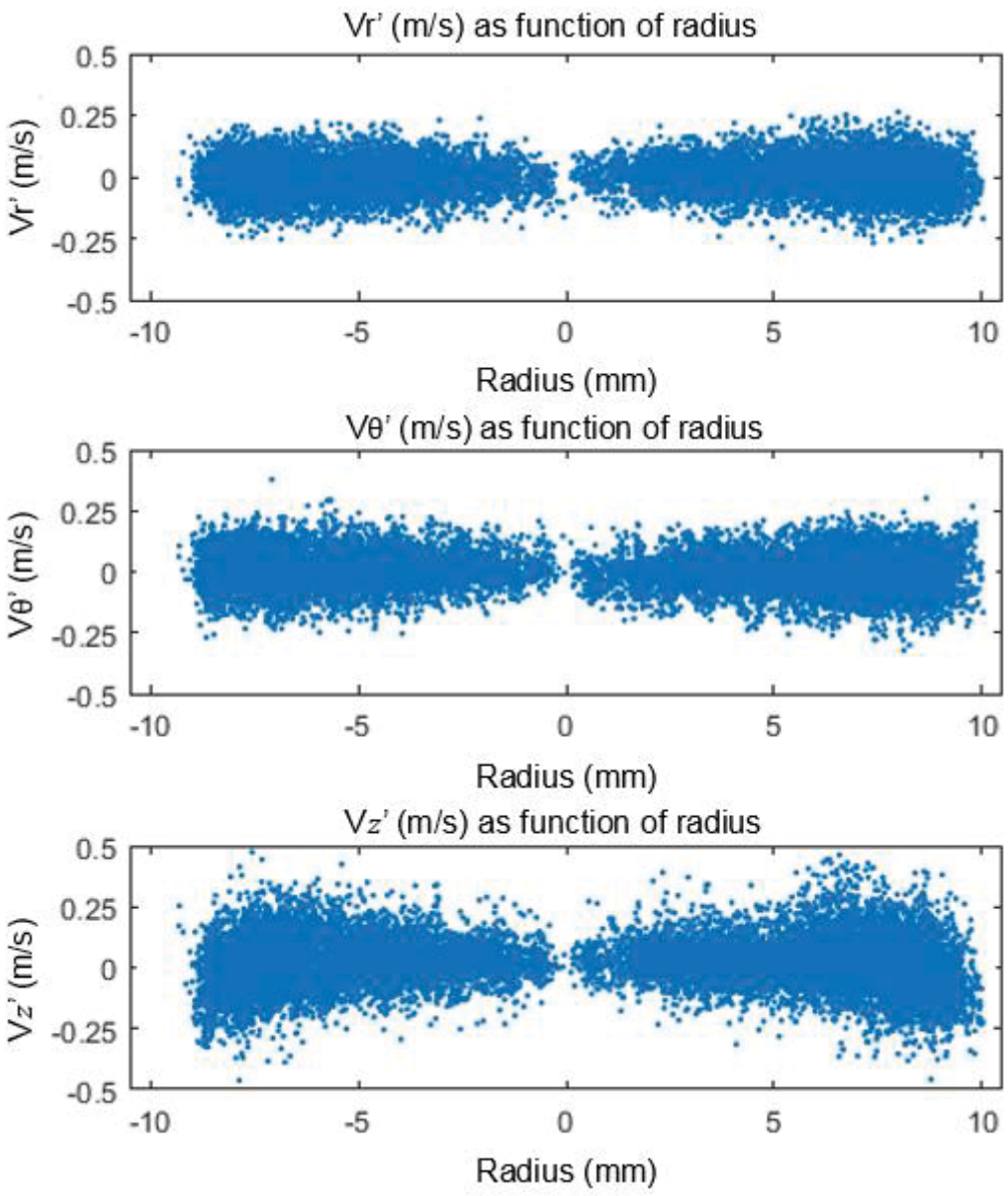


Figure 25. Point by point velocity fluctuation for an individual trajectory in each coordinate direction. V_r fluctuations (top), V_θ fluctuations (middle) and V_z fluctuations (bottom)

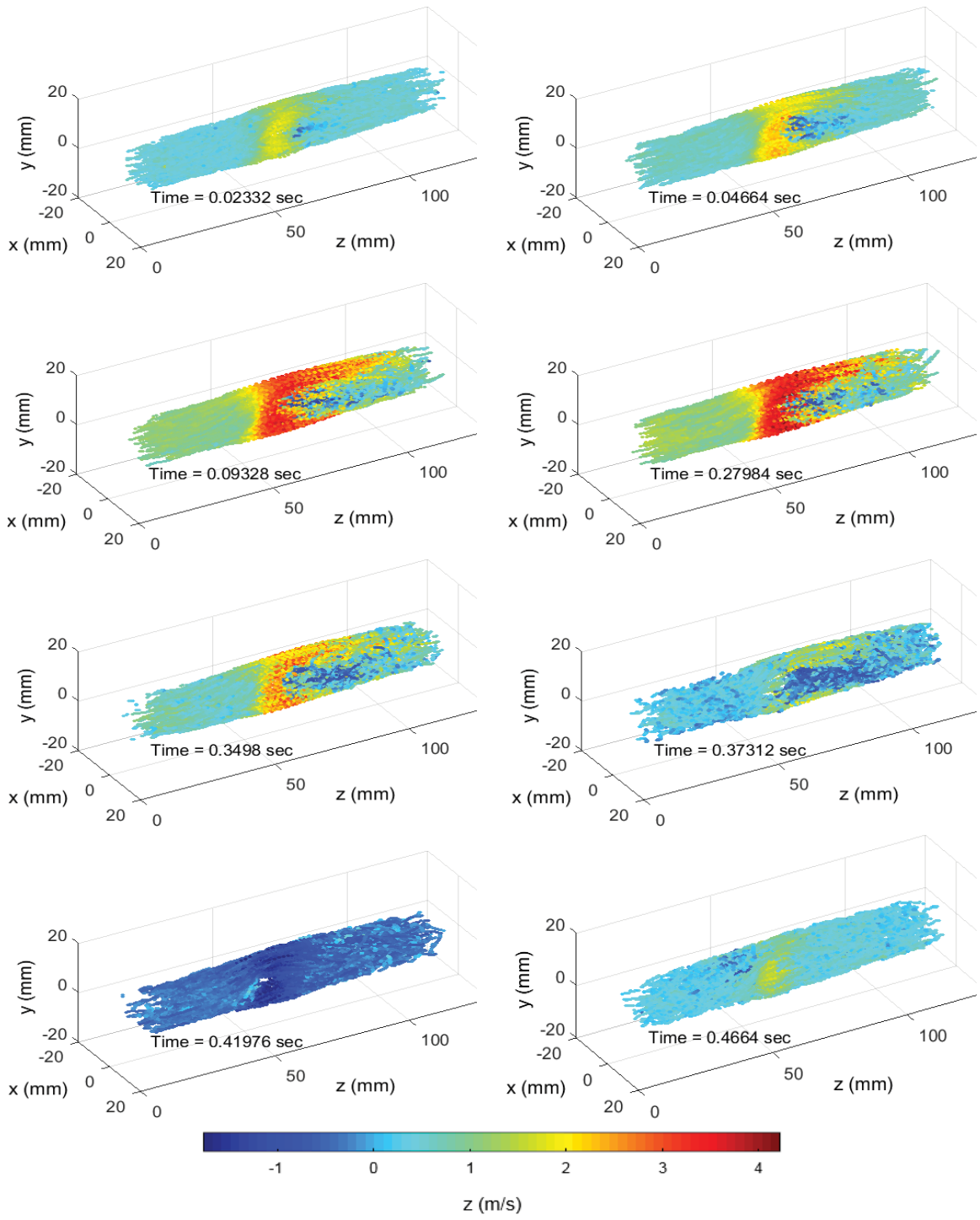


Figure 26. 8 of 20 pulsatile flow frames showing full flow, flow reversal and flow recovery for a tube with pinched cross-section

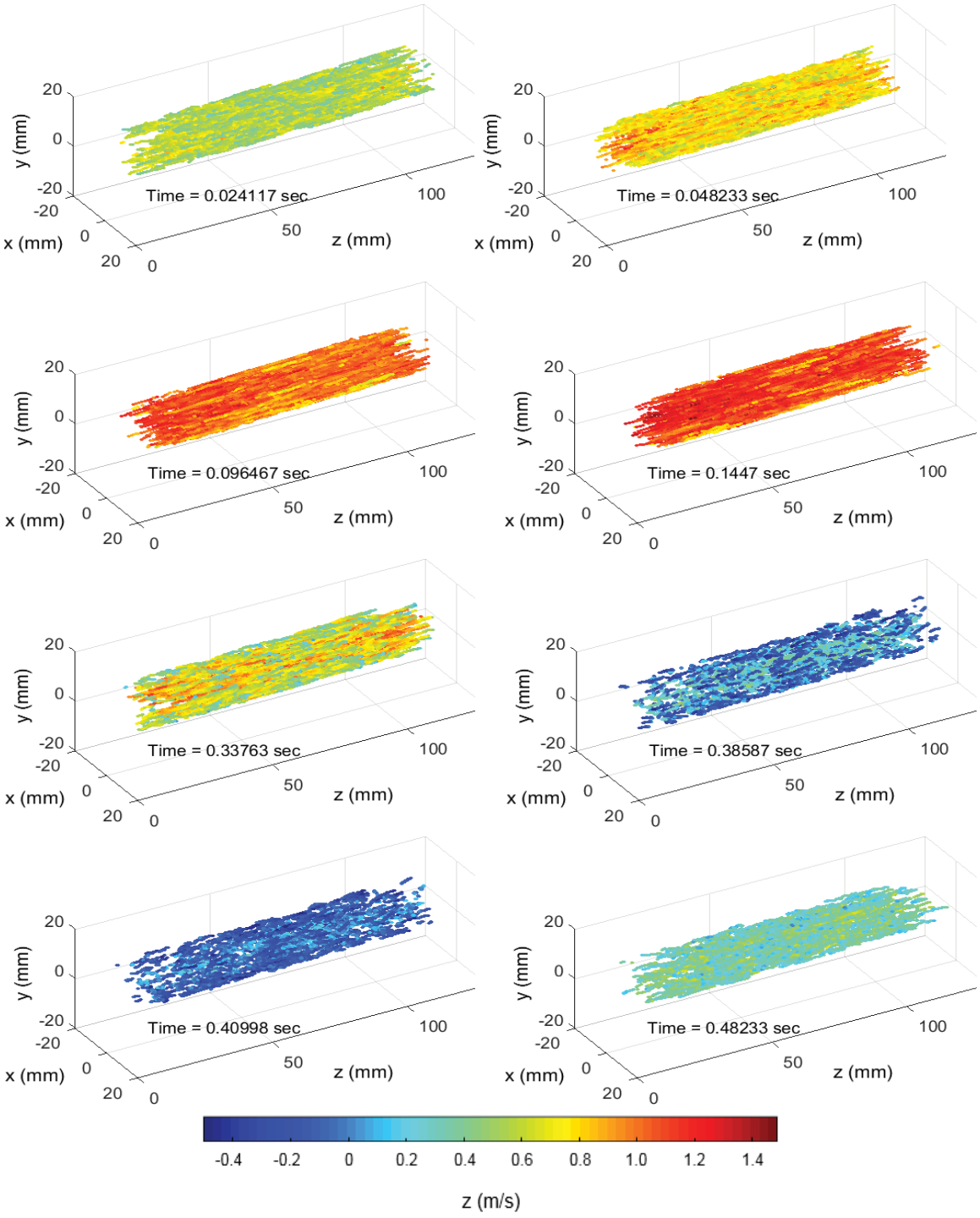
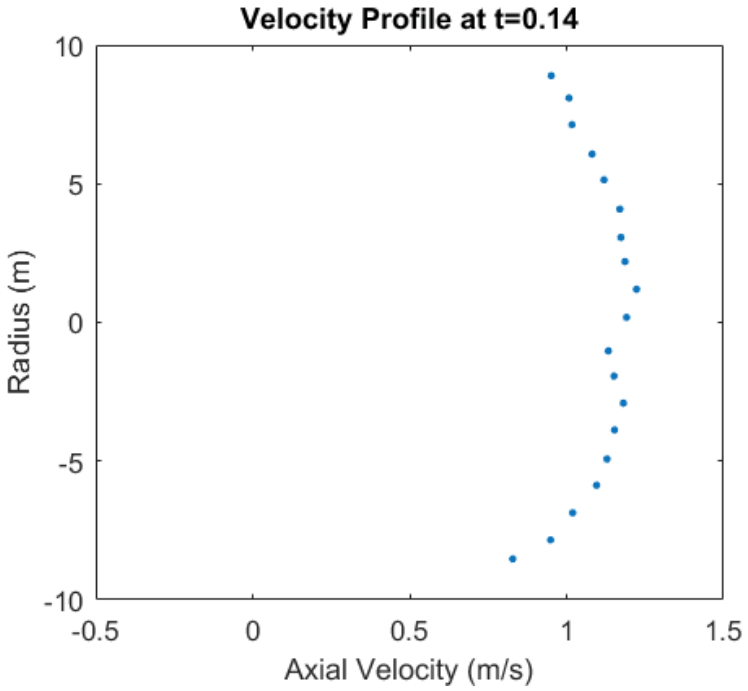
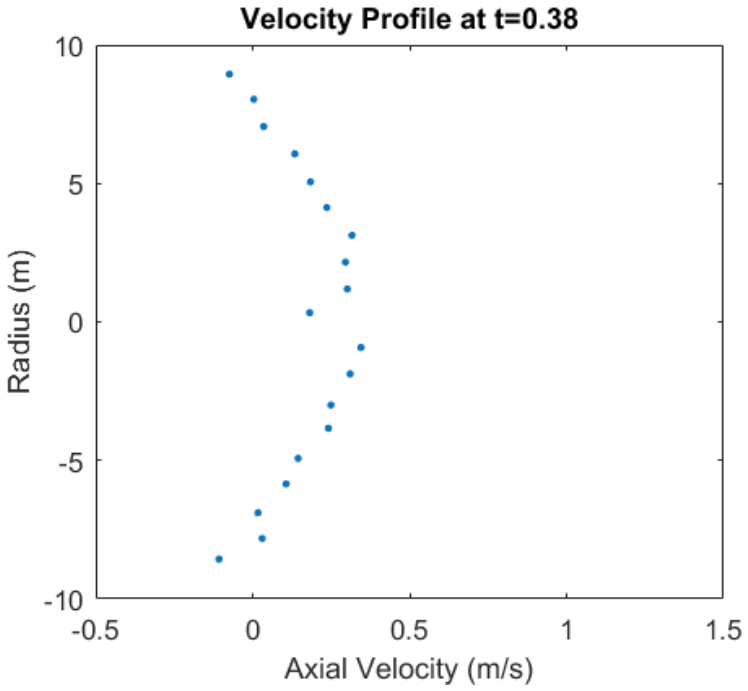


Figure 27. 8 of 20 pulsatile flow frames indicating full flow, flow reversal and flow recovery for a tube with circular cross-section

Figure 28. 3 of 20 pulsatile flow frames indicating velocity profile at full flow (A), flow reversal (B) and flow recovering (C) for a tube with circular cross-section

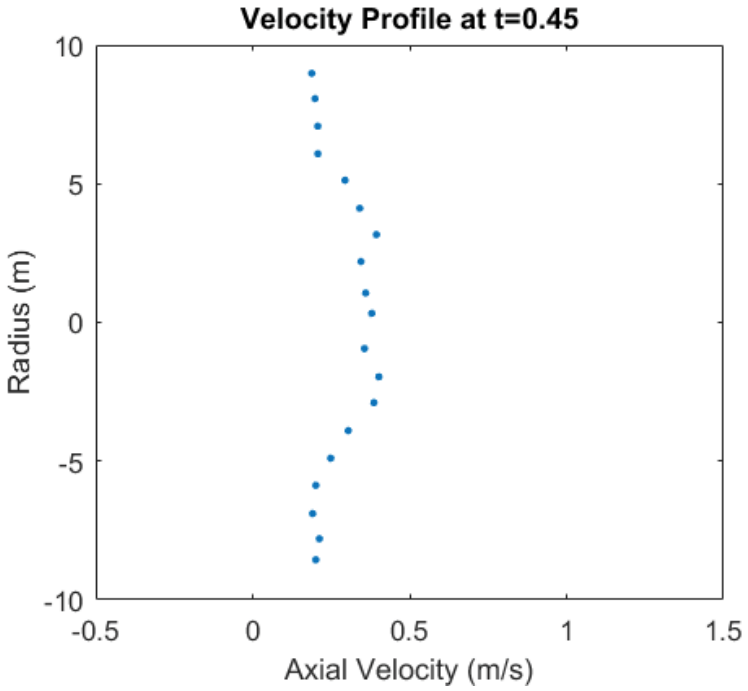


(A)



(B)

Figure 28. Continue



(C)

Figure 28. Continue

the flow reversal and time dependent nature of the flow. A time dependent turbulent kinetic energy is calculated, however, it is not presented herein due to uncertain features of the turbulent kinetic energy accumulated from the rapid velocity changes, tube boundary movement and overall measurement uncertainties. Appendix B shows the MATLAB code used to calculate the time dependent turbulent kinetic energy. The pressure wave velocity and phase angle of the pulsatile flow are also calculated. Figure 29 below shows the inlet and outlet absolute pressure signals. The outlet pressure lags behind the inlet pressure. Hence, the phase angle between the two is calculated based on the lag time. Figure 30 shows the phase angle at 4 regions of each pulse cycle, with a few pulse cycles shown. The phase angle between the inlet and the outlet pressure signal varies within the pulse cycle. The lag time is used to calculate the pressure wave velocity across the test section and results are seen in Figure 31. In both figures, a contribution from the elastic wall is apparent by irregular oscillatory patterns. In pulsatile flow, it is typical to overlay the pressure signal on top of the velocity signal to understand the phase lag between these parameters. Hence, Figure 32 shows the inlet and outlet pressure and individual particle velocity as a function of time. The particle velocities are almost in sync with the outlet pressure since the measurement is taken towards the outlet of the test section.

Uncertainty

The uncertainty in the PEPT measurement is calculated based on the standard deviation of the estimated location divided by the square root of CL. This is a slightly modified method of calculating the uncertainty than a method prescribed by Bickell et al [7]. The uncertainty for the line density method is calculated by using the full-width half maximum of estimated particle location instead of standard deviation. Figure 33 below shows the uncertainty in x, y and z coordinate direction for the steady flow experiment and pulsatile flow in pinched and circular cross-section tube. The average uncertainty in the steady flow is 0.23, 0.23 and 0.16 mm in x, y and z direction respectively. The average uncertainty in pulsatile is 0.31, 0.31, 0.20 mm pinched cross-section tube and 0.29, 0.30, 0.19 mm for circular cross-section tube in x, y and z directions respectively. The uncertainties are lowest near the center of the bore in all coordinate direction due to peak detection efficiency at that geometric location. On the other hand, the uncertainties are highest near the edge of the detector, since a smaller number of CLs are collected in that part of the FOV of the scanner. Recent discoveries by others in the group have improved understanding of position uncertainty in M-PEPT using FPI. The uncertainties reported here and in prior literature underestimate uncertainty in these measurements caused by scanner artifacts. These artifacts are detailed in a University of Tennessee Nuclear Engineering Dissertation in progress by Roque Santos.

