Traditional Image Processing and Modern Computer Vision Techniques for the Study of Two-Phase CO_2 Flow

Shai Kadish

Thesis presented for the Degree of

MASTER OF ENGINEERING

in the department of Electrical Engineering

UNIVERSITY OF CAPE TOWN

December 2021

Supervisors: Professor Jarryd Son, Professor Edward Boje, Professor Sahal Yacoob and David Schmid

Contents

Table of Figures

Table of Tables

Nomenclature

Symbols

Some Symbols seen below represent multiple values, but the context makes the use unambiguous.

a	Ellipse Major-Axis length
b	Ellipse Minor-Axis Length
d	Diameter [m]
Fr	Froude Number, $Fr = \frac{u^2}{gd}$
G	Mass Flux $[\frac{kg}{m^2 s}]$
g	Gravitational Constant, g=9.81 $[\frac{m}{s^2}]$
I	Image intensity
m	Mass [kg] OR Gradient of a straight line
$\dot{m}$	Mass Flow Rate $[\frac{kg}{s}]$
P	Pressure, $[\frac{N}{m^2}$ or Pa]
Re	Reynolds Number, $Re = \frac{\rho u d}{\mu}$
U	Superficial Velocity $[\frac{m}{s}]$
u	Velocity $[\frac{m}{s}]$
V	Volume $[m^3]$
x	Vapour Quality
α	Void fraction
θ	Angle of inclination/rotation [rad] OR Near/Far factor
μ	Dynamic Viscosity $[\frac{kg}{ms}]$
ρ	Density $[\frac{kg}{m^3}]$
σ	Surface Tension $[\frac{N}{m}]$ OR Standard Deviation

Subscripts

g	Gas Phase
gm	Drift
h	Homogeneous
l	Liquid Phase
m	Two-Phase mixture
w	Water

Declaration

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signed: Shai Kadish 

15 December 2021

Acknowledgements

This work would not be possible without the support provided by my parents throughout this process. I would also like to acknowledge and thank all my friends, my brother Josh, and my girlfriend Julia, for making the last two years a learning experience which extends beyond the work found here.

I would like to thank Sahal Yacoob for bringing me to this topic, and for his efforts in getting me to CERN. I would also like to thank Professors Jarryd Son and Edward Boje for their invaluable guidance and assistance throughout the completion of this thesis.

Finally, I would like to acknowledge and thank David Schmid for his abundant generosity in time and spirit throughout this process. Thank you, David, for always making yourself available to help me better understand the topic, for data acquisition, or for a chat. You also helped make my time at CERN a very special memory I will have for a lifetime.

In addition to the above, I would like to acknowledge the following:

This work is based on the research supported by wholly / in part by the National Research Foundation of South Africa (Grant Numbers: 131702 and 129448). Computations were performed using facilities provided by the University of Cape Town's ICTS High Performance Computing team: hpc.uct.ac.za.

Abstract

The work presented here details the development of software-based tools for the extraction of physical parameters which describe two-phase (gas-liquid) upwardly flowing CO_2, for the purpose of using these parameters as sensor data for a control feedback loop, and for the automatic detection of flow regime transition, which is useful for the development of flow regime maps. The core focus of this thesis is the development of these tools in such a way that their primary input is an image or set of images. To achieve this, two schools of thought are explored: First, traditional image processing techniques are employed to study the flow. These techniques require manual image feature selection, and they make use of purpose-built algorithms to extract the desired parameters from an input image using these features. The second approach makes use of modern computer vision techniques, where the image features are automatically learnt through machine learning, and an end-to-end network design makes use of these features to extract the desired output without manual tuning.

Traditional image processing is used to develop an algorithm which extracts the void fraction value from an image of bubbly flow. This algorithm works by detecting individual bubbles within the input image, and then estimating the volume of each bubble (with uncertainty) in order to calculate the final void fraction. The outputs seen from this algorithm correlated well with those produced by established models for calculating void fraction, but the problem with this algorithm is its limited scope of use: it is only applicable to images of bubbly flow, a flow regime which exists for only a small portion of the total possible vapour quality range under steady state conditions.

Two different tools, which share a similar architecture, and which classify flow regime and vapour quality respectively, were successfully developed using modern computer vision techniques. These models both take in video clips as their inputs. The approach makes use of deep learning to train a *convolutional neural network (CNN),* which is used for individual frame classification and image feature extraction, and a deep *long short-term memory (LSTM)* network, used to capture temporal features present in a sequence of image feature sets, and to make a final vapour quality or flow regime classification. The proposed architecture achieves accurate flow regime and vapour quality classifications in practical application to two-phase CO_2 flow in vertical tubes based on off-line data and an on-line prototype implementation, developed as a proof of concept for the use of these models within a feedback control loop. The successful application of the *LSTM* network reveals the significance of temporal information for image based studies of multi-phase flow.

When comparing these parallel developments, the advantages and disadvantages of the two approaches can be clearly seen. Traditional image processing requires far more extensive domain specific knowledge and manual fine tuning, but this approach allows for a user to clearly understand the outputs of the algorithm, whether they are correct or incorrect, as the internal mechanisms of the algorithm are all purpose built. This is not the case for deep learning based modern computer vision methodologies, which are more of a “black box”. These methods require a large amount of training data, but less domain specific knowledge, as the important features from the input data do not need to be manually selected and processed by the user. This leads to high performing systems which are difficult to understand and debug. There are different cases in which each of these methods would be preferable, but with the rapid evolution of deep learning and computer vision over the last few years, deep learning based computer vision appears to be replacing the traditional approach in many cases.

Introduction

The Detector-Technology Cooling Group [1], based at CERN, is responsible for the designing and construction of future cooling units for high-luminosity experiments undertaken at the large hadron collider (LHC), with one such experiment being the ATLAS experiment [2]. With CERN moving away from commonly used refrigerants which are harmful to the environment, towards natural refrigerants, the Detector-Technology Cooling Group has begun designing future cooling units which make use of Carbon Dioxide (CO_2) as the coolant. The Detector-Technology Cooling Group has been involved in research aimed to better understand the thermo- and fluid dynamical properties of CO_2 to design appropriate and efficient refrigeration systems.

The current work aims to develop tools to assist in that research, specifically in the study of vertical upwardly flowing two-phase gas-liquid CO_2 flow. The tools developed herein make use of image processing and computer vision techniques to extract parameters describing the fluid, by using images of the flow as inputs to these various methods. In addition to describing these tools, the current work includes the research and development undertaken in producing them.

The dissertation is divided into three parts. In *Part I*, background research related to the fluid-dynamical aspects of two-phase CO_2 flow is explored, with this research informing the development of all other work done. This section also includes a review of existing methods used to study two-phase flow, and a description of the testing environment used to produce the images required for further study. *Part II* details research and algorithms which make use of traditional image-processing techniques to classify vapour content in bubbly flow. *Part III* includes all research and methods which make use of computer vision techniques. After investigating methods which make use of both the more traditional image processing techniques, and the more modern computer vision techniques, this thesis concludes by discussing the differences between these two fields, and what has been made clear by their comparison.

Part I – Two-Phase Flow

This chapter details the flow characteristics under investigation in *Parts II and III*, with the research under review here informing all later developments. Also included here is a review of existing methods used to study two-phase flow, and a description of the testing environment used.

1 Parameters of two-phase flow

Detailed below are some key parameters commonly used to describe two-phase flow, and their applications.

Figure 1.1- Two-phase flow; Gas bubbles are dispersed in a continuous upwardly flowing liquid phase.

1.1 Pressure-drop

A key parameter of two- phase flow is the pressure drop between two points in the flow [3]. The pressure drop is the difference in total pressure between two points of a fluid carrying network. This difference is the nett effect of three different sources of pressure change: hydrostatic pressure drop, which is a difference in pressure due to changes in elevation, frictional pressure drop, which comes about as a result of fluctuating kinetic and potential energy due to friction between the fluid and the flow channel walls, as well as within the fluid itself, and acceleration pressure drop, which is a result of fluctuating fluid momentum when flowing from an inlet to an outlet [4]. Traditionally, the most complex of these to analyse is the frictional pressure drop. Much work in the modelling of frictional pressure drop is based on the seminal work by Lockhart and Martinelli [5], where the Lockhart-Martinelli parameter was introduced, which is used frequently in subsequent works studying two-phase flow pressure drop [6] [7]. Pressure drop is an important design variable relating to the power required to transport fluid in a multi-phase system, as well as an input in turbulence modelling for multiphase flow [8].

1.2 Flow Regime

Another important parameter used to characterize gas-liquid flow is the flow regime. The flow regime describes the spatial distribution between the gas and liquid phases, with the different regimes being identified by gas bubble characteristics inherent to each regime [9]. Different flow regimes can be observed across different flow channel shapes, orientations, and operating conditions, with different flow regimes also arising because of properties of the flow itself, such as phase velocity and vapour quality [10]. That said, the flow regimes commonly observed in vertical flow are well defined in the literature [11]. With an increase in gas velocity, typically an increase in bubble size can be observed [11], affecting the regime definition for a given two-phase flow.

Much of the literature aims to relate flow regime classes to measured physical characteristics of the flow, and to define the regimes and their transitions in terms of these physical characteristics. The mapping of these flow patterns and physical characteristics can be represented in a flow-regime map, where the boundaries between the different flow regimes are clearly defined [12]. These maps can describe flow regimes across various parameters (such as mass flow rate and phase velocity) and are unique for a set of experimental conditions and fluids being analysed, with vastly different maps being observed with seemingly minor changes to experimental conditions. An inherent issue in the efficacy of flow regime maps is that the transition between regimes is gradual, rather than a clearly defined transition as the maps describe. Additionally, due to the qualitative nature of flow patterns, the definition of a specific flow regime is subjective, leading to differences in regime definitions across the literature, without any consistent physical parameter being used to verify the occurrence of a flow regime.

The classification of the flow regime for multiphase flow is essential in many industrial sectors such as energy producing industries, metallurgical industries, and many chemical processing industries. The study of flow regime is ubiquitous in such a diverse range of sectors because it reveals essential information about the flow behavior as well as physical flow parameters of the two-phase flow under investigation. The flow pattern of the fluid is important in calculating the pressure drop because the spatial distribution of the gas and liquid phases along both the flow channel's cross section and its length are required when calculating the frictional pressure drop of the system [10].

There has been work done in developing flow regime maps for CO_2 multi-phase flow in a horizontal orientation [13], and very recently, in a vertical orientation [14]. The latter work made use of the flow characterisation tools reported in this dissertation. The specific flow regimes explored in the current work are detailed in *Part III, Section 2.1.*

1.3 Parameters of individual bubbles

In flow regimes commonly observed whilst low vapour qualities (below 0.2) and low gas velocities are present, such as bubbly and slug flow, individual bubbles can be observed within the flow. Work has been done in modelling the shapes of these bubbles as they change depending on certain dimensionless parameters describing the flow [15]. This work shows that flows which are defined by low Reynolds Numbers and low Eotvos numbers have bubbles which are more spherical in shape. As these dimensionless parameters increase, the bubbles will tend to be more ellipsoidal in shape, and with high Eotvos and Reynolds Numbers, bubbles which are capped by a spherical side with an adjacent flat side can be observed. Such a bubble can be seen in *Figure 3.1*. Bubbly flow regimes will likely have bubbles of these three shapes, and as such, the geometric parameters of individual bubbles can be approximated in terms of ellipsoidal parameters, given that a spherical bubble can be seen as a special case of an ellipsoidal bubble, and spherical capped bubbles can be approximated closely by an ellipsoidal shape. Although this assumption holds for most bubbles seen in the Bubbly regime, there are cases where bubbles have non-ellipsoidal shapes. The estimation of these bubbles shapes using ellipses affects the potential accuracy of the algorithm developed in *Part II.*

1.4 Vapour Quality

While void fraction (discussed below) refers to the volumetric fraction of two-phase flow, the vapour quality, x, refers the mass fraction of a saturated mixture which is vapour, and can be expressed as follows:

$$x = \frac{m_g}{m_g + m_l} \tag{1.1}$$

By this definition, a saturated liquid will have a vapour quality of x=0, and a saturated vapour will have a quality of x=1.