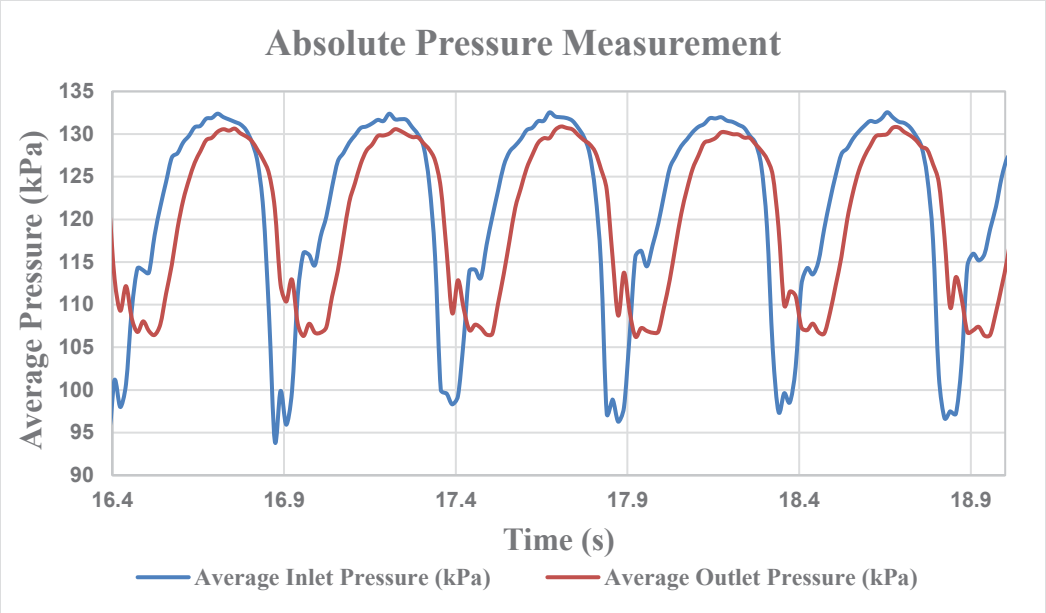


Figure 29. Absolute pressure measurement at the inlet and the outlet of the test section

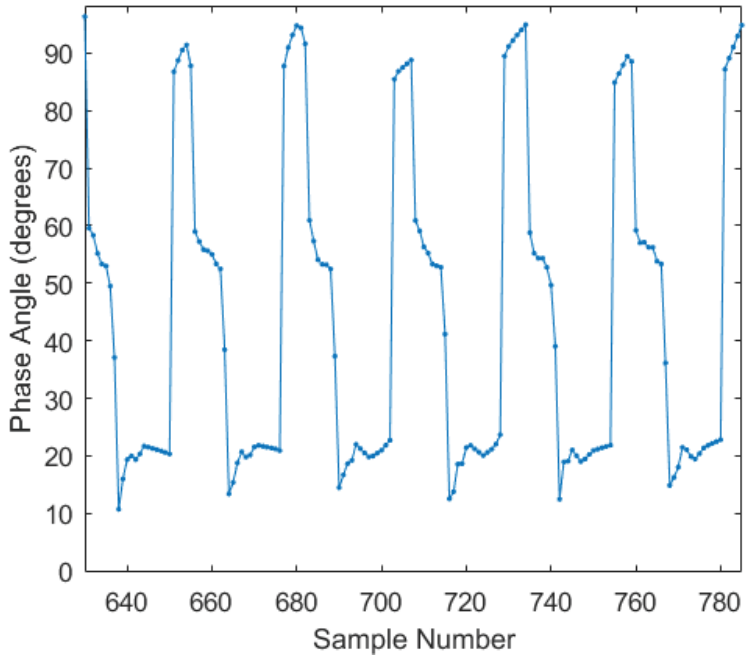


Figure 30. Phase Angle based on first few pulse cycles

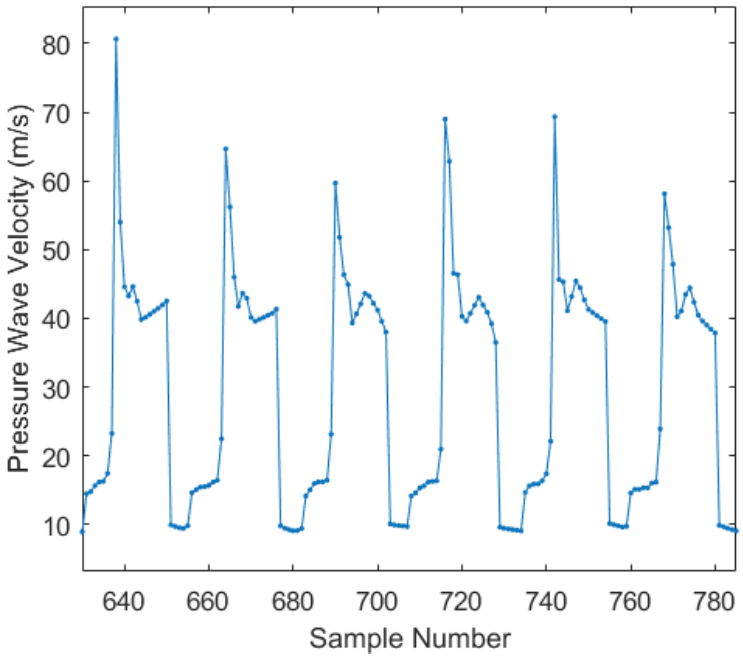


Figure 31. Pressure wave velocity based on first few pulse cycles

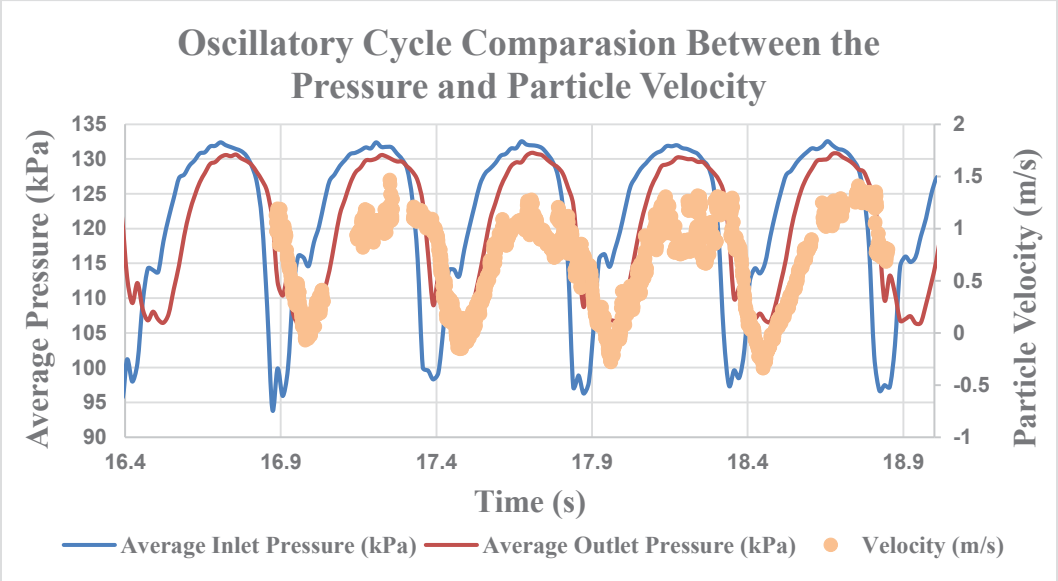
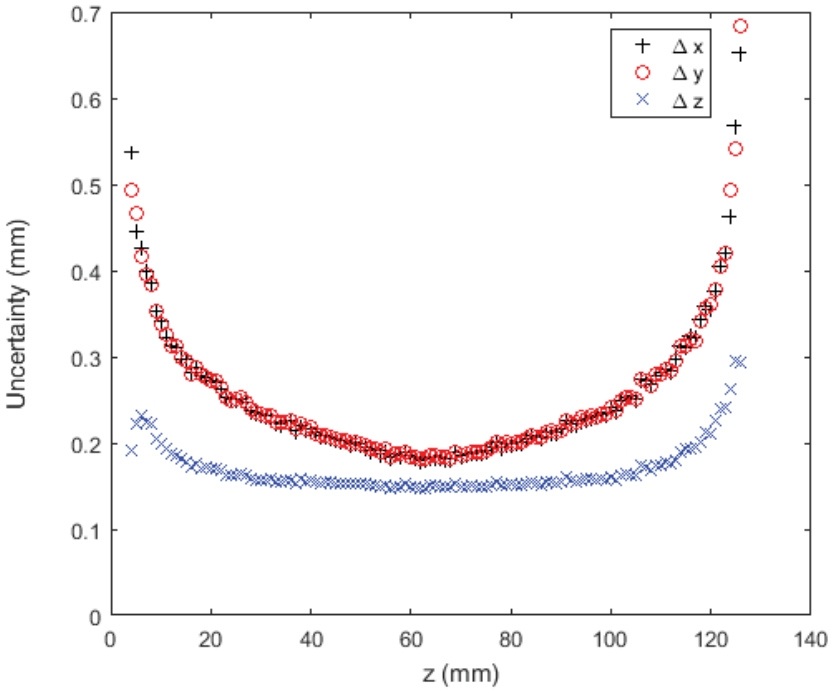
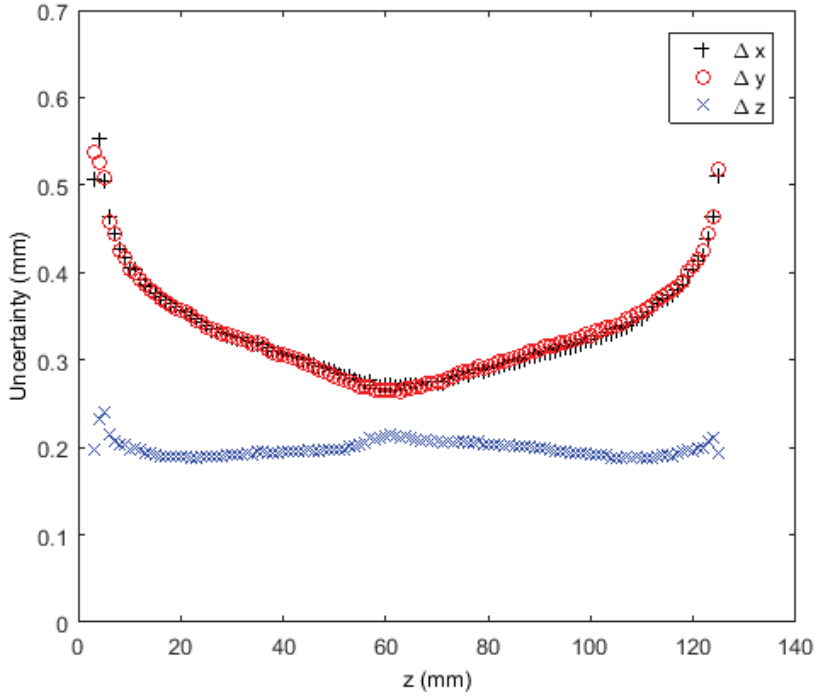


Figure 32. Comparison between the absolute pressure measurement and point by point individual particle velocity

Figure 33. Average location uncertainty in PEPT experiments as function of axial location A). Steady flow experiment B). Pulsatile flow with pinched-cross section tube C). Pulsatile flow with circular cross section tube

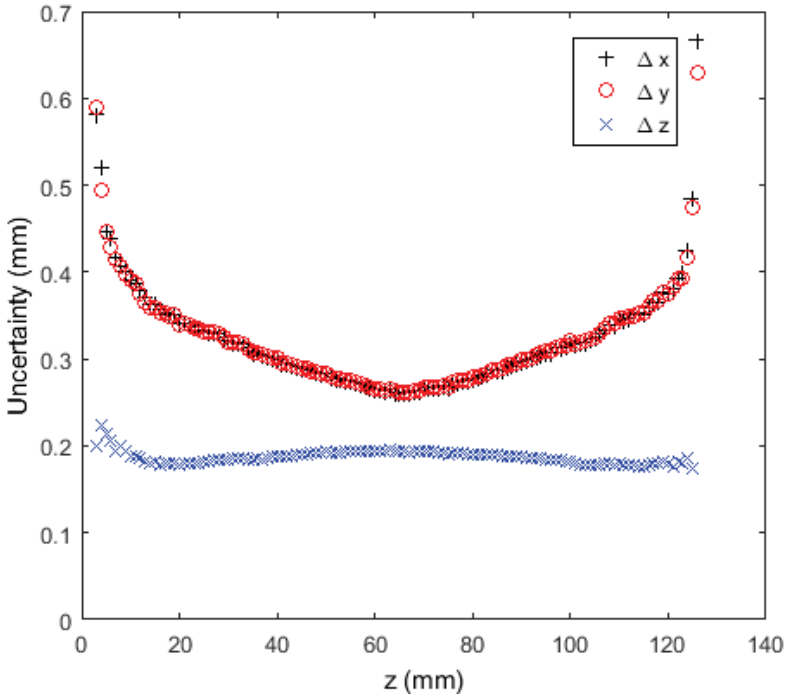


(A)



(B)

Figure 33. Continued



(C)

Figure 33. Continued

CHAPTER SIX

CONCLUSION AND GROUP CONTRIBUTION

A pulsatile flow in a circular and pinched cross-section tube along with steady flow in a PVC tube of 19 mm diameter is measured using Multi-PEPT. Multiple flow characteristics such as velocity profile, velocity field, turbulent kinetic energy and dispersion characteristics are presented. By analyzing these results, it can be concluded that PEPT is capable of characterizing transient repeating pulsatile flow with 2 Hz frequency and mean velocities ranging from 0 to 1 m/s. The PEPT data are measured in the Lagrangian frame and can be used for flow turbulence measurement. Therefore, the particle dispersion and turbulent kinetic energy are assessed. The work present in this thesis also shows an additional application of this measurement. The PEPT measurement can be used to conduct in-depth fluid flow study for hemodynamic application as well as for many hydraulic and mechanical systems.

There are a few aspects of the experiment which could be improved to get better results. The elastic tube caused many wave reflection artifacts in the pulsatile flow. Therefore, it was difficult to understand turbulent kinetic energy and dispersion features in pulsatile flow. In the future, an experiment with rigid tube should be performed to eliminate wave reflections and understand the true pulsatile flow features. Tracer particles in these studies have density near 1.2 times that of the water, and are too large to be considered true fluid tracers. The anion resin beads also degrade after prolonged circualation in the flow loop, limiting the number of traces that can be collected during a single experiment. A tracer particle of smaller diameter with higher activity would lower uncertainty in the measurement. More traces collected would also lead to better statistical data which could be useful for small scale flow turbulence measurement. Lastly, a few improvements in data assessment software could yield better quality results.

Group Contribution

The work presented in this thesis is a result of a group effort provided by the following individuals: Matthew Buttery, Howard Cyr, Seth Langford, Dr. Arthur Ruggles, Roque Santos and Cody Wiggins. Matthew Buttery assisted with the preliminary experiment conducted with the first generation motorized ball valve. He was also the original designer of the pump driven flow loop. Since then the flow loop went through revisions and the most recent design is presented herein. Howard Cyr is a head of the Geoarchaeological and Paleoenvironmental Service Center at UT Core Facilities. He generously gave us access to the Mastersizer 3000 and helped with the particle size measurements. Seth Langford helped with modification of the flow loop as well as experimental activities. Dr. Arthur

Ruggles gave some valuable insight for planning and preparations for the series of experiments described. He was also in charge of the particle activation process for each experiment. Roque Santos operated the PET scanner and assisted with motorized ball valve calibration activities. Cody Wiggins contributed by developing the multi-PEPT algorithm and assisting with post processing some of the data presented herein. He also helped with the experimental activities.

My contribution towards this research involved experimental work as well as few post processing activities. For experimental work, I designed and calibrated each of the three generations of the motorized ball valve and was involved in the construction of the original and all the post revised designs of the flow loop. Seth Langford and I also designed, purchased and assembled the instrumentation suit used herein. I wrote the procedures and data collection sheets for each of the experiments. A standard operating procedure of the flow loop is seen in Appendix B of Langford (2016). Howard Cyr and I operated the particle size analyzer and conducted a particle size measurement on wet anion exchange resins. I was involved with preparation, performance and cleanup process of each experiment. In post processing work, I analyzed the particle size measurements using the Mastersizer 3000 software and calculated the pressure wave velocity as well as velocity profile. I also wrote an algorithm to assess time independent particle dispersion and turbulent kinetic energy. I also coordinated construction and address of reviewer comments of a paper, “Positron emission particle tracking in pulsatile flow” in the journal *experiments in fluids* [23]. I also prepared 5 posters and 2 conference papers as primary author on research related to M-PEPT.