The vapour quality of a fluid can be determined mathematically when the specific enthalpy of the fluid is known, such as in flow boiling tests [16]. Because of this, vapour quality is often used as an input parameter when modelling the more difficult to measure void fraction of a system [10]. Apart from this, vapour quality is also directly related to the flow boiling heat transfer behaviour of a system, and the related flow boiling heat transfer coefficients [16]. This is because with a constant saturation temperature, an increase in the enthalpy of a system due to latent heat added during a boiling process results in the vapour quality of the system increasing due to this heat acquisition [17]. As such, measuring the vapour quality of the system is crucial in modelling the flow behaviour of two-phase flow in boiling tests.

1.5 Void Fraction

The key parameter of multiphase flow to be investigated in the *Part II* of this work is the void fraction of the flow, α, also known as gas hold-up. This parameter is defined as the fraction of total volume within the two-phase flow which belongs to the gas phase, and can be expressed as follows:

$$\alpha = \frac{V_g}{V_g + V_l} \tag{1.2}$$

This equation represents the volumetric void fraction, while the cross-sectional void fraction can be defined by a similar ratio in terms of the cross-sectional area of the flow channel. The use of either the volumetric or cross-sectional void fraction will depend on the measurement technique used in determining the void fraction. This single parameter can be used to derive other useful two-phase parameters such as two-phase density, and the velocities of the gas and liquid phases. Two-phase density is an essential parameter used to calculate the hydrostatic pressure drop, and as such, an accurate void fraction measurement is required to model the pressure drop in gas-liquid flow [10].

Unlike vapour quality, which can be calculated directly when the latent heat added to a system is known, the measurement of void fraction presents more of a challenge due to the complexities inherent in measuring gas volume, with some of these measurement techniques discussed in greater detail in *Part I, Section 3*. There are many existing correlations used to model void fraction as a function of both the fluid properties and the experimental conditions of the flow [18] [19] [20]. These models of void fraction prediction are discussed in greater detail in the next section.

2 Existing void fraction correlations

As discussed in the next section, there are myriad experimental methods by which to generate data describing physical properties of two-phase flow. Using the data generated from these techniques, correlations can be developed to model relationships between physical parameters. There have been many such correlations developed to model the response of void fraction to changes in more easily measured parameters. While these correlations can be accurate, developing correlations that are applicable to a broad set of flow channel and fluid characteristics has proven to be difficult. Void fraction correlations are extremely sensitive to the characteristics of the channel in which the two-phase fluid flows, such as pipe diameter and orientation [10]. The various flow regimes, and different phase flow rates

also have a large effect on the accuracy of each correlation, and as such, in order to produce accurate predictions from an existing model, the pipe orientation, pipe diameter, flow pattern, fluid properties and phase flow rates under investigation must be considered when selecting such a model [18] [10]. In short, although there exist many void fraction correlations, most of these correlations are restricted in their general usefulness to the experimental conditions by which they were generated.

Despite the uniqueness of each correlation, they can still be classified in terms of the underlying model upon which they are based. The different void fraction correlations can be classified within five groups: homogeneous correlations, slip ratio correlations, drift flux correlations, $K\alpha_h$ correlations, and Miscellaneous empirical correlations [10] [20] [19]. These groups are respectively discussed in *sections 2.1 to 2.5* below, with correlations developed under comparable experimental conditions to the data generated in the current work being highlighted in the text, and included in *Table 1.1* below. Additionally, all parameters seen in *Table 1.1* which are used to describe these correlation groups are detailed in each groups respective section.

Table 1.1- Existing Void Fraction Correlations

Slip Ratio correlations				$\alpha = \left[1 + A\left(\frac{1-x}{x}\right)^p \left(\frac{\rho_g}{\rho_l}\right)^q \left(\frac{\mu_l}{\mu_g}\right)^r\right]^{-1}, S = \frac{u_g}{u_l}$
Correlation	A	p	q	r
Lockhart and Martinelli [5]	0.28	0.64	0.36	0.07
Chisholm [20]	$\sqrt{1 - x\left(1 - \frac{\rho_l}{\rho_g}\right)}$	1	1	0
Premoli et al [21]	$1 + F_1\left(\frac{y}{1+yF_2} - yF_2\right)^{0.5}$, where $F_1 = 1.578(Re_l^{-0.19})\left(\frac{\rho_l}{\rho_g}\right)^{0.22}$, $F_2 = 0.0273We_l(Re_l^{-0.51})\left(\frac{\rho_l}{\rho_g}\right)^{-0.08}$, $y = \frac{\alpha_h}{1-\alpha_h}$	1	1	0
Homogeneous	1	1	1	0
Slip Ratio	S	1	1	0
$K\alpha_h$ correlations				$\alpha = K\alpha_h$
Correlation	K			
Hughmark [22]	$\frac{Re^{\frac{1}{6}}Fr^{\frac{1}{8}}}{\lambda^{\frac{1}{4}}}$, where $\lambda = \frac{U_l}{U_l + U_g}$			
Armand [23] -Massena [24]	$0.833 + 0.167\text{x}$			
Drift Flux correlations				$\alpha = \frac{U_g}{C_0 U_m + U_{gm}}$

Correlation	U_{gm}	C_0
Rouhani and Axelsson [25]	$1.18\left(\frac{g\sigma(\rho_l-\rho_g)}{\rho_l^2}\right)^{0.25}$	$1+0.2(1-x)$
Toshiba [26]	0.45	1.08
Bhagwat and Ghajar [27]	$\left(\frac{\mu_l}{\mu_w}\right)^{-0.25}(0.35\sin(\theta)+0.54\cos(\theta))\left(\frac{9.81D(\rho_l-\rho_g)}{\rho_l}\right)^{0.5}(1-\alpha)^{-0.5\sin(\theta)}$, Where θ describes the flow channel orientation	$\left(\frac{1}{1+cos(\theta)^{1.25}}\right)^{(1-\alpha)^{0.5}}+0.18\left(\frac{U_l}{U_m}\right)^{0.1}$

2.1 Homogeneous Model

The Homogeneous model assumes that the liquid and gas phases of the flow are travelling at the same velocity and share a definite interface [10]. It can be derived from first principles by using the definitions of vapour quality and void fraction, and is expressed as follows:

$$\alpha_h = \left[1+\left(\frac{1-x}{x}\right)\left(\frac{\rho_g}{\rho_l}\right)\right]^{-1} \tag{1.3}$$

Because of the limited cases where the gas and liquid phases are flowing at the same velocity, the homogeneous model is reasonably accurate only for those circumstances. The flow regimes where this model could be applied include those observed when a low vapour quality is present, such as bubbly flow, and those when a high vapour quality is present, such as annular or mist flows [10]. The homogeneous model is a special case of the slip model, where the slip ratio *S* is equal to one.

2.2 Slip ratio model

Like the homogeneous model, the slip ratio model assumes that the gas and liquid phases have a definite interface, but unlike the homogeneous model, it assumes that the two phases flow at different velocities. As such, the slip ratio model is also known as the separated flow model. This is represented mathematically by introducing a slip ratio *S* into the homogeneous model equation as follows:

$$\alpha = \left[1+\left(\frac{1-x}{x}\right)\left(\frac{\rho_g}{\rho_l}\right)S\right]^{-1} \tag{1.4}$$

$$S = \frac{u_g}{u_l} \tag{1.5}$$

The slip ratio model is more general than the homogeneous model, because in most cases the slip ratio is not unity. For vertical upward flow and horizontal flow, the gas phase is generally observed to flow faster than the liquid phase due to inertial and buoyancy effects, and as such S>1, with the homogeneous void fraction being the upper limit of possible void fraction values in this case. In downward flow, it is possible that S<1 from the gas phase flowing at a smaller velocity than the liquid phase as a result of the effects of gravity, leading to the homogeneous void fraction being the lower limit in this case. The slip ratio model has been further generalized by Butterworth [29] as follows:

$$\alpha = \left[1 + A\left(\frac{1-x}{x}\right)^{p}\left(\frac{\rho_g}{\rho_l}\right)^{q}\left(\frac{\mu_l}{\mu_g}\right)^{r}\right]^{-1} \tag{1.6}$$

Where *A, p, q* and *r* are constants assigned different values depending on the specific correlation being used, with these values being derived for a given correlation through curve fitting. *Table 1.1* shows three new models which have been shown to perform well on vertical flow [10] [18], as well as the slip ratio and homogeneous model, described using the above format.

The slip ratio model relies on the assumption that the liquid and vapour phases are separated by a definite interface, with both phases flowing at different velocities [10]. Regimes for which this assumption is correct, and which can therefore be described effectively by the slip ratio model include stratified and annular flow regimes, with the above listed correlations [5][21][22] having been shown to perform best on annular flow [18].

2.3 $K\alpha_h$ Correlations

This model type is derived by multiplying the homogeneous void fraction prediction by a coefficient *K:*

$$\alpha = K\alpha_h \tag{1.7}$$

Although when first introduced by Armand [24], this coefficient was a fixed constant, in most other cases this coefficient is found using a function which aims to model and include certain flow characteristics in the calculation of void fraction, such as pressure [30], the two-phase mixture velocity [23] and more [31]. Because these correlations are based on the same assumptions as the homogeneous model, they are applicable to the same flow regimes; those of bubbly, annular, and mist flow [18]. The two best performing $K\alpha_h$ correlations on vertical flow [18] [31] are described in *Table 1.1*.

2.4 Drift Flux Model

The drift flux model and its parameters are of the general form:

$$\alpha = \frac{U_g}{C_0 U_m + U_{gm}} \tag{1.8}$$

$$U_m = U_l + U_g$$

$$U_g = u_g \alpha$$

$$U_l = u_l(1-\alpha).$$

This model assumes that the flow can be described as one phase dispersed within another continuous phase, with the flow patterns best described by this model being bubbly, slug, and mist flows. In this model U_{gm} represents the drift flux, or drift velocity. This value describes the rate at which the gas phase is flowing through a unit plane normal to the channel axis, which is itself flowing at velocity U_m (The two-phase mixture velocity). Calculation of the drift flux depends on the energy transfer and momentum between phases, with various formulations for this value existing across the different models [10]. The value C_0 is known as the distribution parameter, and it describes the distribution of the gas phase in terms of the two-phase mixture within a cross section of the channel [10]. The various drift flux models differentiate themselves in terms of the methods by which the distribution parameter and drift velocity are calculated.

It has been shown that drift flux models produce the most accurate correlations for vertical up flow in two-phase mixtures [18] [31], with the work of Rouhani and Axelsson [26] consistently emerging as a high performing model in predicting void fraction across all flow regimes for vertical up flow [18] [31] [10]. This model has also been shown to effectively describe horizontal two-phase CO_2 flow [13], which is encouraging for its use in modelling vertical CO_2 flow. This correlation, and other high performing models for vertical flow [18] [31] [10], are shown in *Table 1.1*.

Superficial Velocities

These velocities represent a hypothetical flow velocity for a given phase, with the value representing the velocity of that phase in the case where it is the only phase flowing through the channel under investigation. These values are readily known and unambiguous, unlike real velocity which varies across a medium. Superficial velocities are *Equation 2.45* reveals the calculation used in deriving these values.