REFERENCES

[1] M. R. Hawkesworth, D. J. Parker, P. Fowles, J. F. Crilly, N. L. Jefferies, and G. Jonkers, “Nonmedical applications of a positron camera,” *Nucl. Inst. Methods Phys. Res. A*, vol. 310, no. 1–2, pp. 423–434, 1991.
[2] B. Y. Zhang, “Positron Emission Tomography (PET) for Flow Measurement,” p. 10, 2011.
[3] D. J. Parker and X. Fan, “Positron emission particle tracking-Application and labelling techniques,” *Particuology*, vol. 6, no. 1, pp. 16–23, 2008.
[4] D. J. Parker, C. J. Broadbent, P. Fowles, M. R. Hawkesworth, and P. McNeil, “Positron emission particle tracking - a technique for studying flow within engineering equipment,” *Nucl. Inst. Methods Phys. Res. A*, vol. 326, no. 3, pp. 592–607, 1993.
[5] Z. Yang, D. J. Parker, P. J. Fryer, S. Bakalis, and X. Fan, “Multiple-particle tracking-an improvement for positron particle tracking,” *Nucl. Instruments Methods Phys. Res. Sect. A Accel. Spectrometers, Detect. Assoc. Equip.*, vol. 564, no. 1, pp. 332–338, 2006.
[6] A. C. Hoffmann, C. Dechsiri, F. van de Wiel, and H. G. Dehling, “PET investigation of a fluidized particle: spatial and temporal resolution and short term motion,” *Meas. Sci. Technol.*, vol. 16, no. 3, pp. 851–858, 2005.
[7] M. Bickell, A. Buffler, I. Govender, and D. J. Parker, “A new line density tracking algorithm for PEPT and its application to multiple tracers,” *Nucl. Instruments Methods Phys. Res. Sect. A Accel. Spectrometers, Detect. Assoc. Equip.*, vol. 682, pp. 36–41, 2012.
[8] C. Wiggins, R. Santos, and A. Ruggles, “A novel clustering approach to positron emission particle tracking,” *Nucl. Instruments Methods Phys. Res. Sect. A Accel. Spectrometers, Detect. Assoc. Equip.*, vol. 811, pp. 18–24, 2016.
[9] C. Wiggins, R. Santos, and A. Ruggles, “A feature point identification method for positron emission particle tracking with multiple tracers,” *Nucl. Instruments Methods Phys. Res. Sect. A Accel. Spectrometers, Detect. Assoc. Equip.*, vol. 843, pp. 22–28, 2017.
[10] M. Haslam and M. Zamir, “Pulsatile flow in tubes of elliptic cross sections.,” *Ann. Biomed. Eng.*, vol. 26, no. 5, pp. 780–7, 1998.
[11] M. Zamir and R. Budwig, “Physics of Pulsatile Flow,” *Appl. Mech. Rev.*, vol. 55, no. 2, p. B35, 2002.
[12] R. Trip, D. J. Kuik, J. Westerweel, and C. Poelma, “An experimental study of transitional pulsatile pipe flow,” *Phys. Fluids*, vol. 24, no. 1, 2012.
[13] A. Y. Usmani and K. Muralidhar, “Pulsatile flow in a compliant stenosed asymmetric model,” *Exp. Fluids*, vol. 57, no. 12, 2016.
[14] A. Anssari-Benam, M. Tafazzoli-Shadpour, N. Fatouraee, A. Pashaiee, and M. M. Khani, “A new system to analyze pulsatile flow characteristics in elastic tubes for hemodynamic applications,” *Am. J. Appl. Sci.*, vol. 5, no. 12, pp. 1730–1736, 2008.
[15] Q. Bao, D. Newport, M. Chen, D. B. Stout, and A. F. Chatziioannou, “Performance Evaluation of the Inveon Dedicated PET Preclinical

Tomograph Based on the NEMA NU-4 Standards," *J. Nucl. Med.*, vol. 50, no. 3, pp. 401–408, 2009.
[16] J. C. Crocker and D. G. Grier, "Methods of Digital Video Microscopy for Colloidal Studies," *J. Colloid Interface Sci.*, vol. 179, no. 1, pp. 298–310, 1996.
[17] I. F. Sbalzarini and P. Koumoutsakos, "Feature point tracking and trajectory analysis for video imaging in cell biology," *J. Struct. Biol.*, vol. 151, no. 2, pp. 182–195, 2005.
[18] Y. A. Hassan and R. E. Canaan, "Full-field bubbly flow velocity measurements using a multiframe particle tracking technique," *Exp. Fluids*, vol. 12, no. 1–2, pp. 49–60, 1991.
[19] H. Kuhn, "The Hungarian Method for the assignment problem," *Nav. Res. Logist. Q.*, vol. 2, pp. 83–97, 1955.
[20] H. Kuhn, "Variants of the Hungarian method for assignment problems," *Nav. Res. Logist. Q.*, vol. 3, pp. 253–258, 1956.
[21] J. Munkres, "Algorithms for the Assignment and Transportation Problems," *J. Soc. Ind. Appl. Math.*, vol. 5, no. 1, pp. 32–38, 1957.
[22] S. T. Langford, "Jet Flow Validation of Positron Emission Particle Tracking Utilizing High Speed Video," 2016.
[23] N. Patel, C. Wiggins, and A. Ruggles, "Positron emission particle tracking in pulsatile flow," *Exp. Fluids*, vol. 58, no. 5, pp. 1–12, 2017.