2.5 Miscellaneous Correlations

Any correlation which does not fit into any of the above categories is considered a miscellaneous or general correlation. Some of these correlations are derived empirically by analysing data sets produced by specific experimental set ups [31], and as such, these correlations are less robust to changes in experimental conditions than models from other categories. Many of these correlations make use of the Lockhart Martinelli parameter [32] [33] [34], which is the square root of the ratio of the liquid pressure gradient to the gas pressure gradient [10]. Because of the turbulent nature of the gas and liquid phases, deriving these gradients is challenging, and as a result, the Lockhart-Martinelli parameter, X_{tt}, is often simplified to the following mathematical expression when included in void fraction correlations:

$$X_{tt} = \left(\frac{1-x}{x}\right)^{0.9} \left(\frac{\rho_g}{\rho_l}\right)^{0.5} \left(\frac{\mu_l}{\mu_g}\right)^{0.1} \tag{1.9}$$

Although these more empirical models of vapour quality have shown success in modelling horizontal flow [31], there have been few such models developed for vertical flow. These equations also represent an older methodology in modelling void fraction relationships, with very few modern correlations being developed by this empirical method.

3 Existing methods used to measure two-phase flow data

The techniques used in measuring the flow characteristics of two-phase flow are commonly classified into two groups: Intrusive and non-intrusive. Intrusive measurements include any measurement technique which requires direct contact with the two-phase fluid to take a measurement, whereas non-intrusive measurements rely on known physical principles to derive measurements from the two-phase flow without any direct contact [35].

3.1 Intrusive Measurements

An intrusive device which can be used for measurement is the U-tube Manometer [36] [37]. This device measures the pressure at two different points in a tube, providing a “differential pressure” for the system, which has been used to calculate void fraction directly, when assuming the two-phase flow is in steady state and flowing with a known homogeneous phase velocity [38]. A wire-mesh sensor [39] specially designed for measurements of two-phase flow has been used to derive myriad parameters from the measured flow, including void fraction [40] [41] [42]. A “cyclone separator” or “cyclone chamber” can be used to separate the gas and liquid phases of the flow to allow for direct void fraction measurement [43]

[44]. A highly accurate intrusive technique commonly used to measure void fraction is the quick-closing-valves (QCV) technique. This method uses a set of valves which can be activated to rapidly isolate a section of the flow, which can then be used for direct analysis. This allows for detailed measurement of the phase composition present in a section of the flow [45] [46] [18].

Some intrusive methods of measuring two-phase flow have become outdated due to the low spatial resolution provided by the intrusive sensors, and due to their requirement of a system compatible with their measurement apparatus, and for their limitations in online measurement. Additionally, many intrusive measurement methods have the potential to disrupt the measurement of the natural flow [35]. Non-intrusive measurement avoids some of these problems because direct contact with the flow is not required for this measurement type.

3.2 Non-Intrusive Measurements

A commonly used and versatile non-intrusive device used is the optical fibre probe [47] [48]. This technique measures a signal in the form of a light intensity reflected onto the probe. This light intensity will change depending on the refractive index of the section of fluid that the probe is observing, and this information is useful in determining the phase composition, bubble size distribution and void fraction of the mixture [49]. A conductive probe can similarly be used to generate a signal describing a cross-sectional slice of the flow for analysis, but instead of light intensity the signal is generated by variations in the voltage level detected by the probe [50] [42] [49]. Although the conductive probe must be in direct contact with the fluid, this technique is considered non-intrusive, because taking measurements using this method, without disturbing the flow, is a well solved problem [50] [42] [49].

Through the application of radioactive source detectors, used to detect the densitometry of X-rays [51] [52] [53] and Gamma-rays [54], non-intrusive flow measurement can be achieved. Although these techniques can produce highly detailed data on the gas-liquid flow being measured, they require specialized measurement equipment and the consideration of radiation safety. Additionally, measuring the void fraction by reading the signal generated through the attenuation of a laser array being emitted through the flow has proven to be an accurate and non-intrusive technique [55] [56] [49].

3.3 Image Processing and Computer Vision techniques for Flow Measurement

With improvements to high-speed digital video cameras, these devices have become a promising non-intrusive tool in the analysis of two-phase flow, as they can produce high temporal and high spatial resolution flow data [35]. With the aid of reflective prisms or mirrors, a single camera can be used to capture multiple angles of the flow at once in order to re-construct the bubbly flow in 3D, allowing for detailed analysis of the flow and its characteristics [57] [58] [59] [60] [61]. Although this stereo imaging approach to the processing of the two-phase flow data produces detailed results, it requires multiple viewing angles into the flow, which is not available for all systems. For the case where there is only one viewing angle available for the observation of the tube, traditional image processing techniques are often employed for the analysis of the bubbly flow [62] [35] [63] [64] [65] [61] [66] [67].

A common first step in the analysis of bubbly flow regimes using image processing techniques is to detect and segment the bubbles within a given image, to measure their geometry. This information can then be used to calculate bubble size distribution [62] [65], void fraction [35], and bubble velocity [67]. Segmenting individual bubbles and bubble clusters from the background of an image can be achieved though thresholding, background subtraction or with edge detection algorithms.

In segmenting these bubbles within the image, a ubiquitous problem is that of detecting individual bubbles within a bubble cluster. These clusters form when bubbles overlap within the image, resulting in connecting boundaries between bubbles being observed within the 2D projection. Many segmentation techniques make use of shape outlines in their separation of items within an image. Since the outline of a bubble cluster is produced by multiple bubbles, the segmentation of these clusters requires the separation of curve sections along the cluster outline in such a way that the unique bubbles within the cluster are effectively segmented. Examples of bubble clusters forming between multiple bubbles can be seen below in *Figure 1.2*.

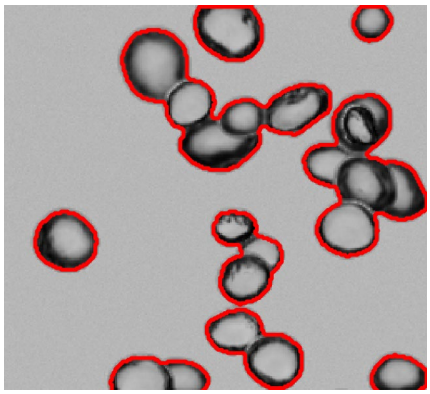

Figure 1.2-Bubble Clusters; Here, the outlines of bubble clusters are shown in red. This image was generated using BubGAN.

One method to segment the bubbles within the cluster is by connecting point (also known as breakpoints) detection [62] [63] [68] [69] [70]. This technique uses various methods to determine which of the points along a cluster outline qualify as connecting points between bubbles. Once the bubble cluster has been segmented into groups of curve segments separated by these connecting points, different algorithms can be deployed to group a given segment with other segments corresponding to the same bubble. With the curves grouped, the shape of the bubble described by these curves can be determined by using the points along the different curve segments with an ellipse fitting algorithm [71] [72]. An alternative method of segmentation utilizes the watershed method to separate overlapping bubbles [73] [35] [64] [65]. A third method of segmenting bubbles within a cluster is by using the circular Hough-transform to detect the bubbles [66], but this technique is less common due to its accuracy decrease for non-circular bubbles. There are also techniques which apply a more integrated approach, utilizing various image processing techniques (such as watershed segmentation, skeletonization, morphological filters, breakpoint detection, adaptive thresholding, etc.) in an algorithm in order to achieve bubble segmentation [35] [64]. These methods typically culminate in the fitting of an ellipse to the grouped segments, with each ellipse representing a bubble. The various algorithms, mentioned above, which are used to measure and segment bubbly flow are explored in greater detail in *Part II, Section 1*.

In recent years, there has been a rise in the application of machine learning and deep neural networks in solving problems of bubble detection and segmentation [74] [75] [76] [77]. Many of these techniques make use of Convolutional Neural Networks (CNNs) [77] [78] [76] [75], a common architecture for deep neural networks used in computer vision tasks.

These networks can be trained to detect individual bubbles, whether they are in a cluster or solitary [76] [77]. This technique therefore bypasses the bubble-cluster problem. In addition to this, the accuracy of the networks can increase with more training data, allowing for potential improvement to a network's

performance. BubCNN represents a pretrained RCNN neural network specially trained to detect bubbles over a range of different images [76]. This network makes use of two cascading CNNs, with the work of Polataev et al. [77] making use of a similar architecture. The first of these networks is an RCNN used to locate individual bubbles, and the second is a shape regression CNN used to estimate the parameters describing each bubbles shape. This network was trained using real world images, as well as synthetic images produced by BubGAN [79]. This diverse set of training data makes the network more robust to variations in image characteristics such as noise and light intensity. Software has been published which allows for a user to input their own set of images and labelled bubble data to perform transfer learning with the network, improving the performance of BubCNN on similar images to those used in the transfer learning [76]. This is a powerful tool for any research into bubble detection, as the accuracy of BubCNN has the potential to be improved even further once transfer learning has been performed.

Because of the visual differences between flow regimes, computer vision techniques can also be used to develop networks which can effectively identify the flow regime present in a given image [80] [81] [82], with a novel implementation of computer vision techniques to classify flow regime described in *Part III, Section 2*. In terms of using traditional image processing techniques to define flow regimes, problems arise because of the complexity of the images produced by high vapour quality flow regimes. Traditional image processing approaches have the potential to classify bubbly flow regimes, by treating the classification as a segmentation problem, but once bubbles are no longer visible in the more complex images, these approaches will fail.

Traditional image processing techniques suffer from a lack of robustness to variations in image characteristics, which could come about from slight changes in experimental conditions. Image characteristics such as noise, colour intensity, image gradients, and changes to image lighting all have an enormous effect on the effectiveness of each algorithm, and as such, creating a generally applicable technique to detect and segment bubbles using traditional image processing poses a challenge, as a small change to experimental conditions could lead to the breakdown of an algorithm. As such, each algorithm developed using traditional image processing must be tuned to specific experimental conditions and image characteristics present in the dataset being analysed. This fact makes it difficult to practically reproduce and compare results seen in literature on real world data, as this would require standardized datasets across the development of the various algorithms. This has led to the testing and tuning of the vast majority of the image processing and computer vision techniques mentioned above being performed using synthetic bubble images, with some of the synthetic images representing bubbles as simple ellipses on a uniform background [76] [62]. This means that many of the above-mentioned algorithms used for segmentation and detection might perform well in ideal scenarios but could suffer when the inconsistencies found in real world images are introduced. BubGAN [78], a tool for developing realistic synthetic images of bubbly flow, can be used to test an algorithm or network on more realistic synthetic bubble images [82] [83]. Although the individual bubbles produced by BubGAN appear realistic, the interaction between bubbles is not modelled by this network, resulting in the contact between bubbles appearing as the overlapping of bubbles, rather than any deformation of the bubbles. This effect can be seen in *Figure 1.2.* Some algorithms have also been designed to take advantage of the consistencies found in BubGAN images, which might not be present in all cases. For example, the work of Fu et al. [34] includes the use of an “inner bubble” which is a characteristic seen commonly in BubGAN bubbles, but not necessarily, or as consistently, in real world bubbles.

Image Features

The above-mentioned methods, and the methods developed in the current work, make use of, and reference to, the concept of image features. It is therefore important to define what is meant by low-level and high-level image features.

Image features are elements found within an image which are used to model a machine's semantic understanding of that image [85]. Low-level image features describe those features which can be directly measured from the image data, such as colour, texture, and shape [85]. High-level image features are more semantically meaningful in an image's analysis, and describe abstract aspects of an image, such as the labels of objects within the image, and the general detection of objects within an image [86].

Traditional image processing techniques make use of algorithms to interpret these low-level image features, and from them, extract useful high-level information found within the image. In contrast, deep learning-based computer vision techniques are used to directly extract high-level image features for further analysis.

4 Experimental Setup

Initially, because of unforeseeable external factors, access to images from the primary experimental environment, found at CERN, was not possible. As a result, a preliminary experimental setup was created to generate bubbly flow images locally. This setup was created so that development and experimentation could begin in terms of the processing of bubbly flow data. This initial test setup generated bubbly flow images in a glass tube by pumping water up through the tube using a 5 W submersible water pump. The pump was submerged in water to about a third of the height of its inlet, so that air was pumped through the glass tube with the water, creating bubbly air-water two-phase flow. The improvements in the processing of these images can be seen below in *Figure 1.3*.