APPENDIX

Appendix A – Steady Flow Diffusion, TKE and Velocity Profile Calculation

```
clc
close all
clear all

Trajectories=312;
%Trajectories
lastlength=0;
%Initializing the matrix dimension
length_prv=0;
% Data_prv=ones(500000,3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Reading each position files and performing various calculations
%
% Reading the files and filtering trajectory files with less
%
% than 4 data points.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%


waitbar1=waitbar(0,'Please Wait 1/3 ....');
for avg=1:Trajectories

File_name=sprintf('FilteredPosition%d.txt',avg);
% Filename
read_file_prv=dlmread(File_name,'\t',2,0);
% Reading the file

[m,n]=size((read_file_prv));

if m>4

for avg2=(1:m)
Data_prv((length_prv+avg2),1)=(read_file_prv(avg2,2));
% x-position
Data_prv((length_prv+avg2),2)=(read_file_prv(avg2,3));
% y-position
Data_prv((length_prv+avg2),3)=(read_file_prv(avg2,4));
% z-position
Data_prv_A(length_prv+avg2,1)=avg;
% Trajectory incrementor

end
end
```

```
length_prv=length(Data_prv_A);
waitbar(avg/Trajectories);

end

close(waitbar1)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Centering the pipe by subtracting the data set by average position
%
% Average is calculated by (max+min)/2. Radius and Angle are calculated
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

mean_x=(max(Data_prv(:,1))+min(Data_prv(:,1)))/2;
% Average velocity in x direction
mean_y=(max(Data_prv(:,2))+min(Data_prv(:,2)))/2;
% Average velocity in y direction
mean_z=(max(Data_prv(:,3))+min(Data_prv(:,3)))/2;
% Average velocity in z direction
Data=0;
% Data=ones(500000,7);
last_length=0;
ii=0;
hh=1;
traj=0;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%                                User Inputs
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

prompt0=input('Plot 3D image of trajectories (Y/N)?','s');
Prompt_1=input('Would you like to analyze TKE (Y/N)?','s');

if Prompt_1=='Y'
prompt_slice=input('How many concentric circular divisions?');
Prompt_2=input('Plot individual trajectories (Y/N)?','s');
Prompt_3=input('Plot velocity fluctuation (Y/N)?','s');

end
```

```
Prompt_4=input('Would you like to assess the diffusivity (Y/N)?','s');

if Prompt_4=='Y'
prompt_x_position=input('Enter the X position of interest(mm):');
% User input X position of interest (mm)
prompt_y_position=input('Enter the Y position of interest(mm):');
% User input Y position of interest (mm)
prompt_z_position=input('Enter the Z position of interest (mm):');
% User input Z position of interest (mm)
prompt_Xgrid_size=input('Enter the X grid size (mm):');
% User input grid size (mm)
prompt_Ygrid_size=input('Enter the Y grid size (mm):');
% User input grid size (mm)
prompt_Zgrid_size=input('Enter the Z grid size (mm):');
% User input grid size (mm)
prompt_Z_Traj_limit=input('Enter the minimum trajectory length (mm):');
% User input the minimum trajectory length
prompt_position=input('How many positions would you like to
analyze?:');              % User input the number of analysis to be
conducted
prompt5=input('Enter Z position where you want to evaluate the
distribution (mm):');         % User input the Z position threshold of
interest

end

Prompt_5=input('Would you like to analyze the velocity profile
(Y/N)?','s');

if Prompt_5=='Y'
radius_div=input('Enter the number of radius divisions to get single
velocity profile:');
lower_diameter_val=input('Enter the upper diameter value:');
upper_diameter_val=input('Enter the lower diameter value:');
divisions=input('Enter the number of divisions for the 1/7th law:');
radius_val=input('Enter the radius of the tube:');

end


waitbar2=waitbar(0,'Please Wait 2/3....');

for T=1:Trajectories

File_name=sprintf('FilteredPosition%d.txt',T);
% Filename
read_file=dlmread(File_name,'\t',2,0);
% Reading the file

[m,n]=size((read_file));

if prompt0=='Y'
```

```
figure(1)

plot3((127-(read_file(:,4))),(read_file(:,2)-mean_x),(read_file(:,3)-
mean_y))                                % plotting the z vs. y
xlabel('Z position (mm)')
ylabel('X position (mm)')
zlabel('Y position (mm)')


hold on

end

if m>4

traj=traj+1;

for a=(1:m)

Data(last_length+a,1)=T;
% trajectory
% time_adj(a,1)=((read_file(a,1))-(read_file(1,1)));
% time adjustment to ensure all the position files start at t=0 ms
time_adj(a,1)=(read_file(a,1));
Data((last_length+a),2)=time_adj(a,1);
% adjusted time (ms)
Data(last_length+a,3)=(read_file(a,2)-mean_x);
% x position (mm)
Data(last_length+a,4)=(read_file(a,3)-mean_y);
% y position (mm)
Data(last_length+a,5)=sqrt(((read_file(a,2)-
mean_x)^(2))+((read_file(a,3)-mean_y)^(2)));  % Radius (mm)
Data(last_length+a,6)=(127-(read_file(a,4)));
% z position (mm)
Data(last_length+a,7)=rad2deg(atan((read_file(a,3)-
mean_y)/(read_file(a,2)-mean_x)));     % angle (degree)
Data_A(last_length+a,1)=T;
% title={'Trajectory' 'Adjusted Time (s)' 'X Position (mm)' 'Y Position
(mm)' 'Radius (mm)' 'Z Position (mm)' 'Angle'};
% Data_Matrix=[title;num2cell(Data(:,:))];

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Calculating the velocity of each coordinate direction (x,y,z)
%
% The velocity coordinate system are changed from cartesian
%
% to cylindrical coordinate system.
%
```

```
% Mean velocity and velocity fluctuations of each trajectory
%
% are calculated.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
for i=(1:(m-1))

    delta_u(last_length+i-ii,1)=(Data(last_length+i+1,3)-
(Data(last_length+i,3)));              % Change in x-position
    delta_v(last_length+i-ii,1)=(Data(last_length+i+1,4)-
(Data(last_length+i,4)));              % Change in y-position
    delta_w(last_length+i-ii,1)=(Data(last_length+i+1,6)-
(Data(last_length+i,6)));              % Change in z-position
    delta_t(last_length+i-ii,1)=(Data(last_length+i+1,2)-
Data(last_length+i,2));                % Change in time

    u_t_zero(hh,1)=(delta_u(last_length+i-
ii,1))/(delta_t(last_length+i-ii,1));
    v_t_zero(hh,1)=(delta_v(last_length+i-
ii,1))/(delta_t(last_length+i-ii,1));
    w_t_zero(hh,1)=(delta_w(last_length+i-
ii,1))/(delta_t(last_length+i-ii,1));

    u_t(last_length+i-ii,1)=(delta_u(last_length+i-
ii,1))/(delta_t(last_length+i-ii,1));       % x velocity (m/s)
    v_t(last_length+i-ii,1)=(delta_v(last_length+i-
ii,1))/(delta_t(last_length+i-ii,1));       % y velocity (m/s)

    v_r_zero(hh,1)=(u_t(last_length+i-
ii,1))*cosd(Data(last_length+i+1,7))...
                  +(v_t(last_length+i-
ii,1))*sind(Data(last_length+i+1,7));
    v_theta_zero(hh,1)=-(u_t(last_length+i-
ii,1))*sind(Data(last_length+i+1,7))...
                      +(v_t(last_length+i-
ii,1))*cosd(Data(last_length+i+1,7));

    Modata(last_length+i-ii,1)=Data(last_length+i+1,1);
% Trajectory
    Modata(last_length+i-ii,2)=Data(last_length+i+1,2);
% Time (ms)
    x_val_zero(hh,1)=Data(last_length+i+1,3);
    y_val_zero(hh,1)=Data(last_length+i+1,4);
    z_val_zero(hh,1)=Data(last_length+i+1,6);
    Modata(last_length+i-ii,3)=Data(last_length+i+1,3);
% X-position (mm)
    Modata(last_length+i-ii,4)=Data(last_length+i+1,4);
% Y-position (mm)
    Modata(last_length+i-ii,5)=Data(last_length+i+1,6);
% Z-position (mm)
    Modata(last_length+i-ii,6)=Data(last_length+i+1,7);
% Theta (degrees)
```

```
    if Modata(last_length+i-ii,6)<0
    radius(last_length+i-ii,1)=-Data(last_length+i+1,5);
    else
        radius(last_length+i-ii,1)=Data(last_length+i+1,5);
    end

    Modata(last_length+i-ii,7)=radius(last_length+i-ii,1);
% Radius (mm)


    Modata(last_length+i-ii,8)=(u_t(last_length+i-
ii,1))*cosd(Data(last_length+i+1,7))...
                              +(v_t(last_length+i-
ii,1))*sind(Data(last_length+i+1,7));        % V_r Velocity (m/s)
    Modata(last_length+i-ii,9)=-(u_t(last_length+i-
ii,1))*sind(Data(last_length+i+1,7))...
                              +(v_t(last_length+i-
ii,1))*cosd(Data(last_length+i+1,7));        % V_theta Velocity (m/s)
    Modata(last_length+i-ii,10)=(delta_w(last_length+i-
ii,1))/(delta_t(last_length+i-ii,1));   % W_velocity (m/s)



    hh=hh+1;


end

%%  Mean Velocity of each trajectories  %%

    v_r_bar(traj,1)=sum(v_r_zero(:,1))/(hh);
    v_theta_bar(traj,1)=sum(v_theta_zero(:,1))/(hh);
    w_bar(traj,1)=sum(w_t_zero(:,1))/(hh);

%% Velocity fluctuations of each trajectories %%

for xy=(1:(m-1))

    Modata(last_length+xy-ii,11)=v_r_bar(traj,1);
% Mean V_r
    Modata(last_length+xy-ii,12)=v_theta_bar(traj,1);
% Mean V_theta
    Modata(last_length+xy-ii,13)=w_bar(traj,1);
% Mean V_z
    Modata(last_length+xy-ii,14)=(Modata(last_length+xy-ii,8))-
(v_r_bar(traj,1));          % Velocity fluctuations (V_r')
    Modata(last_length+xy-ii,15)=(Modata(last_length+xy-ii,9))-
(v_theta_bar(traj,1));      % Velocity fluctuations (V_theta')
    Modata(last_length+xy-ii,16)=(Modata(last_length+xy-ii,10))-
(w_bar(traj,1));           % Velocity fluctuations (w')
```

```
end


% Clearing each elements for plotting individual trajectories %

u_t_zero(:,:)=[];
v_t_zero(:,:)=[];
v_r_zero(:,:)=[];
v_theta_zero(:,:)=[];
w_t_zero(:,:)=[];
x_val_zero(:,:)=[];
y_val_zero(:,:)=[];
z_val_zero(:,:)=[];

% Matrix array increments %%

last_length=length(Data_A);
ii=ii+1;
hh=1;

end

waitbar(T/Trajectories);

end

close(waitbar2)

title2={'Trajectory' 'Adjusted Time (ms)' 'X Position (mm)'...
    'Y Position (mm)' 'Z position (mm)' 'Theta (degrees)'...
    'Radius (mm)' 'V_r (m/s)' 'V_theta (m/s)' 'V_z (m/s)'...
    'Mean V_r (m/s)' 'Mean V_theta (m/s)' 'Mean V_z (m/s)'...
    'V_r fluctuations (m/s)' 'V_theta fluctuations (m/s)'...
    'V_z fluctuations (m/s)'};
Modata_labled=([title2;num2cell(Modata(:,:))]);
% dlmwrite('Turbulence_data_file.txt',Modata,'\t')



% Plotting the velocity profile as function of radius of the pipe

figure

scatter(Modata(:,10),radius(:,1),'.')
title('Radial Position as Function of Velocity')
xlabel('Velocity (m/s)')
ylabel('Radius (mm)')
savefig('Steady_flow_vel_profile')
```

```
figure

subplot(3,1,1)
plot(Modata(:,7),Modata(:,14),'.')
title('flux as function of radius')
xlabel('Radius (mm)')
ylabel('vr flux (m/s)')

subplot(3,1,2)
plot(Modata(:,7),Modata(:,15),'.')
title('flux as function of radius')
xlabel('Radius (mm)')
ylabel('vtheta flux (m/s)')

subplot(3,1,3)
plot(Modata(:,7),Modata(:,16),'.')
title('flux as function of radius')
xlabel('Radius (mm)')
ylabel('vz flux (m/s)')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%                           flux parts
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

Nxgrid=22;  % Grid Size
Nygrid=1;
gridx=(max(Modata(:,7))*2)/(Nxgrid); % step size (mm)
gridy=(max(Modata(:,16))*2)/(Nygrid);  % step size y (mm)
low_r=-max(Modata(:,7));  % Initial Lower limit (mm)
high_r=-max(Modata(:,7))+gridx;  % Initial Upper limit (mm)
low_flx=-max(Modata(:,16));
high_flx=-max(Modata(:,16))+gridy;
ia=1;
ib=1;

for gx=(1:Nxgrid)

    for gx_inc=(1:length(Modata))

     if Modata(gx_inc,7)>=low_r && Modata(gx_inc,7)<high_r...          %
Logic states to include all the parameters within
         || Modata(gx_inc,7)>=-low_r && Modata(gx_inc,7)<-high_r      %
the radius limits based on the divisions
        gx_traj(ia,1)=Modata(gx_inc,1);
        if ia>=2
            if gx_traj(ia-1,1)~=gx_traj(ia,1)
               gx_num_traj=gx_traj+1;
               gx_traj=gx_num_traj;
```

```
            end
        end
        gx_v_r1(ia,1)=Modata(gx_inc,8);
        gx_v_theta1(ia,1)=Modata(gx_inc,9);
        gx_w_t1(ia,1)=Modata(gx_inc,10);
        gx_time1(ia,1)=Modata(gx_inc,2);
        gx_theta1(ia,1)=Modata(gx_inc,6);
        gx_radius1(ia,1)=Modata(gx_inc,7);
        gx_v_r_bar1(ia,1)=Modata(gx_inc,11);
        gx_v_theta_bar1(ia,1)=Modata(gx_inc,12);
        gx_w_bar1(ia,1)=Modata(gx_inc,13);
        gx_v_r_flx1(ia,1)=Modata(gx_inc,14);
        gx_v_r_flx1_square(ia,1)=(Modata(gx_inc,14))^(2);
        gx_v_theta_flx1(ia,1)=Modata(gx_inc,15);
        gx_v_theta_flx1_square(ia,1)=(Modata(gx_inc,15))^(2);
        gx_w_flx1(ia,1)=Modata(gx_inc,16);
        gx_w_flx1_square(ia,1)=(Modata(gx_inc,16))^(2);
        ia=ia+1;
     end

    end

    for gy=(1:Nygrid)


    for gy_inc=(1:length(gx_v_r1))

     if gx_w_flx1(gy_inc,1)>=low_flx && gx_w_flx1(gy_inc,1)<high_flx
% Logic states to include all the parameters within

% the radius limits based on the divisions

        gy_traj(ib,1)=gx_traj(gy_inc,1);
        if ib>=2
            if gy_traj(ib-1,1)~=gy_traj(ib,1)
               gy_num_traj=gy_traj+1;
               gy_traj=gy_num_traj;
            end
        end
        gy_v_r1(ib,1)=gx_v_r1(gy_inc,1);
        gy_v_theta1(ib,1)=gx_v_theta1(gy_inc,1);
        gy_w_t1(ib,1)=gx_w_t1(gy_inc,1);
        gy_time1(ib,1)=gx_time1(gy_inc,1);
        gy_theta1(ib,1)=gx_theta1(gy_inc,1);
        gy_radius1(ib,1)=gx_radius1(gy_inc,1);
        gy_v_r_bar1(ib,1)=gx_v_r_bar1(gy_inc,1);
        gy_v_theta_bar1(ib,1)=gx_v_theta_bar1(gy_inc,1);
        gy_w_bar1(ib,1)=gx_w_bar1(gy_inc,1);
        gy_v_r_flx1(ib,1)=gx_v_r_flx1(gy_inc,1);
        gy_v_r_flx1_square(ib,1)=(gx_v_r_flx1(gy_inc,1))^(2);
        gy_v_theta_flx1(ib,1)=gx_v_theta_flx1(gy_inc,1);
        gy_v_theta_flx1_square(ib,1)=(gx_v_theta_flx1(gy_inc,1))^(2);
        gy_w_flx1(ib,1)=gx_w_flx1(gy_inc,1);
```

```
    gy_w_flx1_square(ib,1)=(gx_w_flx1(gy_inc,1))^(2);
    ib=ib+1;

 else
     continue
 end

end

if ib==1

    low_flx=high_flx;
    high_flx=high_flx+gridy;
    continue

else

 Mean_vr(gy,gx)=(sum(gy_v_r1(:,1)))/(ib-1);
 Mean_vtheta(gy,gx)=(sum(gy_v_theta1(:,1)))/(ib-1);
 Mean_vz(gy,gx)=(sum(gy_w_t1(:,1)))/(ib-1);
 Mean_radius(gy,gx)=(sum(gy_radius1(:,1)))/(ib-1);
 Mean_vr_flx(gy,gx)=(sum(gy_v_r_flx1(:,1)))/(ib-1);
 Mean_vtheta_flx(gy,gx)=(sum(gy_v_theta_flx1(:,1)))/(ib-1);
 Mean_vz(gy,gx)=(sum(gy_w_flx1(:,1)))/(ib-1);

 low_flx=high_flx;
 high_flx=high_flx+gridy;
 ib=1;
 gy_v_r1(:,:)=[];
 gy_v_theta1(:,:)=[];
 gy_w_t1(:,:)=[];
 gy_time1(:,:)=[];
 gy_theta1(:,:)=[];
 gy_radius1(:,:)=[];
 gy_v_r_bar1(:,:)=[];
 gy_v_theta_bar1(:,:)=[];
 gy_w_bar1(:,:)=[];
 gy_v_r_flx1(:,:)=[];
 gy_v_r_flx1_square(:,:)=[];
 gy_v_theta_flx1(:,:)=[];
 gy_v_theta_flx1_square(:,:)=[];
 gy_w_flx1(:,:)=[];
 gy_w_flx1_square(:,:)=[];

end


end

% Increments

 low_r=high_r;
```

```
    high_r=high_r+gridx;
    ia=1;
    ib=1;
    low_flx=-max(Modata(:,16));
    high_flx=-max(Modata(:,16))+gridy;

    % Clearing individual trajectory information

    gx_v_r1(:,:)=[];
    gx_v_theta1(:,:)=[];
    gx_w_t1(:,:)=[];
    gx_time1(:,:)=[];
    gx_theta1(:,:)=[];
    gx_radius1(:,:)=[];
    gx_v_r_bar1(:,:)=[];
    gx_v_theta_bar1(:,:)=[];
    gx_w_bar1(:,:)=[];
    gx_v_r_flx1(:,:)=[];
    gx_v_r_flx1_square(:,:)=[];
    gx_v_theta_flx1(:,:)=[];
    gx_v_theta_flx1_square(:,:)=[];
    gx_w_flx1(:,:)=[];
    gx_w_flx1_square(:,:)=[];


end







%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%
%
%                            TKE Analysis
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

% Prompt_1=input('Would you like to analyze TKE?','s')

if Prompt_1=='Y'

% Plotting five divisions of the velocity profile
% Steps are divided according to the highest radius values %
```

```
diameter=max(radius(:,1));
% diameter (mm)
division=prompt_slice;
% division
steps=(diameter/division);
% step size (mm)
y=1;
Bottom_low=-max(radius(:,1));
% Initial Lower limit (mm)
Bottom_high=-max(radius(:,1))+steps;
% Initial Upper limit (mm)
Top_low=max(radius(:,1))-steps;
Top_high=max(radius(:,1));

o=1;
oo=1;
length_flx=0;
traj=1;

% Prompt_2=input('Plot individual trajectories (Y/N)?','s')
% Prompt_3=input('Plot velocity fluctuation (Y/N)?','s')

waitbar3=waitbar(0,'Please Wait 3/3....');

for z=(1:(division))



Bottom_low_step_limit=Bottom_low;
Bottom_high_step_limit=Bottom_high;
Top_low_step_limit=Top_low;
Top_high_step_limit=Top_high;

%% Dividing the profile

for x=(1:length(Modata))

    if Modata(x,7)>=Bottom_low_step_limit &&
Modata(x,7)<Bottom_high_step_limit...         % Logic states to include
all the parameters within
            || Modata(x,7)>=Top_low_step_limit &&
Modata(x,7)<Top_high_step_limit          % the radius limits based on
the divisions

        traj1(y,1)=Modata(x,1);
        if y>=2
            if traj1(y-1,1)~=traj1(y,1)
               num_traj=traj+1;
               traj=num_traj;
            end
        end
```

```
        v_r1(y,1)=Modata(x,8);
        v_theta1(y,1)=Modata(x,9);
        w_t1(y,1)=Modata(x,10);
        time1(y,1)=Modata(x,2);
        theta1(y,1)=Modata(x,6);
        radius1(y,1)=Modata(x,7);
        v_r_bar1(y,1)=Modata(x,11);
        v_theta_bar1(y,1)=Modata(x,12);
        w_bar1(y,1)=Modata(x,13);
        v_r_flx1(y,1)=Modata(x,14);
        v_r_flx1_square(y,1)=(Modata(x,14))^(2);
        v_theta_flx1(y,1)=Modata(x,15);
        v_theta_flx1_square(y,1)=(Modata(x,15))^(2);
        w_flx1(y,1)=Modata(x,16);
        w_flx1_square(y,1)=(Modata(x,16))^(2);
        y=y+1;
    end

end

% Turbulent Kinetic Energy Assesment of each division

Mean_v_r_flx1_square(z,1)=(sum(v_r_flx1_square(:,1)))/(y);
Mean_v_theta_flx1_square(z,1)=(sum(v_theta_flx1_square(:,1)))/(y);
Mean_w_flx1_square(z,1)=(sum(w_flx1_square(:,1)))/(y);
TKE(z,1)=(1/2)*((Mean_v_r_flx1_square(z,1))+(Mean_v_theta_flx1_square(z
,1))...
         +(Mean_w_flx1_square(z,1)));
Avg_bottom(z,1)=((Bottom_high_step_limit)+(Bottom_low_step_limit))/2;
Avg_top(z,1)=((Top_high_step_limit)+(Top_low_step_limit))/2;
v_r_flx1_avg(z,1)=(sum(v_r_flx1(:,1)))/(y);
v_theta_flx1_avg(z,1)=(sum(v_theta_flx1(:,1)))/(y);
v_z_flx1_avg(z,1)=(sum(w_flx1(:,1)))/(y);
htraj(z,1)=num_traj;


%% Plotting the results %%

bb=1;


if Prompt_2=='Y'

figure


%% Plotting Individual Trajectories %%

for w=(1:Trajectories)

for b=(1:length(traj1))
```

```
    if traj1(b,1)==w
        traj1_zero(bb,1)=traj1(b,1);
        v_r1_zero(bb,1)=v_r1(b,1);
        v_theta1_zero(bb,1)=v_theta1(b,1);
        w_t1_zero(bb,1)=w_t1(b,1);
        time1_zero(bb,1)=time1(b,1);
        theta1_zero(bb,1)=theta1(b,1);
        radius1_zero(bb,1)=radius1(b,1);
        v_r_bar1_zero(bb,1)=v_r_bar1(b,1);
        v_theta_bar1_zero(bb,1)=v_theta_bar1(b,1);
        w_bar1_zero(bb,1)=w_bar1(b,1);
        v_r_flx1_zero(bb,1)=v_r_flx1(b,1);
        v_theta_flx1_zero(bb,1)=v_theta_flx1(b,1);
        w_flx1_zero(bb,1)=w_flx1(b,1);
        bb=bb+1;




    end

end

if bb>1


subplot(3,3,1)
plot(v_r1_zero,radius1_zero,'*')
hold on
title(sprintf('Radial Position vs. V_r(t) from r=%.02f\n to r=%.02f\n,
r=%.02f\n to r=%.02f\n'...

,Bottom_low_step_limit,Bottom_high_step_limit,Top_low_step_limit,Top_hi
gh_step_limit))

xlabel('V_r(t) (m/s)')
ylabel('Radius (mm)')

subplot(3,3,2)
plot(time1_zero,v_r1_zero)
hold on
title(sprintf('V_r(t) vs. Time from r=%.02f\n to r=%.02f\n, r=%.02f\n
to r=%0.02f\n'...

,Bottom_low_step_limit,Bottom_high_step_limit,Top_low_step_limit,Top_hi
gh_step_limit))

xlabel('Time (ms)')
ylabel('V_r(t) (m/s)')
```

```
subplot(3,3,3)

plot(time1_zero,v_r_bar1_zero)
hold on
title(sprintf('V_r(bar) vs. Time from r=%.02f\n to r=%.02f\n, r=%.02f\n
to r=%0.02f\n'...

,Bottom_low_step_limit,Bottom_high_step_limit,Top_low_step_limit,Top_hi
gh_step_limit))

xlabel('Time (ms)')
ylabel('Mean Velocity (m/s)')

subplot(3,3,4)
plot(v_theta1_zero,radius1_zero,'*')
hold on
title(sprintf('Radial Position vs. V_theta(t) from r=%.02f\n to
r=%.02f\n, r=%.02f\n to r=%.02f\n'...

,Bottom_low_step_limit,Bottom_high_step_limit,Top_low_step_limit,Top_hi
gh_step_limit))

xlabel('V_(theta)(t) (m/s)')
ylabel('Radius (mm)')

subplot(3,3,5)
plot(time1_zero,v_theta1_zero)
hold on
title(sprintf('V_theta(t) vs. Time from r=%.02f\n to r=%.02f\n,
r=%.02f\n to r=%0.02f\n'...

,Bottom_low_step_limit,Bottom_high_step_limit,Top_low_step_limit,Top_hi
gh_step_limit))

xlabel('Time (ms)')
ylabel('V_theta(t) (m/s)')

subplot(3,3,6)

plot(time1_zero,v_theta_bar1_zero)
hold on
title(sprintf('V_theta(bar) vs. Time from r=%.02f\n to r=%.02f\n,
r=%.02f\n to r=%0.02f\n'...

,Bottom_low_step_limit,Bottom_high_step_limit,Top_low_step_limit,Top_hi
gh_step_limit))

xlabel('Time (ms)')
ylabel('Mean Velocity (m/s)')
```

```
subplot(3,3,7)
plot(w_t1_zero,radius1_zero,'*')
hold on
title(sprintf('Radial Position vs. V_z(t) from r=%.02f\n to r=%.02f\n,
r=%.02f\n to r=%.02f\n'...

,Bottom_low_step_limit,Bottom_high_step_limit,Top_low_step_limit,Top_hi
gh_step_limit))

xlabel('V_z(t) (m/s)')
ylabel('Radius (mm)')

subplot(3,3,8)
plot(time1_zero,w_t1_zero)
hold on
title(sprintf('V_z(t) of Time from r=%.02f\n to r=%.02f\n, r=%.02f\n to
r=%0.02f\n'...

,Bottom_low_step_limit,Bottom_high_step_limit,Top_low_step_limit,Top_hi
gh_step_limit))

xlabel('Time (ms)')
ylabel('V_z(t) (m/s)')

subplot(3,3,9)

plot(time1_zero,w_bar1_zero)
hold on
title(sprintf('Mean Velocity as function of Time from r=%.02f\n to
r=%.02f\n, r=%.02f\n to r=%0.02f\n'...

,Bottom_low_step_limit,Bottom_high_step_limit,Top_low_step_limit,Top_hi
gh_step_limit))

xlabel('Time (ms)')
ylabel('Mean Velocity (m/s)')

% Clearing individual trajectory information for plotting purposes



traj1_zero(:,:)=[];
v_r1_zero(:,:)=[];
v_theta1_zero(:,:)=[];
w_t1_zero(:,:)=[];
time1_zero(:,:)=[];
theta1_zero(:,:)=[];
radius1_zero(:,:)=[];
v_r_bar1_zero(:,:)=[];
v_theta_bar1_zero(:,:)=[];
w_bar1_zero(:,:)=[];
v_r_flx1_zero(:,:)=[];
```

```
v_theta_flx1_zero(:,:)=[];
w_flx1_zero(:,:)=[];
bb=1;

end

end

end


if Prompt_3=='Y'

figure


subplot (3,1,1)
hist(v_r_flx1(:,1),100)
title(sprintf('V_r Fluctuation from r=%0.02f\n to r=%0.02f\n,
r=%0.02f\n to r=%0.02f\n'...

,Bottom_low_step_limit,Bottom_high_step_limit,Top_low_step_limit,Top_hi
gh_step_limit))
xlabel('V_r Fluctuations (m/s)')
ylabel('Frequency')

subplot (3,1,2)
hist(v_theta_flx1(:,1),100)
title(sprintf('V_theta Fluctuation from r=%0.02f\n to r=%0.02f\n,
r=%0.02f\n to r=%0.02f\n'...

,Bottom_low_step_limit,Bottom_high_step_limit,Top_low_step_limit,Top_hi
gh_step_limit))
xlabel('V_theta Fluctuations (m/s)')
ylabel('Frequency')

subplot (3,1,3)
hist(w_flx1(:,1),100)
title(sprintf('V_z Fluctuation from r=%0.02f\n to r=%0.02f\n,
r=%0.02f\n to r=%0.02f\n'...

,Bottom_low_step_limit,Bottom_high_step_limit,Top_low_step_limit,Top_hi
gh_step_limit))
xlabel('V_z Fluctuations (m/s)')
ylabel('Frequency')

end

% Clearing all the plotting parameters of interest to ensure only
% individual trajectories are plotted
```

```
traj1(:,:)=[];
w_t1(:,:)=[];
time1(:,:)=[];
theta1(:,:)=[];
radius1(:,:)=[];
v_r_bar1(:,:)=[];
v_theta_bar1(:,:)=[];
w_bar1(:,:)=[];
v_r_flx1(:,:)=[];
v_theta_flx1(:,:)=[];
w_flx1(:,:)=[];
v_r_flx1_square(:,:)=[];
v_theta_flx1_square(:,:)=[];
w_flx1_square(:,:)=[];
y=1;
q=1;
traj=1;

% Matrix array incrementer

length_flx=0;

Bottom_low=Bottom_high_step_limit;
Bottom_high=Bottom_high_step_limit+steps;
Top_low=Top_high_step_limit;
Top_high=Top_low_step_limit;
Top_low=Top_low_step_limit-steps;

waitbar(z/division);

end


ly=1;
lw=1;
h=figure;
jl=1;



for la=(1:division)

   Rad_TKE(ly,1)=Avg_bottom(la,lw)/(-min(Avg_bottom));  % r/R
   Rad_TKE(ly,2)=TKE(la,lw);
   Rad_TKE(ly+1,1)=Avg_top(la,lw)/(max(Avg_top));  % r/R
   Rad_TKE(ly+1,2)=TKE(la,lw);
   Rad_v_r_flx_avg(jl,2)=v_r_flx1_avg(la,lw);  % v_r'
   Rad_v_theta_flx_avg(jl,2)=v_theta_flx1_avg(la,lw);  % v_theta'
   Rad_v_z_flx_avg(jl,2)=v_z_flx1_avg(la,lw);  %v_z'
   Rad_v_flx(jl,1)=Avg_top(la,lw)/(max(Avg_top));  % Radius
```

```
   Rad_traj(jl,1)=htraj(la,lw);  %Number of trajectory per conentric
circle
   ly=ly+2;
   jl=jl+1;

end


plot(Rad_TKE(:,2),Rad_TKE(:,1),'.')
xlabel('TKE (m^2/s^2)')
ylabel('r/R')
title('TKE throughout the pipe')
axis([0.001,0.017,-1,1]);

savefig('tke_steady_flow_15div');

figure

subplot(2,1,1)
plot(Rad_v_flx(:,1),Rad_v_r_flx_avg(:,2),'--*')
hold on
plot(Rad_v_flx(:,1),Rad_v_theta_flx_avg(:,2),'--*')
hold on
plot(Rad_v_flx(:,1),Rad_v_z_flx_avg(:,2),'--*')
axis([0,1,-0.09,0.09]);
xlabel('r/R')
ylabel('Velocity Fluctuations (m/s)')
legend('v_r flx','v_t flx','v_z flx')
title('TKE throughout the pipe')

subplot(2,1,2)
plot(Rad_v_flx(:,1),Rad_traj(:,1),'--o')
title('Number of trajectories as function of radial location')
xlabel('r/R')
ylabel('Number of Trajectories')
axis([0,1,0,300]);

savefig('vel_flx_num_traj_steady_flow_15div');

close(waitbar3)

else
end




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
```

```
%
%
%                              Diffusivity
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%


% Prompt_4=input('Would you like to assess the diffusivity (Y/N)?','s')

if Prompt_4=='Y'


% User Input for region of interest and grid size %%

% prompt_x_position=input('Enter the X position of interest(mm):');
% User input X position of interest (mm)
x_position=prompt_x_position;

% prompt_y_position=input('Enter the Y position of interest(mm):');
% User input Y position of interest (mm)
y_position=prompt_y_position;

% prompt_z_position=input('Enter the Z position of interest (mm):');
% User input Z position of interest (mm)
z_position=prompt_z_position;

% prompt_Xgrid_size=input('Enter the X grid size (mm):');
% User input grid size (mm)
Xgrid_size=prompt_Xgrid_size;

% prompt_Ygrid_size=input('Enter the Y grid size (mm):');
% User input grid size (mm)
Ygrid_size=prompt_Ygrid_size;

% prompt_Zgrid_size=input('Enter the Z grid size (mm):');
% User input grid size (mm)
Zgrid_size=prompt_Zgrid_size;


% % Lower and Upper limits for each dimension %%

xll=(x_position)-(Xgrid_size);
% X position lower limit (mm)
xhl=(x_position)+(Xgrid_size);
% X position higher limit (mm)
yll=(y_position)-(Ygrid_size);
% Y position lower limit (mm)
```

```
yhl=(y_position)+(Ygrid_size);
% Y position higher limit (mm)
zll=(z_position)-(Zgrid_size);
% Z position lower limit (mm)
zhl=(z_position)+(Zgrid_size);
% Z position higher limit (mm)
ll=0;
% Increment
aa=1;
% Increment
db=1;
% Increment
xx=2;
% Increment
dbb=0;
% Increment
gg=0;
% Increment

% prompt_Z_Traj_limit=input('Enter the minimum trajectory length
(mm):');              % User input the minimum trajectory length
Z_Trajectory_Required=prompt_Z_Traj_limit;


% Plotting the rectangle around the region of interest %%


figure

% 8 corner point of the rectangle

b1=[zll,xll,yll];
b2=[zhl,xll,yll];
b3=[zhl,xhl,yll];
b4=[zll,xhl,yll];
b5=[zll,xll,yhl];
b6=[zhl,xll,yhl];
b7=[zhl,xhl,yhl];
b8=[zll,xhl,yhl];

plot3([zll zhl zhl zll],[xll xll xhl xhl],[yll yll yll yll],'-k')
% Bottom
hold on

plot3([zll zhl zhl zll],[xll xll xhl xhl],[yhl yhl yhl yhl],'-k')
% Top
plot3([zhl zhl zhl zhl],[xll xll xhl xhl],[yll yhl yhl yll],'-k')
% Right
plot3([zll zll zll zll],[xll xhl xhl xll],[yll yll yhl yhl],'-k')
% Left
plot3([zll zll zhl zhl],[xll xll xll xll],[yll yhl yhl yll],'-k')
% Front
```

```
plot3([zll zhl zhl zll],[xhl xhl xhl xhl],[yll yll yhl yhl],'-k')
% Back

axis([0 127 -20 20 -20 20])


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Getting the trajectories that pass through the region of interest
%
% Storing its information in Matrix called "gData"
%
% Getting maximum x,y,z position for each trajectory and storing its
%
% information in Matrix called "maxgData"
%
% User inputs the z position threshold and trajectories are sorted to
%
% meet that requirement and they are plotted and histogram is created
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

waitbar4=waitbar(0,'Please Wait Diffusivity 4/5....');

for grid=(1:length(Data))

    if (Data(grid,4)>=yll) && (Data(grid,4)<=yhl) &&
(Data(grid,6)>=zll)...
            && (Data(grid,6)<=zhl) && (Data(grid,3)>=xll)...
% If statement ensures that all the trajectories are with in the lower
and upper limits
                                    && (Data(grid,3)<=xhl)
        Traj(xx,1)=(Data(grid,1));
% Trajectory matrix


        if (Traj(xx,1))>(Traj((xx-1),1))
% If statement ensures that different trajectory goes through
        for m=(grid:length(Data))
            if (Data(m,1))==Traj(xx,1)
                gt(aa,1)=(Data(m,1));
                gData(ll+aa,1)=(Data(m,1));
% Trajectory numbers stored in Matrix called "gData"
                gta(aa,1)=(Data(m,2));
                gData(ll+aa,2)=(Data(m,2));
% Adjusted time stored in Matrix called "gData"
                xgp(aa,1)=(Data(m,3));
                gData(ll+aa,3)=(Data(m,3));
% X grid position stored in Matrix called "gData"
                ygp(aa,1)=(Data(m,4));
                gData(ll+aa,4)=(Data(m,4));
% Y grid position stored in Matrix called "gData"
```

```
                zgp(aa,1)=(Data(m,6));
                gData(ll+aa,5)=(Data(m,6));
% Z grid position stored in Matrix called "gData"
                plot3(zgp,xgp,ygp)
% plotting Z grid position as function Y grid position
                hold on
                axis([0 127 -20 20 -20 20])
                title('Particle Trajectories in 3D')
                xlabel('Z position (mm)')
                ylabel('X position (mm)')
                zlabel('Y position (mm)')

                aa=aa+1;
% Increment
            end

        end

        maxgData(db,1)=Traj(xx,1);
% Storing trajectory number in Matrix called "maxData"
        maxgData(db,2)=max(xgp);
% Getting maximum X grid position within the same trajectory (mm)
        maxgData(db,3)=max(ygp);
% Getting maximum Y grid position within the same trajectory (mm)
        maxgData(db,4)=max(zgp);
% Getting maximum Z grid position within the same trajectory (mm)
        mingData(db,1)=Traj(xx,1);
% Storing trajectory number in Matrix called "minData"
        mingData(db,2)=min(xgp);
% Getting minimum X grid position within the same trajectory (mm)
        mingData(db,3)=min(ygp);
% Getting minimum Y grid position within the same trajectory (mm)
        mingData(db,4)=min(zgp);
% Getting minimum Z grid position within the same trajectory (mm)


        Length_Z_Trajectory(db,1)=(maxgData(db,4))-(mingData(db,4));
% Calculating the length of the trajectory

        for zz=(1:length(zgp))
        if Length_Z_Trajectory(db,1)>=Z_Trajectory_Required
            fgData(dbb+1,1)=gt(zz,1);
% Filtered trajectory
            fgData(dbb+1,2)=gta(zz,1);
% Filtered adjusted time
            fgData(dbb+1,3)=xgp(zz,1);
% Filtered X position
            fgData(dbb+1,4)=ygp(zz,1);
% Filtered Y position
            fgData(dbb+1,5)=zgp(zz,1);
% Filtered Z position
```

```
            dbb=length(fgData(:,1));
% Increment

        end

        end


        xx=xx+1;
% Increment
        ll=length(gData);
% Increment
        aa=1;
% Increment
        db=db+1;
% Increment
        gt(:,:)=[];
% clearing trajectory number
        gta(:,:)=[];
% clearing adjusted grid time to plot all the trajectories separately
        xgp(:,:)=[];
% clearing X grid position to plot all the trajectories separately
        ygp(:,:)=[];
% clearing Y grid position to plot all the trajectories separately
        zgp(:,:)=[];
% clearing Z grid position to plot all the trajectories separately

        end
    end

    waitbar(grid/length(Data));
end

close(waitbar4);

min_maxxgp=min(maxgData(:,2));
% minimum X grid position of local maximum x grid position (mm)
min_maxygp=min(maxgData(:,3));
% minimum Y grid position of local maximum y grid position (mm)
min_maxzgp=min(maxgData(:,4));
% minimum Z grid position of local maximum z grid position (mm)


g=1;
% Increment
dii=1;
% Increment
daa=2;
% Increment
l=1;
% Increment
```

```
yy=1;
% Increment
ww=1;
% Increment
dm=1;
% Increment

% prompt_position=input('How many positions would you like to
analyze?:');              % User input the number of analysis to be
conducted
position=prompt_position;

waitbar5=waitbar(0,'Please Wait Diffusivity 5/5....');

for mm=1:(position)

% prompt5='Enter Z position where you want to evaluate the distribution
(mm):';         % User input the Z position threshold of interest
prompt_z=(prompt5);

zz=prompt_z;


for i=(1:length(fgData))
% Sorting the grid data to meet the threshold limit

    if fgData(i,5)<=zz
        hgData(g,:)=fgData(i,:);
% Storing the trajectory information that meets the threshold limits
        g=g+1;
% Increment

    end
end



figure; hold on
subplot(2,2,1.5)
% Plotting a rectangle around the region of interest

plot3([zll zhl zhl zll],[xll xll xhl xhl],[yll yll yll yll],'-k')
% Bottom
hold on

plot3([zll zhl zhl zll],[xll xll xhl xhl],[yhl yhl yhl yhl],'-k')
% Top
plot3([zhl zhl zhl zhl],[xll xll xhl xhl],[yll yhl yhl yll],'-k')
% Right
plot3([zll zll zll zll],[xll xhl xhl xll],[yll yll yhl yhl],'-k')
% Left
```

```
plot3([zll zll zhl zhl],[xll xll xll xll],[yll yhl yhl yll],'-k')
% Front
plot3([zll zhl zhl zll],[xhl xhl xhl xhl],[yll yll yhl yhl],'-k')
% Back


hold on

for do=1
for h=(1:(length(hgData)))
% plotting each trajectory that meets the threshold limit

    if h==1
        hta(l,1)=hgData(h,2);
% Adjusted Time (ms)
        hxgp(l,1)=hgData(h,3);
% X position (mm)
        hygp(l,1)=hgData(h,4);
% Y position (mm)
        hzgp(l,1)=hgData(h,5);
% Z position (mm)
        l=l+1;
% Increment

    else

    if hgData(h,1)==hgData((h-1),1)

        hta(l,1)=hgData(h,2);
% Adjusted Time (ms)
        hxgp(l,1)=hgData(h,3);
% X position (mm)
        hygp(l,1)=hgData(h,4);
% Y position (mm)
        hzgp(l,1)=hgData(h,5);
% Z position (mm)



        plot3(hzgp,hxgp,hygp)
% plotting the trajectories from the region of interest to analysis
threshold
        axis([0 127 -20 20 -20 20])
        title(['Particle Positions in 3D at Z Position of ' num2str(zz)
'mm'])
        xlabel('Z position (mm)')
        ylabel('X position (mm)')
        zlabel('Y position (mm)')

        l=l+1;
% Increment
        max_hz(yy,1)=max(hzgp(:,1));
% local maximum Z grid position
```

```
        % Plotting diffusivity %

        diffusivity_mat(dm,1)=hta(1,1);  % Initial Time (ms)
        diffusivity_mat(dm,2)=hta(length(hta),1);  % Final Time (ms)
        diffusivity_mat(dm,3)=hxgp(1,1);  % Initial x position (mm)
        diffusivity_mat(dm,4)=hxgp(length(hta),1);  % Final x position
(mm)
        diffusivity_mat(dm,5)=hygp(1,1);  % Initial y position (mm)
        diffusivity_mat(dm,6)=hygp(length(hta),1);  % Final y position
(mm)
        diffusivity_mat(dm,7)=hzgp(1,1);  % Initial z position (mm)
        diffusivity_mat(dm,8)=hzgp(length(hta),1);  % Final z position
(mm)



    else

         for v=(1:length(hzgp))
% Getting the corresponding Y grid position of maximum Z grid position
             if hzgp(v,1)==max_hz(yy,1)
                 max_hy(ww,1)=hygp(v,1);
                 max_hx(ww,1)=hxgp(v,1);
                 ww=ww+1;
             end
         end



        dm=dm+1;


        hta(:,:)=[];
% Clearing the adjusted time
        hxgp(:,:)=[];
% Clearing the X position
        hygp(:,:)=[];
% Clearing the Y position
        hzgp(:,:)=[];
% Clearing the Z position
        l=1;
% Increment
        yy=yy+1;
% Increment

    end
    end
```

```
end

         for v=(1:length(hzgp))
% Getting the corresponding Y grid position of maximum Z grid position
             if hzgp(v,1)==max_hz(yy,1)
                 max_hy(ww,1)=hygp(v,1);
                 max_hx(ww,1)=hxgp(v,1);
                 ww=ww+1;
             end
         end

end




        subplot(2,2,4)
        [~,~]=hist(max_hy,5);
% plotting histogram of Y position dispersion
        histfit(max_hy,5,'normal')
        title(['Distribution of the Particles at Z Position:'
num2str(zz)    'mm'])
        xlabel('Y position (mm)')
        ylabel('Frequency')

        subplot(2,1,1)
        [f_y,center_y]=hist(max_hy,5);
% plotting histogram of Y position dispersion
        histfit(max_hy,5,'normal')
        title(['Distribution of the Particles at Z Position:'
num2str(zz)    'mm'])
        xlabel('Y position (mm)')
        ylabel('Frequency')

        subplot(2,2,3)
        [~,~]=hist(max_hx,5);
% plotting histogram of X position dispersion
        histfit(max_hx,5,'normal')
        title(['Distribution of the Particles at Z Position:'
num2str(zz)    'mm'])
        xlabel('X position (mm)')
        ylabel('Frequency')
        hold all

        subplot(2,1,2)
        [f_x,center_x]=hist(max_hx,5);
% plotting histogram of X position dispersion
        histfit(max_hx,5,'normal')
        title(['Distribution of the Particles at Z Position:'
num2str(zz)    'mm'])
        xlabel('X position (mm)')
        ylabel('Frequency')
```

```
        hold all


        %% Mean and Standard Deviation

        vv=1;
        for v=(1:length(f_y))
% Calculating the mean x position and y position
            mul_y(vv,1)=(f_y(1,v))*(center_y(1,v));
            mul_x(vv,1)=(f_x(1,v))*(center_x(1,v));
            vv=vv+1;
        end

        mean_y=(sum(mul_y(:,1)))/(sum(f_y));
        mean_x=(sum(mul_x(:,1)))/(sum(f_x));

        qq=1;
        for q=(1:length(max_hx))
% Calculating the standard deviation in x and y
            std_y(qq,1)=((max_hy(q,1))-(mean_y))^(2);
            std_x(qq,1)=((max_hx(q,1))-(mean_x))^(2);
            qq=qq+1;
        end

        std_y=sqrt((sum(std_y(:,1)))/(sum(f_y)));
        std_x=sqrt((sum(std_x(:,1)))/(sum(f_x)));
        sum_f_x=sum(f_x);
        sum_f_y=sum(f_y);

%         [mean,std]=normfit(max_hy)
%         pdf=normpdf(f,mean,std)
%         [hh,ww]=chi2gof(pdf,'alpha',0.05)


        hold on
        axes('Position',[0.8 0 1 1],'Visible','off')
        string={['\bf Stats'];['\rm f_{x}='
num2str(sum_f_x,'%0.2f')];...                                          %
Placing the stats on the figure
         [' f_{y}=' num2str(sum_f_y,'%0.2f')]; [' \mu_{x}='
num2str(mean_x,'%0.2f')];...
         [' \mu_{y}=' num2str(mean_y,'%0.2f')]; [' \sigma_{x}='
num2str(std_x,'%0.2f')];...
         [' \sigma_{y}=' num2str(std_y,'%0.2f')]; };

        text(0.105,0.75,string);


        hgData(:,:)=[];
% Clearing the "hgData" matrix
        hta(:,:)=[];
% Clearing the adjusted time
```

```
        hxgp(:,:)=[];
% Clearing the X position
        hygp(:,:)=[];
% Clearing the Y position
        hzgp(:,:)=[];
% Clearing the Z position
        max_hz(:,:)=[];
% Clearing the maximum Z position
        max_hy(:,:)=[];
% Clearing the maximum Y position
        max_hx(:,:)=[];
% Clearing the maximum X position
        g=1;
% Increment
        dii=1;
% Increment
        daa=2;
% Increment
        l=1;
% Increment
        yy=1;
% Increment
        ww=1;
% Increment

 waitbar(mm/position);

end

close(waitbar5);



dma=1;
figure;

    subplot(3,1,1)
    plot(diffusivity_mat(:,3),diffusivity_mat(:,4),'.');
    xlabel('Initial x position (mm)')
    ylabel('Final x position(mm)')
    title('Diffusivity in x direction')

    subplot(3,1,2)
    plot(diffusivity_mat(:,5),diffusivity_mat(:,6),'.');
    xlabel('Initial y position (mm)')
    ylabel('Final y position(mm)')
    title('Diffusivity in y direction')

    subplot(3,1,3)
    plot(diffusivity_mat(:,7),diffusivity_mat(:,8),'.');
    xlabel('Initial z position (mm)')
    ylabel('Final z position(mm)')
```

```
    title('Diffusivity in z direction')

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%                          Binned velocity profile
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

if Prompt_5=='Y'

sort_Modata=sortrows(Modata(:,:),7);  % Sort the data from lowest to
highest radius

h=figure;  % Figure
radiv=radius_div;  % Number of slices on for the velocity profile
minradinc=min(sort_Modata(:,7));  % Minimum Radius
maxradinc=max(sort_Modata(:,7));  % Maximum Radius
radinc=(maxradinc-minradinc)/radiv;  % Average radius slice
upperradinc=minradinc+radinc;  % Upper Radius Limit
rad=1;  % Increment
rt=1;  % Increment
valavg=1;  % Increment
op=1;  % Increment

        while rt<length(Modata(:,1))

            if sort_Modata(rt,7)>=(minradinc) &&
sort_Modata(rt,7)<=(upperradinc)  % Consider only if radius is in-
between the limits
                radData_r(rad,1)=sort_Modata(rt,7);  % Qualified radius
                radData_vz(rad,1)=sort_Modata(rt,10);  % Qualified
Velocity in Z
                rad=rad+1;  % Increment
                rt=rt+1;  % Increment

            else

                if rad==1;  % If their is only one point, than skp it.
                    avgData_vzr(valavg,1)=avgData_vzr(valavg-1,1);  %
Radius
                    avgData_vzr(valavg,2)=avgData_vzr(valavg-1,2);  %
Velocity in Z
                    rad=1;  % Increment
                    valavg=valavg+1;  % Increment
                    minradinc=upperradinc;  % Lower Radius Limit
                    upperradinc=upperradinc+radinc;  % Upper Radius
Limit
                    radData_r=[];  % Clearing radius to ensure only one
trajectory is considered at a time
```

```
                    radData_vz=[];  % Clearing velocity in z direction
to ensure only one trajectory is considered at a time
                else

                    avgData_vzr(valavg,1)=sum(radData_vz(:,1))/rad;  %
Average Velocity in z direction
                    avgData_vzr(valavg,2)=sum(radData_r(:,1))/rad;  %
Average Radius
                    rad=1;  % Increment
                    valavg=valavg+1;  % Increment
                    minradinc=upperradinc;  % Lower Radius Limit
                    upperradinc=upperradinc+radinc;  % Upper Radius
Limit
                    radData_r=[];  % Clearing radius to ensure only one
trajectory is considered at a time.
                    radData_vz=[];  % Clearing velocity in z direction
to ensure only one trajectory is considered at a time.
                end

            end
        end

        % Plotting all the average velocity in z direction and average
        % radius

        plot(avgData_vzr(:,1),avgData_vzr(:,2),'.','MarkerSize',10)
        title('Velocity Profile')
        xlabel('Axial Velocity (m/s)')
        ylabel('Radius (mm)')

        hold on

%% 1/7th power law for velocity profile

vmax=max(avgData_vzr(:,1));  % Maximum Velocity
r=transpose(linspace(lower_diameter_val,upper_diameter_val,divisions));
% Dividing the pipe radius

for ana=(1:divisions)
 v(ana,1)= vmax*(1-(abs(r(ana,1))/(radius_val)))^(1/7);   % 1/7th law
equation
end

plot(v(:,1),r(:,1),'.','Markersize',10);
legend('Measured','Analytical (1/7th Law)')

end
```