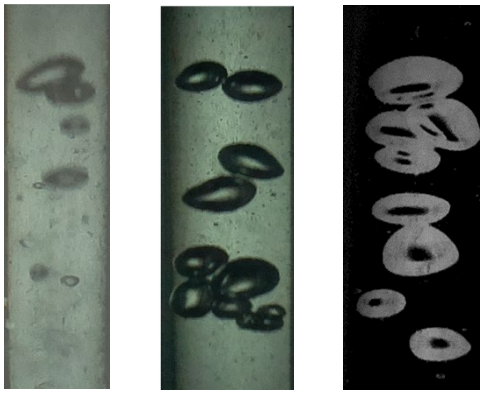

Figure 1.3- Images from the initial test setup; (Left) An image from an uncalibrated camera, (Centre) With effective camera calibration, (Right) With pre-processing.

The details of the primary experimental rig (developed by David Schmidt for CERN) used to generate two-phase flow can be found in [14], with its further application being described therein. This test rig was used to generate datasets of videos and images which are used in the development and validation of the tools described in the proceeding sections of the current work. Photographs of this test rig can be seen in *Figure 1.4.*

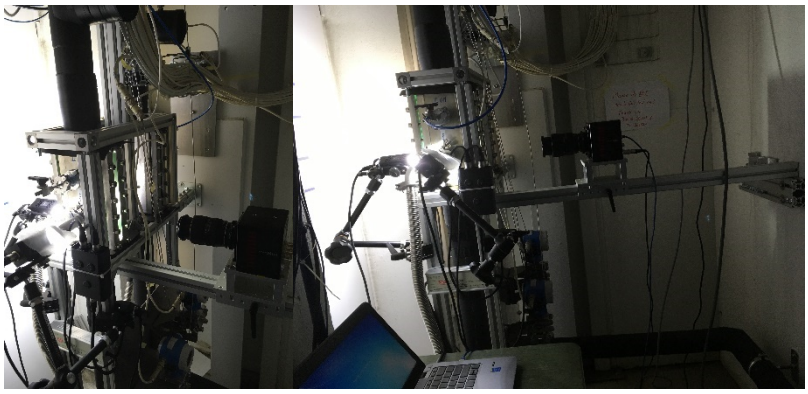

Figure 1.4-Primary Experimental setup; The camera sits a fixed distance from the observable tube section to capture consistent images. Also seen here is the lighting setup to achieve the consistent illumination seen in the images.

The vertical two-phase up flow found in these datasets was captured from images produced by an 8 mm inner diameter and 12 mm outer diameter circular glass viewing section. Constant mass flow rates and fluid specific enthalpies, ranging from 5 g/s to 23 g/s, and 115 kJ/kg to 445 kJ/kg respectively, were adjusted to produce a diverse set of flow images and videos. The pressure of the channel ranged from 1683 kPa to 3970 kPa. The various combinations of mass flow rates and specific enthalpies at the inlet of the test section produce image data representing the full range of possible vapour qualities (from 0 to 1), and all identifiable vertical flow regimes, as can be seen in *Figure 3.2*. The measurement and tolerance of the above experimental conditions is further detailed in the work of Schmidt [14].

The high-speed camera used to capture flow data was the Photron FastCam Mini AX100. A shutter speed of 1/300000 s was used. The use of this fast shutter speed minimizes motion blur. A frame rate of 13600 fps was used. This extremely fast frame rate was used to capture high speed video data, which can then be down sampled to be interpreted at a frame rate that is optimal for a given task. Recording flow data at a high frame rate also allows for greater temporal resolution in its analysis. An image resolution of 512 x 512 was used when capturing the flow data. Consistent backlighting was applied to the test section to retain consistency in the image intensity seen across the flow data captured. The FastCam was fixed at 30 cm from the viewing section in the test rig. A grid pattern, made up of 1 mm x 1 mm squares was inserted into the viewing section to calibrate the relationship between pixels and meters for a given input image. This relationship was found to be 22 pixels/mm, after image distortion due to camera and tube extrinsics were corrected for, using techniques discussed in *Part II, Section 1.1.5*.

In terms of the digital workstation used to carry out image processing and computer vision tasks, the proceeding work was carried out in Matlab [87] and Python [88] environments, using a system powered by a 2.3 GHz Intel® Xeon® E5-2630 CPU, 128GB RAM and an NVIDIA® Kepler™ K40M GPU with 12GB of GPU accelerator memory. The Matlab image processing toolbox [89] was used extensively in *Part II*, while the work done in *Part III* relied heavily on Python libraries such as OpenCV [90] and PyTorch [91].

Part II – Traditional Image Processing

This chapter details all research and work done in developing a tool capable of measuring void fraction from a given input image. The first section explores commonly used image processing techniques and algorithms, which are used throughout the development of the void fraction measurement tool. The development and testing of this tool is detailed in the second section of this chapter. All unprocessed real-world flow images found in this chapter were generated using the primary test rig, described in [14] with all processed image data seen here being original to this work.

1 Commonly Used Image Processing Techniques and Algorithms

Traditional image processing techniques can be used to solve the bubble clustering problem, as well as general segmentation problems within an image of bubbly flow. This section will detail the techniques which can be used in the development of a larger algorithm to achieve effective bubble detection and segmentation, and will expand on some of the techniques mentioned in Part I, Section 3.3.

1.1 Digital Images and Image Processing

A grayscale image can be represented by an intensity matrix $I_0(x, y)$, where x and y represent the coordinates of a pixel along the horizontal and vertical axis of the image, respectively. In all cases here, the images being processed will be 8-bit grayscale images, meaning that the values found in an intensity matrix are between 0 and 255. The ides of "Image Processing" refers to the use of any algorithm or filter applied to an input image in order to achieve a specific outcome. These include, but are not limited to, image segmentation, feature extraction, and noise reduction.