Appendix B – Time Dependent TKE Calculation

```
clc
close all
clear all

Trajectories=1000;
%Trajectories
lastlength=0;
%Initializing the matrix dimension
length_prv=0;
% Data_prv=ones(500000,3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Reading each position files and performing various calculations
%
% Reading the files and filtering trajectory files with less
%
% than 4 data points.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%


waitbar1=waitbar(0,'Please Wait 1/3 ....');
for avg=1:Trajectories

File_name=sprintf('FilteredPosition%d.txt',avg);
% Filename
read_file_prv=dlmread(File_name,'\t',2,0);
% Reading the file

[m,n]=size((read_file_prv));

if m>4

for avg2=(1:m)
Data_prv((length_prv+avg2),1)=(read_file_prv(avg2,2));
% x-position
Data_prv((length_prv+avg2),2)=(read_file_prv(avg2,3));
% y-position
Data_prv((length_prv+avg2),3)=(read_file_prv(avg2,4));
% z-position
Data_prv_A(length_prv+avg2,1)=avg;
% Trajectory increment

end
end

length_prv=length(Data_prv_A);
```

```
waitbar(avg/Trajectories);

end

close(waitbar1)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Centering the pipe by subtracting the data set by average position
%
% Average is calculated by (max+min)/2. Radius and Angle are calculated
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

mean_x=(max(Data_prv(:,1))+min(Data_prv(:,1)))/2;
% Average velocity in x direction
mean_y=(max(Data_prv(:,2))+min(Data_prv(:,2)))/2;
% Average velocity in y direction
mean_z=(max(Data_prv(:,3))+min(Data_prv(:,3)))/2;
% Average velocity in z direction
Data=0;
% Data=ones(500000,7);
last_length=0;
ii=0;
hh=1;
traj=0;



for T=1:Trajectories

File_name=sprintf('FilteredPosition%d.txt',T);
% Filename
read_file=dlmread(File_name,'\t',2,0);
% Reading the file

[m,n]=size((read_file));

% if prompt0=='Y'
%
% figure(1)
%
% plot3((127-(read_file(:,4))),(read_file(:,2)-mean_x),(read_file(:,3)-
mean_y))                                % plotting the z vs. y
% xlabel('Z position (mm)')
% ylabel('X position (mm)')
% zlabel('Y position (mm)')
%
%
% hold on
%
```

```
% end

if m>4

traj=traj+1;

for a=(1:m)

Data(last_length+a,1)=T;
% trajectory
% time_adj(a,1)=((read_file(a,1))-(read_file(1,1)));
% time adjustment to ensure all the position files start at t=0 ms
time_adj(a,1)=(read_file(a,1));
Data((last_length+a),2)=time_adj(a,1);
% adjusted time (ms)
Data(last_length+a,3)=(read_file(a,2)-mean_x);
% x position (mm)
Data(last_length+a,4)=(read_file(a,3)-mean_y);
% y position (mm)
Data(last_length+a,5)=sqrt(((read_file(a,2)-
mean_x)^(2))+((read_file(a,3)-mean_y)^(2)));  % Radius (mm)
Data(last_length+a,6)=(127-(read_file(a,4)));
% z position (mm)
Data(last_length+a,7)=rad2deg(atan((read_file(a,3)-
mean_y)/(read_file(a,2)-mean_x)));     % angle (degree)

    if Data(last_length+a,7)<0
        Data(last_length+a,5)=-Data(last_length+a,5);
    else
        Data(last_length+a,5)=Data(last_length+a,5);
    end


Data_A(last_length+a,1)=T;

% title={'Trajectory' 'Adjusted Time (s)' 'X Position (mm)' 'Y Position
(mm)' 'Radius (mm)' 'Z Position (mm)' 'Angle'};
% Data_Matrix=[title;num2cell(Data(:,:))];

end

% Matrix array increments %%

last_length=length(Data_A);
ii=ii+1;
hh=1;

end

end
```

```
% Reading in trigger time marker file

triggerT=dlmread('triggerTime.txt','\t',1,0);
trigger(1)=triggerT(1,1);

%Add mid trigger since each period consist of 2 pulse cycle

xyz=2;  % Initial value

for trig=1:length(triggerT)-1


midtrigger=(triggerT(trig)+triggerT(trig+1))/2;  % Getting mid trigger
since 2 pulse per cycle
trigger(xyz,1)=midtrigger;  % Mid trigger
trigger(xyz+1,1)=triggerT(trig+1);  % Creating matrix with mid trigger

xyz=xyz+2;  % Increments

end

scantime=0;  % Initial value
nPeriod=0;  % Initial value
frames=20;  % Number of divisions of each pulse cycle

for inc=(1:length(trigger)-1)
    scantime=scantime+(trigger(inc+1)-trigger(inc));  % Dividing actual
time into each cycle
    nPeriod=nPeriod+1;  % Increments
end

Period=scantime/nPeriod;  % Period
timeslice=Period/frames;  % time per frame
ts=timeslice;             % time per frame


xv=1;  % Initial value
wx=0;  % Initial value
bin=1;  % Initial value
prvscantime=0;  % Initial value
newscantime=(trigger(xv+1)-trigger(xv))/frames;  % Diving each cycles
into  desired divisions
scantimeinc=newscantime;
hj=1;  % Initial value
wxy=1;  % Initial value


sortData=sortrows(Data(:,:),2);  % Sorting the Data matrix from lowest
to highest time

% Taking out the initial steady flow data
```

```
for lk=1:length(Data(:,1))
 if (sortData(lk,2)<trigger(1))
     hj=hj+1;
     wxy=wxy+1;
 end
end

% Binning the data based on each pulse cycle divisions

while hj<=length(Data(:,1))

    if (sortData(hj,2)<=(trigger(xv)+scantimeinc) &&
sortData(hj,2)>=(trigger(xv)+prvscantime))
        bin=1+wx;  % Bin number
        sortData(hj,8)=bin;
        hj=hj+1;  %Increment
    else
        wx=wx+1;  % Increment
        bin=1+wx;
        prvscantime=scantimeinc;  % previous scan time
        scantimeinc=newscantime*bin;  % new scan time
          if bin>frames
             xv=xv+1;
             newscantime=(trigger(xv+1)-trigger(xv))/frames;
             wx=0;
             prvscantime=0;
             scantimeinc=newscantime;
          end
    end


end

sorted_frame_Data=sortrows(sortData(:,:),8);  % Sorting data based on
lowest to highest bin number

% Assigning random numbers with additional frame number to ensure
% we have the data for max number of frames and not max number - 1


sorted_frame_Data(length(Data(:,1)),1)=500;
sorted_frame_Data(length(Data(:,1)),2)=max(sorted_frame_Data(:,2))+1;
sorted_frame_Data(length(Data(:,1)),3)=1;
sorted_frame_Data(length(Data(:,1)),4)=1;
sorted_frame_Data(length(Data(:,1)),5)=1;
sorted_frame_Data(length(Data(:,1)),6)=1;
sorted_frame_Data(length(Data(:,1)),7)=1;
sorted_frame_Data(length(Data(:,1)),8)=frames+1;

% Increments
```

```
wq=wxy;
op=1;
cframe=1;
oi=1;
xh=1;
wh=1;
ty=1;
xz=1;
ta=1;
prv_ta=0;
ga=1;
yz=1;
traj=1;

while wq<length(Data(:,1))+1

    if sorted_frame_Data(wq,8)==cframe

        Mosort_Data(op,1)=sorted_frame_Data(wq,1);  % Traj
        Mosort_Data(op,2)=sorted_frame_Data(wq,2);  % Time (seconds)
        Mosort_Data(op,3)=sorted_frame_Data(wq,3);  % X-val
        Mosort_Data(op,4)=sorted_frame_Data(wq,4);  % Y-val
        Mosort_Data(op,5)=sorted_frame_Data(wq,5);  % Radius
        Mosort_Data(op,6)=sorted_frame_Data(wq,6);  % Z-val
        Mosort_Data(op,7)=sorted_frame_Data(wq,7);  % Theta
        Mosort_Data(op,8)=sorted_frame_Data(wq,8);  % bins (aka:
frames)


        wq=wq+1;  % Increments
        op=op+1;  % Increments



    else

        % Taking all the data and separating it by trajectory number %

        sort_traj_Data=sortrows(Mosort_Data(:,:),1);

        % Adding random trajectory to ensure last trajectory is counted


sort_traj_Data(length(sort_traj_Data(:,1))+1,1)=max(sort_traj_Data(:,1)
)+1;

        while oi<(length(sort_traj_Data(:,1)))

            if sort_traj_Data(oi,1)==sort_traj_Data(oi+1,1)
```

```
                u_t(xz,1)=(sort_traj_Data(oi+1,3)-
sort_traj_Data(oi,3))/(sort_traj_Data(oi+1,2)-(sort_traj_Data(oi,2)));
%vx
                v_t(xz,1)=(sort_traj_Data(oi+1,4)-
sort_traj_Data(oi,4))/(sort_traj_Data(oi+1,2)-(sort_traj_Data(oi,2)));
%vy
                w_t(xz,1)=(sort_traj_Data(oi+1,6)-
sort_traj_Data(oi,6))/(sort_traj_Data(oi+1,2)-(sort_traj_Data(oi,2)));
%vz

vr(xz,1)=(u_t(xz,1))*cosd(sort_traj_Data(oi,1))+(v_t(xz,1))*sind(sort_t
raj_Data(oi,1));                    %v_r
                vtheta(xz,1)=-
(u_t(xz,1))*sind(sort_traj_Data(oi,1))+(v_t(xz,1))*cosd(sort_traj_Data(
oi,1));               %v_theta

                    Modata(xh,1)=sort_traj_Data(oi+1,1);  % traj
                    Modata(xh,2)=sort_traj_Data(oi+1,2);  % Time
                    Modata(xh,3)=sort_traj_Data(oi+1,3);  % X-val
                    Modata(xh,4)=sort_traj_Data(oi+1,4);  % Y-val
                    Modata(xh,5)=sort_traj_Data(oi+1,5);  % radius
                    Modata(xh,6)=sort_traj_Data(oi+1,6);  % Z-val
                    Modata(xh,7)=sort_traj_Data(oi+1,7);  % Theta
                    Modata(xh,8)=sort_traj_Data(oi+1,8);  % Bins
                    Modata(xh,9)=u_t(xz,1);  %Vx
                    Modata(xh,10)=v_t(xz,1); %Vy
                    Modata(xh,11)=w_t(xz,1); %Vz
                    Modata(xh,12)=vr(xz,1);  %Vr
                    Modata(xh,13)=vtheta(xz,1);  %Vtheta

                    xh=xh+1;  % Increment
                    oi=oi+1;  % Increment
                    xz=xz+1;  % Increment

                else

                    if xz==1


                        oi=oi+1;  % Increments
                        xz=1;  % Increments
                        u_t(:,:)=[];   % clearing u velocity
                        v_t(:,:)=[];  % clearing v velocity
                        w_t(:,:)=[];  % clearing w velocity
                        vr(:,:)=[];  % clearing vr velocity
                        vtheta(:,:)=[];  % clearing vtheta velocity

                        continue

                    else

                    avg_vr(ty,1)=sum(vr(:,1))/(xz-1);  % average vr
```

```
                avg_vtheta(ty,1)=sum(vtheta(:,1))/(xz-1);  % average
vtheta
                avg_vz(ty,1)=sum(w_t(:,1))/(xz-1);  % average vz

                for ta=(1:(xz-1))


                        Modata(ta+prv_ta,14)=avg_vr(ty,1);  % average
vr
                        Modata(ta+prv_ta,15)=avg_vtheta(ty,1);  %
average vtheta
                        Modata(ta+prv_ta,16)=avg_vz(ty,1);  % average
vz
                        Modata(ta+prv_ta,17)=Modata(ta+prv_ta,12)-
Modata(ta+prv_ta,14);  % vr flux
                        Mosort_Data(ga,9)=Modata(ta+prv_ta,17);  % vr
flux
                        Modata(ta+prv_ta,18)=Modata(ta+prv_ta,13)-
Modata(ta+prv_ta,15);  % vtheta flux
                        Mosort_Data(ga,10)=Modata(ta+prv_ta,18);  %
vtheta flux
                        Modata(ta+prv_ta,19)=Modata(ta+prv_ta,11)-
Modata(ta+prv_ta,16);  % vz flux
                        Mosort_Data(ga,11)=Modata(ta+prv_ta,19);  %vz
flux
                        ga=ga+1;

                end

                prv_ta=prv_ta+ta;  % Increment
                ty=ty+1;  % Increment
                oi=oi+1;  % Increment
                xz=1;  % Increment
                u_t(:,:)=[];
                v_t(:,:)=[];
                w_t(:,:)=[];
                vr(:,:)=[];
                vtheta(:,:)=[];

                end

            end

        end

        figure
        subplot(3,1,1)
        plot(Modata(:,5),Modata(:,17),'.')
        title('vr flx vs. radius')
        xlabel('Radius(mm)')
        ylabel('vr flx(m/s)')
```

```
        subplot(3,1,2)
        plot(Modata(:,5),Modata(:,18),'.')
        title('vtheta flx vs. radius')
        xlabel('Radius (mm)')
        ylabel('vtheta flx (m/s)')

        subplot(3,1,3)
        plot(Modata(:,5),Modata(:,19),'.')
        title('vz flx vs. radius')
        xlabel('Radius (mm)')
        ylabel('vz flx (m/s)')

        diameter=max(Mosort_Data(:,5));
% diameter (mm)
        division=15;
% division
        steps=(diameter/division);
% step size (mm)
        y=1;  % Increment
        Bottom_low=-max(Mosort_Data(:,5));
% Initial Lower limit (mm)
        Bottom_high=-max(Mosort_Data(:,5))+steps;
% Initial Upper limit (mm)
        Top_low=max(Mosort_Data(:,5))-steps;
        Top_high=max(Mosort_Data(:,5));

        o=1;  % Increment
        oo=1;  % Increment

        % Dividing pipe into different slices

        for z=(1:(division))



        Bottom_low_step_limit=Bottom_low;  % Bottom lower limit
        Bottom_high_step_limit=Bottom_high;  % Bottom upper limit
        Top_low_step_limit=Top_low;  % Top lower limit
        Top_high_step_limit=Top_high;  % Top upper limit

        %% Dividing the profile

        for x=(1:length(Mosort_Data))

            if Mosort_Data(x,5)>=Bottom_low_step_limit &&
Mosort_Data(x,5)<Bottom_high_step_limit...         % Logic states to
include all the parameters within
                    || Mosort_Data(x,5)>=Top_low_step_limit &&
Mosort_Data(x,5)<Top_high_step_limit          % the radius limits based
on the divisions

                traj1(y,1)=Mosort_Data(x,1);  % Trajectory
```

```
                if y>=2
                if traj1(y-1,1)~=traj1(y,1)
                    num_traj=traj+1;
                    traj=num_traj;
                end
                end
                time1(y,1)=Mosort_Data(x,2);  % Time (ms)
                radius1(y,1)=Mosort_Data(x,5);  % Radius (mm)
                theta1(y,1)=Mosort_Data(x,7);  % Theta (angle)
                v_r_flx1(y,1)=Mosort_Data(x,9);  % Vr fluctuations
                v_r_flx1_square(y,1)=(Mosort_Data(x,9))^(2);  % Vr^2
fluctuations
                v_theta_flx1(y,1)=Mosort_Data(x,10);  % Vtheta
fluctuations
                v_theta_flx1_square(y,1)=(Mosort_Data(x,10))^(2);  %
Vtheta^2 fluctuations
                w_flx1(y,1)=Mosort_Data(x,11);  % Vz fluctuations
                w_flx1_square(y,1)=(Mosort_Data(x,11))^(2);  % Vz^2
fluctuations
                y=y+1;  % Incrementor

            end

        end

        % Turbulent Kinetic Energy Assessment of each division


        Mean_v_r_flx1_square(z,1)=(sum(v_r_flx1_square(:,1)))/(y);  %
Average Vr fluctuations

Mean_v_theta_flx1_square(z,1)=(sum(v_theta_flx1_square(:,1)))/(y);  %
Average Vtheta fluctuations
        Mean_w_flx1_square(z,1)=(sum(w_flx1_square(:,1)))/(y);  %
Average Vz fluctuations

TKE(z,yz)=(1/2)*((Mean_v_r_flx1_square(z,1))+(Mean_v_theta_flx1_square(
z,1))...
                +(Mean_w_flx1_square(z,1)));  % Turbulent Kinetic
Energy

Avg_bottom(z,yz)=((Bottom_high_step_limit)+(Bottom_low_step_limit))/2;
% Increment
        Avg_top(z,yz)=((Top_high_step_limit)+(Top_low_step_limit))/2;
% Increment
        v_r_flx1_avg(z,yz)=(sum(v_r_flx1(:,1)))/(y);  %Vr fluctuations
        v_theta_flx1_avg(z,yz)=(sum(v_theta_flx1(:,1)))/(y);
% Vtheta fluctuations
        v_z_flx1_avg(z,yz)=(sum(w_flx1(:,1)))/(y);  % Vz fluctuations
        htraj(z,yz)=num_traj;  % number of trajectories

        % clearing individual trajectory information for plotting
```

```
        traj=1;
        y=1;
        traj1(:,:)=[];
        time1(:,:)=[];
        radius1(:,:)=[];
        theta1(:,:)=[];
        v_r_flx1(:,:)=[];
        v_r_flx1_square(:,:)=[];
        v_theta_flx1(:,:)=[];
        v_theta_flx1_square(:,:)=[];
        w_flx1(:,:)=[];
        w_flx1_square(:,:)=[];
        Bottom_low=Bottom_high_step_limit;
        Bottom_high=Bottom_high_step_limit+steps;
        Top_low=Top_high_step_limit;
        Top_high=Top_low_step_limit;
        Top_low=Top_low_step_limit-steps;

        end
            yz=yz+1;  % Increment
            ga=1;  % Increment
            cframe=cframe+1;  % Increment
            op=1;  % Increment
            oi=1;  % Increment
            Mosort_Data(:,:)=[];  % Increment

    end


end

ly=1;
lw=1;
h=figure;
jl=1;
outName='tke_15div_20frames.gif'

for tinc=(1:frames)

for la=(1:division)

   Rad_TKE(ly,1)=Avg_bottom(la,lw);  % Upper increment value
   Rad_TKE(ly,2)=TKE(la,lw);  %  Avg. Radius
   Rad_TKE(ly+1,1)=Avg_top(la,lw);  % Lower increment value
   Rad_TKE(ly+1,2)=TKE(la,lw);  % Avg. Turbulent kinetic energy
   Rad_v_r_flx_avg(jl,2)=v_r_flx1_avg(la,lw);  % Vr flux
   Rad_v_theta_flx_avg(jl,2)=v_theta_flx1_avg(la,lw);  % Vtheta flux
   Rad_v_z_flx_avg(jl,2)=v_z_flx1_avg(la,lw);  % Vz flux
   Rad_v_flx(jl,1)=Avg_top(la,lw);  % Avg. radius for flx plots
   Rad_traj(jl,1)=htraj(la,lw);  % Trajectory
   ly=ly+2;  % Increment
   jl=jl+1;  % Increment
```

```
end


plot(Rad_TKE(:,2),Rad_TKE(:,1),'.')
xlabel('TKE (m^2/s^2)')
ylabel('Radius (mm)')
title(sprintf('TKE throughout the pipe at t=%0.02f\n',ts/1000))
axis([0.002,0.017,-10,10]);

% Creating gif of the radius as function of average TKE plot

drawnow
frame_a=getframe(h);
ima=frame2im(frame_a);
[imin,cm]=rgb2ind(ima,256);
        if tinc==1

imwrite(imin,cm,outName,'gif','LoopCount',Inf,'DelayTime',1);
        else

imwrite(imin,cm,outName,'gif','WriteMode','append','DelayTime',1);
        end

savefig(sprintf('tke_15div_20frame%d.fig',tinc));  % Saving plot from
each frame

figure

subplot(2,1,1)
plot(Rad_v_flx(:,1),Rad_v_r_flx_avg(:,2),'*')
hold on
plot(Rad_v_flx(:,1),Rad_v_theta_flx_avg(:,2),'*')
hold on
plot(Rad_v_flx(:,1),Rad_v_z_flx_avg(:,2),'--*')
axis([0,10,-0.03,0.03]);
xlabel('Radius (mm)')
ylabel('Velocity Fluctuations (m/s)')
legend('v_r_flx','v_theta_flx','v_z_flx')
title(sprintf('TKE throughout the pipe at t=%0.02f\n',ts/1000))

subplot(2,1,2)
plot(Rad_v_flx(:,1),Rad_traj(:,1),'--o')
title('Number of trajectories as function of radial location')
xlabel('Radius')
ylabel('Number of Trajectories')
axis([0,10,0,400]);

savefig(sprintf('vel_flx_num_traj_15div_20frame%d.fig',tinc));  %
Saving velocity fluctuation plots in vr, vtheta and vz

lw=lw+1;  % Increment
```

```
ts=timeslice*tinc;  % Time increment
h=figure;
la=1;  % Increment
ly=1;  % Increment
jl=1;  % Increment

end
```

Appendix C – Time Dependent Velocity Profile Calculation

```
clc
close all
clear all

Trajectories=500;
%Trajectories
lastlength=0;
%Initializing the matrix dimension
length_prv=0;
% Data_prv=ones(500000,3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Reading each position files and performing various calculations
%
% Reading the files and filtering trajectory files with less
%
% than 4 data points.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%


waitbar1=waitbar(0,'Please Wait 1/3 ....');
for avg=1:Trajectories

File_name=sprintf('FilteredPosition%d.txt',avg);
% Filename
read_file_prv=dlmread(File_name,'\t',2,0);
% Reading the file

[m,n]=size((read_file_prv));

if m>4

for avg2=(1:m)
Data_prv((length_prv+avg2),1)=(read_file_prv(avg2,2));
% x-position
Data_prv((length_prv+avg2),2)=(read_file_prv(avg2,3));
% y-position
Data_prv((length_prv+avg2),3)=(read_file_prv(avg2,4));
% z-position
Data_prv_A(length_prv+avg2,1)=avg;
% Trajectory increment

end
end

length_prv=length(Data_prv_A);
```

```
waitbar(avg/Trajectories);

end

close(waitbar1)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Centering the pipe by subtracting the data set by average position
%
% Average is calculated by (max+min)/2. Radius and Angle are calculated
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

mean_x=(max(Data_prv(:,1))+min(Data_prv(:,1)))/2;
% Average velocity in x direction
mean_y=(max(Data_prv(:,2))+min(Data_prv(:,2)))/2;
% Average velocity in y direction
mean_z=(max(Data_prv(:,3))+min(Data_prv(:,3)))/2;
% Average velocity in z direction
Data=0;
% Data=ones(500000,7);
last_length=0;
ii=0;
hh=1;
traj=0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%
%
%                                User Inputs
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

prompt0=input('Plot 3D image of trajectories (Y/N)?','s');
Prompt_1=input('Would you like to analyze TKE (Y/N)?','s');


Prompt_5=input('Would you like to analyze the time dependent velocity
profile (Y/N)?','s');

if Prompt_5=='Y'
trigger_file=input('Enter the name of the trigger file
(filename.txt):','s');
nFrames=input('Enter the number of frames you would like to analyze:');
radius_div=input('Enter the number of radius divisions to get single
velocity profile:');
```

```
output_file_name=input('Enter the output file name for the velocity
profile video (filename.gif):','s');

end

waitbar2=waitbar(0,'Please Wait 2/3....');

for T=1:Trajectories

File_name=sprintf('FilteredPosition%d.txt',T);
% Filename
read_file=dlmread(File_name,'\t',2,0);
% Reading the file

[m,n]=size((read_file));

if prompt0=='Y'

figure(1)

plot3((127-(read_file(:,4))),(read_file(:,2)-mean_x),(read_file(:,3)-
mean_y))              % plotting the z vs. y
xlabel('Z position (mm)')
ylabel('X position (mm)')
zlabel('Y position (mm)')


hold on

end

if m>4

traj=traj+1;

for a=(1:m)

Data(last_length+a,1)=T;
% trajectory
% time_adj(a,1)=((read_file(a,1))-(read_file(1,1)));
% time adjustment to ensure all the position files start at t=0 ms
time_adj(a,1)=(read_file(a,1));
Data((last_length+a),2)=time_adj(a,1);
% adjusted time (ms)
Data(last_length+a,3)=(read_file(a,2)-mean_x);
% x position (mm)
Data(last_length+a,4)=(read_file(a,3)-mean_y);
% y position (mm)
Data(last_length+a,5)=sqrt(((read_file(a,2)-
mean_x)^(2))+((read_file(a,3)-mean_y)^(2)));  % Radius (mm)
```

```
Data(last_length+a,6)=(127-(read_file(a,4)));
% z position (mm)
Data(last_length+a,7)=rad2deg(atan((read_file(a,3)-
mean_y)/(read_file(a,2)-mean_x)));     % angle (degree)
Data_A(last_length+a,1)=T;
% title={'Trajectory' 'Adjusted Time (s)' 'X Position (mm)' 'Y Position
(mm)' 'Radius (mm)' 'Z Position (mm)' 'Angle'};
% Data_Matrix=[title;num2cell(Data(:,:))];

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Calculating the velocity of each coordinate direction (x,y,z)
%
% The velocity coordinate system are changed from cartesian
%
% to cylindrical coordinate system.
%
% Mean velocity and velocity fluctuations of each trajectory
%
% are calculated.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
for i=(1:(m-1))

    delta_u(last_length+i-ii,1)=(Data(last_length+i+1,3)-
(Data(last_length+i,3)));              % Change in x-position
    delta_v(last_length+i-ii,1)=(Data(last_length+i+1,4)-
(Data(last_length+i,4)));              % Change in y-position
    delta_w(last_length+i-ii,1)=(Data(last_length+i+1,6)-
(Data(last_length+i,6)));              % Change in z-position
    delta_t(last_length+i-ii,1)=(Data(last_length+i+1,2)-
Data(last_length+i,2));                % Change in time

    u_t_zero(hh,1)=(delta_u(last_length+i-
ii,1))/(delta_t(last_length+i-ii,1));
    v_t_zero(hh,1)=(delta_v(last_length+i-
ii,1))/(delta_t(last_length+i-ii,1));
    w_t_zero(hh,1)=(delta_w(last_length+i-
ii,1))/(delta_t(last_length+i-ii,1));

    u_t(last_length+i-ii,1)=(delta_u(last_length+i-
ii,1))/(delta_t(last_length+i-ii,1));       % x velocity (m/s)
    v_t(last_length+i-ii,1)=(delta_v(last_length+i-
ii,1))/(delta_t(last_length+i-ii,1));       % y velocity (m/s)

    v_r_zero(hh,1)=(u_t(last_length+i-
ii,1))*cosd(Data(last_length+i+1,7))...
                  +(v_t(last_length+i-
ii,1))*sind(Data(last_length+i+1,7));
```

```
    v_theta_zero(hh,1)=-(u_t(last_length+i-
ii,1))*sind(Data(last_length+i+1,7))...
                      +(v_t(last_length+i-
ii,1))*cosd(Data(last_length+i+1,7));

    Modata(last_length+i-ii,1)=Data(last_length+i+1,1);
% Trajectory
    Modata(last_length+i-ii,2)=Data(last_length+i+1,2);
% Time (ms)
    x_val_zero(hh,1)=Data(last_length+i+1,3);
    y_val_zero(hh,1)=Data(last_length+i+1,4);
    z_val_zero(hh,1)=Data(last_length+i+1,6);
    Modata(last_length+i-ii,3)=Data(last_length+i+1,3);
% X-position (mm)
    Modata(last_length+i-ii,4)=Data(last_length+i+1,4);
% Y-position (mm)
    Modata(last_length+i-ii,5)=Data(last_length+i+1,6);
% Z-position (mm)
    Modata(last_length+i-ii,6)=Data(last_length+i+1,7);
% Theta (degrees)

    if Modata(last_length+i-ii,6)<0
    radius(last_length+i-ii,1)=-Data(last_length+i+1,5);
    else
        radius(last_length+i-ii,1)=Data(last_length+i+1,5);
    end

    Modata(last_length+i-ii,7)=radius(last_length+i-ii,1);
% Radius (mm)


    Modata(last_length+i-ii,8)=(u_t(last_length+i-
ii,1))*cosd(Data(last_length+i+1,7))...
                              +(v_t(last_length+i-
ii,1))*sind(Data(last_length+i+1,7));        % V_r Velocity (m/s)
    Modata(last_length+i-ii,9)=-(u_t(last_length+i-
ii,1))*sind(Data(last_length+i+1,7))...
                              +(v_t(last_length+i-
ii,1))*cosd(Data(last_length+i+1,7));        % V_theta Velocity (m/s)
    Modata(last_length+i-ii,10)=(delta_w(last_length+i-
ii,1))/(delta_t(last_length+i-ii,1));   % W_velocity (m/s)



    hh=hh+1;


end

%%  Mean Velocity of each trajectories  %%

    v_r_bar(traj,1)=sum(v_r_zero(:,1))/(hh);
```

```
    v_theta_bar(traj,1)=sum(v_theta_zero(:,1))/(hh);
    w_bar(traj,1)=sum(w_t_zero(:,1))/(hh);

%% Velocity fluctuations of each trajectories %%

for xy=(1:(m-1))

    Modata(last_length+xy-ii,11)=v_r_bar(traj,1);
% Mean V_r
    Modata(last_length+xy-ii,12)=v_theta_bar(traj,1);
% Mean V_theta
    Modata(last_length+xy-ii,13)=w_bar(traj,1);
% Mean V_z
    Modata(last_length+xy-ii,14)=(Modata(last_length+xy-ii,8))-
(v_r_bar(traj,1));          % Velocity fluctuations (V_r')
    Modata(last_length+xy-ii,15)=(Modata(last_length+xy-ii,9))-
(v_theta_bar(traj,1));      % Velocity fluctuations (V_theta')
    Modata(last_length+xy-ii,16)=(Modata(last_length+xy-ii,10))-
(w_bar(traj,1));           % Velocity fluctuations (w')



end


% Clearing each elements for plotting individual trajectories %

u_t_zero(:,:)=[];
v_t_zero(:,:)=[];
v_r_zero(:,:)=[];
v_theta_zero(:,:)=[];
w_t_zero(:,:)=[];
x_val_zero(:,:)=[];
y_val_zero(:,:)=[];
z_val_zero(:,:)=[];

% Matrix array increments %%

last_length=length(Data_A);
ii=ii+1;
hh=1;

end

waitbar(T/Trajectories);

end

close(waitbar2)

title2={'Trajectory' 'Adjusted Time (ms)' 'X Position (mm)'...
```

```
    'Y Position (mm)' 'Z position (mm)' 'Theta (degrees)'...
    'Radius (mm)' 'V_r (m/s)' 'V_theta (m/s)' 'V_z (m/s)'...
    'Mean V_r (m/s)' 'Mean V_theta (m/s)' 'Mean V_z (m/s)'...
    'V_r fluctuations (m/s)' 'V_theta fluctuations (m/s)'...
    'V_z fluctuations (m/s)'};
Modata_labled=([title2;num2cell(Modata(:,:))]);
% dlmwrite('Turbulence_data_file.txt',Modata,'\t')



% Plotting the velocity profile as function of radius of the pipe

figure

scatter(Modata(:,10),radius(:,1),'*')
title('Radial Position as Function of Velocity')
xlabel('Velocity (m/s)')
ylabel('Radius (mm)')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%
%
%                   Time dependent velocity profile
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

if Prompt_5=='Y'

triggerT=dlmread(trigger_file,'\t',1,0);
trigger(1)=triggerT(1,1);

%Add mid trigger since each period consist of 2 pulse cycle

xyz=2;

for trig=1:length(triggerT)-1


midtrigger=(triggerT(trig)+triggerT(trig+1))/2;  % Trigger time per
cycle
trigger(xyz,1)=midtrigger;  % Mid trigger
trigger(xyz+1,1)=triggerT(trig+1);

xyz=xyz+2;  % Increment

end
```

```
scantime=0;  % Initial value
nPeriod=0;  % Initial Value
frames=nFrames;  % Number of frames to be analyzed

% Calcuating period per each pulse cycle

for inc=(1:length(trigger)-1)
    scantime=scantime+(trigger(inc+1)-trigger(inc));  % scan time
    nPeriod=nPeriod+1;  % Number of periods per cycle
end

Period=scantime/nPeriod;  % Period per pulse cycle
timeslice=Period/frames;  % Gated time per frame
ts=timeslice;  % time per frame


xv=1;  % Initial value
wx=0;  % Initial value
bin=1; % Initial value
prvscantime=0;  % Initial value
newscantime=(trigger(xv+1)-trigger(xv))/frames;  % scan time increment
scantimeinc=newscantime;  % scan time increment
hj=1;  % Initial value
wxy=1;  % Initial value

sortedModata=sortrows(Modata(:,:),2);  % sorting the data based on time

% Determine the start time of the trigger signal

for lk=1:length(Modata(:,1))
 if (sortedModata(lk,2)<trigger(1))
     hj=hj+1;  % Final hj value is the start of the trigger signal
     wxy=wxy+1;
 end
end



while hj<=length(Modata(:,1))

    if (sortedModata(hj,2)<=(trigger(xv)+scantimeinc) &&
sortedModata(hj,2)>=(trigger(xv)+prvscantime))  % Only consider the
data points that are in-between the specified trigger intervals
        bin=1+wx;  % Bin or frame number
        sortedModata(hj,17)=bin; % sorting the data based on the bin or
frame number
        hj=hj+1;  % Increment
    else
        wx=wx+1;  % Increment
        bin=1+wx;  % Bin number
        prvscantime=scantimeinc;  % Previous scan time increment
        scantimeinc=newscantime*bin;  % New scan time increment
```

```
          if bin>frames
             xv=xv+1;  % Increment
             newscantime=(trigger(xv+1)-trigger(xv))/frames;  % New
scan time
             wx=0;  % Initial value
             prvscantime=0;  % Initial value
             scantimeinc=newscantime;  % Scan time increment
          end
    end



end

sorted_frame_Modata=sortrows(sortedModata(:,:),17);  % Sorting data
based on the bin/frame number
cframe=1;  % Initial frame number
wq=wxy;  % Trigger start time data point
h=figure;
radiv=radius_div;  % Number of radius divisions
minradinc=min(sorted_frame_Modata(:,7));  % Minimum measured radius
value
maxradinc=max(sorted_frame_Modata(:,7));  % Maximum measured radius
value
radinc=(maxradinc-minradinc)/radiv;  % Radius increments based on min
and max value
upperradinc=minradinc+radinc;  % Upper radius value for the first
increment
rad=1;  % Initial value
rt=1;  % Initial value
valavg=1;  % Initial value
op=1;  % Initial value
outName=output_file_name;  % Name of the output file

waitbar6=waitbar(0,'Please Wait time dependent velocity profile
1/1....');

while wq<=length(Modata(:,1))

    if sorted_frame_Modata(wq,17)==cframe
        Data_vzr(op,1)=sorted_frame_Modata(wq,10);   % Z-velocity
(m/s)
        Data_vzr(op,2)=sorted_frame_Modata(wq,7);  % Radius
        Data_vzr(op,3)=sorted_frame_Modata(wq,17);  % Bin/frame number
        wq=wq+1;  % Increment
        op=op+1;  % Increment
        Data_vzr_sorted=sortrows(Data_vzr(:,:),2);  % Sorting data
based on radius

    else

        while rt<length(Data_vzr)
```

```
            if Data_vzr_sorted(rt,2)>=(minradinc) &&
Data_vzr_sorted(rt,2)<=(upperradinc)
                radData_r(rad,1)=Data_vzr_sorted(rt,2);  % Radius
                radData_vz(rad,1)=Data_vzr_sorted(rt,1);  % Z-velocity
(m/s)
                rad=rad+1;  % Increment
                rt=rt+1;  % Increment

            else

                if rad==1;
                    avgData_vzr(valavg,1)=avgData_vzr(valavg-1,1);
                    avgData_vzr(valavg,2)=avgData_vzr(valavg-1,2);
                    rad=1;  % Initial Value
                    valavg=valavg+1;  % Increment
                    minradinc=upperradinc;  % Lower interval increment
value
                    upperradinc=upperradinc+radinc;  % Upper interval
increment value
                    radData_r=[];  % clearing radius values for
plotting
                    radData_vz=[]; % clearing velocity values for
plotting
                    continue
                else

                    avgData_vzr(valavg,1)=sum(radData_vz(:,1))/rad;  %
Average velocity value
                    avgData_vzr(valavg,2)=sum(radData_r(:,1))/rad;  %
Average radius value
                    rad=1;  % Initial value
                    valavg=valavg+1;  % Increment
                    minradinc=upperradinc;  % Lower interval increment
value
                    upperradinc=upperradinc+radinc;  % Upper interval
increment value
                    radData_r=[];  % Clearing radius values for
plotting
                    radData_vz=[];  % Clearing velocity values for
plotting
                end

            end
        end


yya=[max(sorted_frame_Modata(:,7)),max(sorted_frame_Modata(:,7))];
        xa=[-0.5,1.5];

yya2=[min(sorted_frame_Modata(:,7)),min(sorted_frame_Modata(:,7))];
        plot(xa,yya,'g');
        hold on
```

```
        plot(xa,yya2,'g');
        hold on
        plot(avgData_vzr(:,1),avgData_vzr(:,2),'.')
        title(sprintf('Velocity Profile at t=%0.02f\n',ts/1000))
        xlabel('Axial Velocity (m/s)')
        ylabel('Radius (m)')
        axis([-0.5,1.5,-15,15])

        % Creating the gif of the velocity profile

        drawnow
        frame=getframe(h);
        ima=frame2im(frame);
        [imin,cm]=rgb2ind(ima,256);
        if cframe==1

imwrite(imin,cm,outName,'gif','LoopCount',Inf,'DelayTime',.5);
        else

imwrite(imin,cm,outName,'gif','WriteMode','append','DelayTime',.5);
        end

        wq=wq+1;  % Increment
        op=1;  % Initial value
        cframe=cframe+1;  % Frame number
        ts=timeslice*cframe;   % Time increment
        Data_vzr=[];  % clearing the matrix
        storeavgData=avgData_vzr(:,:);  % Storing avg. radius and
velocity
        avgData_vzr=[];  % clearing the avg. velocity
        radData_r=[];  % clearing the radius
        radData_vz=[];  % clearing the z-velocity
        rad=1;  % Initial value
        minradinc=min(sorted_frame_Modata(:,7));  % Lower increment
radius value
        upperradinc=minradinc+radinc;  % Upper increment radius value
        valavg=1;  % Initial value
        rt=1;  % Initial value

        h=figure;
    end

    waitbar(wq/(length(Modata(:,1))));
end

close(waitbar6);

end
```

Appendix D – Pressure Wave Velocity and Phase Angle Calculation

```
clc
clear all
close all


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%
%
%                              Reading data
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

Prompt='Enter the file name in following format (filename.txt):';
% Prompt message asking for filename
filename=input(Prompt,'s');
file=sprintf(filename);   % Formatting data into string
readfile=dlmread(file,'\t',22,1);
% Reading the files starting from line 23


Outlet_Pressure=readfile((1:length(readfile)),1);
% Outlet Pressure (all the rows, 1st column) (PSI)
Inlet_Pressure=readfile((1:length(readfile)),2);
% Inlet Pressure (all the rows, 2nd column)  (PSI)
Differential_Pressure=readfile((1:length(readfile)),3);
% Differential Pressure (all the rows, 3rd column)  (PSID)
Flow_Rate=readfile((1:length(readfile)),4);
% Flow Rate correlated from the differential pressure (Equation in the
labview)  (GPM)
Trigger=readfile((1:length(readfile)),5);
% Trigger Signal  (Volts)
Time=readfile((1:length(readfile)),6);
% Time (seconds)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%                            Graphing Raw Data
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

GraphingPrompt='Would you like to graph raw data (type: Y or N):';
% User Input for graphing the raw data
Read_GraphingPrompt=input(GraphingPrompt,'s');
```

```
if Read_GraphingPrompt=='Y'
% If user wants to see the data plot various graphs

% Graphing Outlet Pressure as Function of Time

OPPrompt='Would you like to graph outlet pressure as function of time?
(type: Y or N):';
Read_OPPrompt=input(OPPrompt,'s');

if Read_OPPrompt=='Y'

    plot(Time,Outlet_Pressure)
    title('Outlet Pressure')
    xlabel('Time (Seconds)')
    ylabel('Outlet Pressure (PSI)')
else
end

% Graphing Inlet Pressure as Function of Time

IPPrompt='Would you like to graph inlet pressure as function of time?
(type: Y or N):';
Read_IPPrompt=input(IPPrompt,'s');

if Read_IPPrompt=='Y'

    figure
    plot(Time,Inlet_Pressure)
    title('Inlet Pressure')
    xlabel('Time (Seconds)')
    ylabel('Inlet Pressure (PSI)')
else
end

% Graphing Differential Pressure as Function of Time

DPPrompt='Would you like to graph differential pressure as function of
time? (type: Y or N):';
Read_DPPrompt=input(DPPrompt,'s');

if Read_DPPrompt=='Y'

    figure
    plot(Time,Differential_Pressure)
    title('Differential Pressure')
    xlabel('Time (seconds)')
    ylabel('Differential Pressure (PSID)')
else
end
```

```
% Graphing Flow Rate as Function of Time

FRPrompt='Would you like to graph flow rate as function of time? (type:
Y or N):';
Read_FRPrompt=input(FRPrompt,'s');

if Read_FRPrompt=='Y'

    figure
    plot(Time,Flow_Rate)
    title('Flow Rate')
    xlabel('Time (seconds)')
    ylabel('Flow Rate (GPM)')
else
end

% Graphing Trigger Signal as function of Time

TPrompt='Would you like to graph trigger as function of time? (type: Y
or N):';
Read_TPrompt=input(TPrompt,'s');

if Read_TPrompt=='Y'

    figure
    plot(Time,Trigger)
    title('Trigger Signal')
    xlabel('Time (seconds)')
    ylabel('Trigger Signal (Volts)')
else
end
else
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%                               Post Processed Data
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%


%% Constants used for calculating flow rate (GPM) as function of
differential pressure (using calibration curve provided by the
manufacturer)

warning('The Flow Calibration Constants are for Orifice Plate B-3.
Change  it if necessary.')
```

```
Beta=0.7;                   % Beta Ratio (Bore Diameter/ Pipe Diameter)
Y=1;                        % Expansion Factor
Fa=1;                       % Thermal Expansion Factor
P_l=62.3;                   % Density at Line (flowing) conditions
(lbs/ft3)
d=0.580;                    % Bore Diameter of the Orifice Plate (inches)
v=0.000001004;              % kinematic viscosity of water at 20C (m^2/s)
D=0.01905;                  % Diameter of the Pipe (m)
Re=40000;                   % Estimated Reynolds Number

%% Calculating the flow rate using calibration equation
for x=(1:length(readfile))

    C=0.5959+(0.0312*(Beta^(2.1)))-
(0.1840*(Beta^(8)))+(91.71*(Beta^(2.5))*((Re)^(-0.75)));
    K=(C*(1))/(sqrt(1-(Beta^(4))));

    if Differential_Pressure(x)<0
        Abs_DP=Differential_Pressure(x);
        FR=-(44.748*(d)^(2)*K*Y*Fa*sqrt((abs(Abs_DP))*27.7076/(P_l)));
    else

FR=44.748*(d)^(2)*K*Y*Fa*sqrt((Differential_Pressure(x))*27.7076/(P_l))
;
    end

    Adjusted_Flow_Rate(x,1)=FR;
    velocity=(abs(FR)*0.0037854*4)/(60*pi*(D*D));
    Re=(velocity*D)/(v);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%                           Trigger Synch
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

%% Trigger synchronization %%

Trigger_Inveon=4.44054;
Trigger_Daq=4.4793;
Trigger_Difference=((Trigger_Daq)-(Trigger_Inveon))+0.02541

for T=(1:length(readfile))
    if Time(T)>=Trigger_Difference && Time(T-1)<Trigger_Difference
        Start_Time=T;
    else
    end
end

for tt=((Start_Time):(length(readfile)))
```

```
    Time_Synch(tt-(Start_Time-1),1)=((Time(tt))-(Time(Start_Time)));
end

%%%%%%%%%%%%%%%%%%%%%%%% Outlet Pressure %%%%%%%%%%%%%%%%%%%%%%%%

%% Constants used to reduce noise (via averaging the data)  %%

Prompt_OP_limits='Enter the lower limit data point for the Outlet
Pressure:';
OPLow_Prompt=input(Prompt_OP_limits);

OPLow=OPLow_Prompt+10089;                                  % Initial Lower
limit of the data set to be averaged  (10089 is when pulse starts)

Prompt_OP_limit='Enter the upper limit increment for the Outlet
Pressure:';
OPHigh_Prompt=input(Prompt_OP_limit);

OPHigh=OPHigh_Prompt+10089;                                % Initial Upper
limit of the data set to be averaged
difference=(OPHigh-(OPLow-1));                             % Incremental
step (differnce in the Upper limit to Lower limit)
OP_Previous=0;                                             % Initial Outlet
Pressure
A_Previous=0;                                              % Initial
Increment
Row_prev=1;                                                % Previous Row
count
Row=1;                                                     % Initial Row
count

%% Averaging the Pressure data in increments of "difference" %%

for z=((10089):((round((length(readfile))/(difference)))-1))


    for a=(OPLow:OPHigh)                                   % Going from
Lower limit to Upper limit
        OP_New=Outlet_Pressure(a);                         % Outlet Pressure
at that current step
        Total_OP=OP_Previous+OP_New;                       % Total Outlet
Pressure ( Sum of the Previous Outlet Pressures + The current Outlet
Pressure)
        OP_Previous=Total_OP;                              % Sum of all the
Previous Outlet Pressures within the limits
        A_New=1;                                           % New Iteration
counter
        A_Total=A_Previous+A_New;                          % Total
Iterations ( Sum of the Previous Iterations + The current Iteration)
        A_Previous=A_Total;                                % Sum of all the
Previous Iternations within the limits
```

```
        Average_OP(Row,1)=((Total_OP)/(A_Total));      % Averaging the
Outlet Pressures within the limits
        Average_Time(Row,1)=(Time_Synch(OPHigh));      % Averaging the
Time  within the limits


    end
    OPLow=OPLow+difference;                            % Steping up to
go to the next Lower limit
    OPHigh=OPHigh+difference;                          % Stepping up to
go to the next Upper limit
    Row_New=1;                                         % Row counter
    Row=Row_prev+Row_New;                              % Total number of
Rows ( Sum of the Previous Rows + The New Rows )
    Row_prev=Row;                                      % Sum of all the
Previous Rows
    OP_Previous=0;                                     % Resetting the
Previous Outlet Pressure to 0
    A_Previous=0;                                      % Resetting the
Previous Iternation to 0


end

%%%%%%%%%%%%%%%%%%%%%%%% Inlet Pressure %%%%%%%%%%%%%%%%%%%%%%%%

%% Constants used to reduce noise (via averaging the data)  %%

Prompt_IP_limits='Enter the lower limit data point for the Inlet
Pressure:';
IP_Low_Prompt=input(Prompt_IP_limits);

IPLow=IP_Low_Prompt+9737;
% Initial Lower limit of the data set to be averaged (9737 is when
pulse starts)

Prompt_IP_limit='Enter the upper limit increment for the Inlet
Pressure:';
IP_High_Prompt=input(Prompt_IP_limit)+9737;

IPHigh=IP_High_Prompt;                                                 %
Initial Upper limit of the data set to be averaged
difference_IP=(IPHigh-(IPLow-1));                                      %
Incremential step (differnce in the Upper limit to Lower limit)
IP_Previous=0;                                                         %
Initial Outlet Pressure
A_Previous_IP=0;                                                       %
Initial Increment
Row_prev_IP=1;                                                         %
Previous Row count
Row_IP=1;                                                              %
Initial Row count
```

```
%% Averaging the Inlet Pressure data in increments of "difference" %%

for zz=((9737):((round((length(readfile))/(difference_IP)))-353))

    for aa=(IPLow:IPHigh)                                           %
Going from Lower limit to Upper limit
        IP_New=Inlet_Pressure(aa);                                  %
Inlet Pressure at that current step
        Total_IP=IP_Previous+IP_New;                                %
Total Outlet Pressure ( Sum of the Previous Outlet Pressures + The
current Outlet Pressure)
        IP_Previous=Total_IP;                                       %
Sum   of all the Previous Outlet Pressures within the limits
        A_New_IP=1;                                                 %
New   Iteration counter
        A_Total_IP=A_Previous_IP+A_New_IP;                          %
Total Iterations ( Sum of the Previous Iterations + The current
Iteration)
        A_Previous_IP=A_Total_IP;                                   %
Sum   of all the Previous Iternations within the limits
        Average_IP(Row_IP,1)=((Total_IP)/(A_Total_IP));             %
Averaging the Outlet Pressures within the limits
        Average_Time_IP(Row_IP,1)=(Time_Synch(IPHigh));
% Averaging the Time  within the limits
    end
    IPLow=IPLow+difference_IP;                                      %
Going to the next Lower limit
    IPHigh=IPHigh+difference_IP;                                    %
Going to the next Upper limit
    Row_New_IP=1;                                                   %
Row counter
    Row_IP=Row_prev_IP+Row_New_IP;                                  %
Total number of Rows ( Sum of the Previous Rows + The New Rows )
    Row_prev_IP=Row_IP;                                             %
Sum of all the Previous Rows
    IP_Previous=0;                                                  %
Set   the Previous Outlet Pressure to 0
    A_Previous_IP=0;                                                %
Set   the Previous Iteration to 0

end

min_OP=min(Average_OP(:,1));
min_IP=min(Average_IP(:,1));
max_OP=max(Average_OP(:,1))-min_OP;
max_IP=max(Average_IP(:,1))-min_IP;


for xyz=(1:length(Average_Time(:,1)))
    Norm_Avg_OP(xyz,1)=((Average_OP(xyz,1)-min_OP)/(max_OP))+0.0203;
    Norm_Avg_IP(xyz,1)=(Average_IP(xyz,1)-min_IP)/(max_IP);
```

```
end

% Plotting average pressure data

plot(Average_Time(:,1),Norm_Avg_OP(:,1),'.-')
hold on
plot(Average_Time_IP(:,1),Norm_Avg_IP(:,1),'.-')

xlabel('Average Time (Seconds)')
ylabel('Normalized Pressure Signal')
legend('Outlet Pressure','Inlet Pressure')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Linearly extrapolating to acquire time at specific normalized
pressure  %
% value
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%



intial_val=0.3;  % Initial normalized pressure value
inc=0;  % Initial increment


for yy=(1:13)

ii_OP=1;  % Increment
interval(1,yy)=intial_val+inc  % Normalized pressure value
last_int_OP=1;  % Increment

% Outlet Pressure

for xx_OP=(1:(length(Norm_Avg_OP)-22))
    if Norm_Avg_OP(xx_OP,1)<=interval(1,yy) &&
Norm_Avg_OP(xx_OP+1,1)>=interval(1,yy) ||
Norm_Avg_OP(xx_OP,1)>=interval(1,yy) &&
Norm_Avg_OP(xx_OP+1,1)<=interval(1,yy)

        dif_OP=xx_OP-last_int_OP;  % Finding the difference and
filtering to avoid double counting

        if dif_OP<=3

            continue

        else
        dt_OP(ii_OP,yy)=(Average_Time(xx_OP+1,1)-
Average_Time(xx_OP,1));  % Difference between current and next time
```

```
        dp_OP(ii_OP,yy)=(Norm_Avg_OP(xx_OP+1,1)-Norm_Avg_OP(xx_OP,1));
% Difference between current and next pressure value

time_OP(ii_OP,yy)=(((dt_OP(ii_OP,yy))/(dp_OP(ii_OP,yy)))*(interval(1,yy
)-Norm_Avg_OP(xx_OP,1)))+Average_Time(xx_OP,1);  % Linearly
extrapolating time at specific pressure value
        int(ii_OP,yy)=xx_OP;  % Increment
        last_int_OP=int(ii_OP,yy);  % Increment
        ii_OP=ii_OP+1;  % Increment

        end

    end

end

% Inlet Pressure

ii_IP=1;  % Increment
last_int_IP=1;  % Increment

for xx_IP=(1:(length(Norm_Avg_IP)-20))
   if Norm_Avg_IP(xx_IP,1)<=interval(1,yy) &&
Norm_Avg_IP(xx_IP+1,1)>=interval(1,yy) ||
Norm_Avg_IP(xx_IP,1)>=interval(1,yy) &&
Norm_Avg_IP(xx_IP+1,1)<=interval(1,yy)

        dif_IP=xx_IP-last_int_IP;  % Finding the difference and
filtering to avoid double counting

        if dif_IP<=3

            continue

        else

        dt_IP(ii_IP,yy)=(Average_Time_IP(xx_IP+1,1)-
Average_Time_IP(xx_IP,1));   % Difference between current and next time
        dp_IP(ii_IP,yy)=(Norm_Avg_IP(xx_IP+1,1)-Norm_Avg_IP(xx_IP,1));
% Difference between current and next pressure

time_IP(ii_IP,yy)=(((dt_IP(ii_IP,yy))/(dp_IP(ii_IP,yy)))*(interval(1,yy
)-Norm_Avg_IP(xx_IP,1)))+Average_Time_IP(xx_IP,1);  % Linearly
extrapolating time at specific pressure value
        int_IP(ii_IP,yy)=xx_IP;  % Increment
        last_int_IP=int_IP(ii_IP,yy);  % Increment
        ii_IP=ii_IP+1;  % Increment

        end

    end
```

```
end

% Calculating the time difference between the outlet and the inlet
pressure

for yz=(1:length(time_IP))
    dt(yz,yy)=(time_OP(yz,yy)-time_IP(yz,yy));

end

inc=0.05*yy;  % Pressure increment

end


i=1;

for row=(1:length(dt(:,1)))

    for col=(1:length(dt(1,:)))

    if mod(row,2)==1

    dt_prime(i,1)=dt(row,col);  % Differential Time (s)
    PWV(i,1)=1.16/dt_prime(i,1);  % Pressure Wave Velocity (m/s)
    theta(i,1)=360*2.07*dt_prime(i,1); % Phase Angle (degrees)
    i=i+1;  % Increment

    else

    dt_prime(i,1)=dt(row,(yy+1)-col);  % Differential Time (s)
    PWV(i,1)=1.16/dt_prime(i,1);  % Pressure Wave Velocity (m/s) (1.16
cm = length of the test section)
    theta(i,1)=360*2.07*dt_prime(i,1);  % Phase Angle (degrees) (2.07 =
pulse frequency)
    i=i+1;  % Increment

    end

    end

end

% Plotting Pressure Wave Velocity and Phase Angle

figure

plot(PWV(:,1),'.-');
```

```
xlabel('Sample Number')
ylabel('Pressure Wave Velocity (m/s)')

figure

plot(theta(:,1),'.-')

xlabel('Sample Number')
ylabel('Phase Angle (degrees)')
```

VITA

Nitant Patel was born in Bardoli, India on May 16, 1995. He moved to United States in 2007 and graduated from Anderson County High School on May 2012. He intended to pursue a business degree, but switched to nuclear engineering after first year of college. On May 2016, he obtained his Bachelor of Science degree in Nuclear Engineering and continued his education by pursing Master of Science in Nuclear Engineering. Nitant interned during the summer of 2017 at Oak Ridge National Laboratory helping support construction of the material test capsules for the high flux isotope reactor (HFIR), and aiding in design assessment of a thermosiphon cooled nuclear fuel irradiation and test platform for HFIR.