1.2 Pre-Processing Techniques

In image processing and computer vision tasks, pre-processing is essential in ensuring that the input images to the algorithm or network being used ensure the best performance from that method. This process involves adjusting the raw images captured by the camera in such a way to highlight the features of the image which represent useful information for the method being used, while supressing those characteristics which reduce the quality of the methods output. Basic foreground-background segmentation can also be achieved by certain techniques in the pre-processing stage. Some of the common forms of pre-processing, and those used throughout this work are discussed below.

1.2.1 Binarization

Binarization is a fundamental technique in image segmentation and in image data reduction. As such, the use of binarization has an inherent trade-off between a loss in image information, and a useful reduction of image complexity. It is the task of separating the foreground and background of an image in a binary manner as follows:

$$g(x, y) = \begin{cases} 1, & if\ foreground \\ 0, & if\ background \end{cases} \tag{2.1}$$

Here $g(x, y)$ represents a binary image where x and y represent the coordinates of the binary pixel map. This is the basic idea of binarization, that all pixels identified as existing in the foreground are assigned a value of 1, and all background pixels are assigned a value of 0.

There are multiple methods which can be used in determining if a section of the image belongs to the foreground or the background. Many of these techniques involve "Thresholding". This is separating the

foreground and background based on whether a given pixel intensity, $I(x, y)$, has a value above or below a threshold value, *T*:

$$g(x,y) = \begin{cases} 1, & I(x,y) > T \\ 0, & I(x,y) \leq T \end{cases} \tag{2.2}$$

This threshold value can be applied in different ways to the image. It can be applied globally, where the same threshold is applied to the entire image. One popular global threshold selection method is the Otsu method [92]. This method finds the threshold by maximizing the variance in pixel intensity between the two classes of foreground and background. The threshold can also be applied taking spatial information in the image into account. This method is known as "adaptive thresholding", and it can perform better than a global threshold in cases where there are variations in illumination across the image. Adaptive thresholding applies a threshold for a spatial neighbourhood within an image, with the threshold being calculated specifically for that neighbourhood. Adaptive thresholding can be applied on a "per-block" basis, where the image is segmented into blocks with a local threshold being applied to each block. "Local Adaptive thresholding" calculates a threshold on a per-pixel basis using local statistics such as variance, range, and the gradient of a neighbourhood of pixels. A popular method of local adaptive thresholding is the Bradley method [93], which calculates the threshold based on the local mean intensity within a neighbourhood of pixels. Adaptive thresholding can also be used in image segmentation, as it can separate overlapping bodies within an object if the algorithm is tuned correctly [70].

For the task at hand, Binarization can be used to classify bubbles and bubble clusters as foreground objects, effectively extracting the raw boundaries of individual bubbles and clusters of bubbles for further processing. Binarization also greatly reduces the amount of data stored in an image while still maintaining some of the essential structural information of the image. The information produced by this pre-processing step is often used as an input for further processing, as it optimizes the information within the original image for more efficient evaluation.

Figure 2.1 presents an example of image binarization using Otsu's method. In this Figure, some of the failings of binarizing a bubbly image can be seen: jagged, noisy edges are seen for some bubbles, and the centre of the bubbles are segmented as background regions. These issues can be improved or fixed using morphological filtering, discussed in the next section. Note that foreground-background separation algorithms perform much better on the noise-free BubGAN images than real-world images, and as such additional methods must be employed to achieve foreground-background separation in real-world images. This is discussed in proceeding sections.

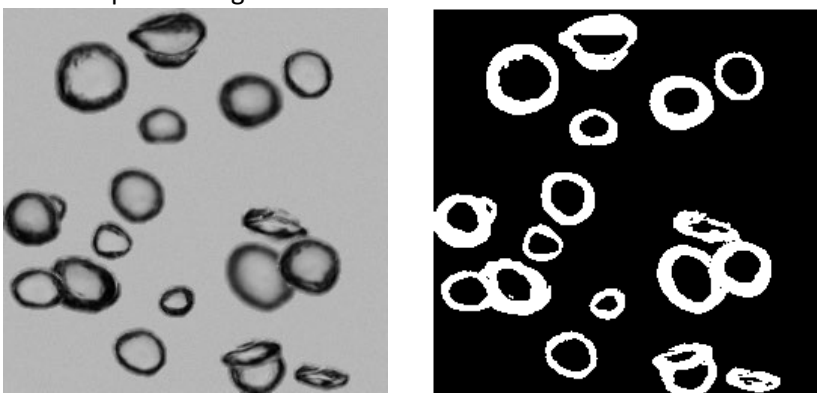

Figure 2.1-Otsu's Method; (Left) Original BubGAN image, (Right) Image Binarized using Otsu's method.

1.2.2 Morphological filtering

Due to the often-imperfect performance of binarization and thresholding, additional corrections may be required to amend for incorrect foreground and background labels. These corrections are applied with the use of morphological filters. These morphological operations are performed by treating the image with a certain structured element. The structured element is an $n \times n$ binary array which is convolved across the binary image in order to adjust the image as is required [94]. Different structured element shapes will adjust the image in different ways, so selecting the structured element for a specific requirement is essential. Additionally, there are multiple morphological operations which make use of the structured element, each with a different effect. There are two base morphological operations upon which others have been developed: Erosion and Dilation.

Erosion is used to remove thin lines and isolated dots which could be false positives. The non-zero (foreground) structures found within an eroded image can always fit within the structures of the original binary image. Instead of reducing the number of positive points defined within a binary image, Dilation increases them. This operation connects unconnected positive regions and fills in false background detections [94]. Other morphological operations include "opening" which involves eroding and then dilating the original image. Opening is used to break narrow bridges and eliminate thin structures. Another morphological operation is "closing", which involves dilating followed by eroding. This operation fuses narrow breaks and eliminates small holes. As opposed to dilating and eroding an image, opening and closing do not change the size of the original foreground region in the binary image. This is useful in cases where the preservation of the original bodies shape is important [94].

Morphological filters in their basic form are tools for improving the binarization stage of pre-processing, but they can be used as parts of more complex operations such as skeletonization, discussed later. *Figure 2.2* presents the application of morphological filters to the binarized image seen in *Figure 2.1*.

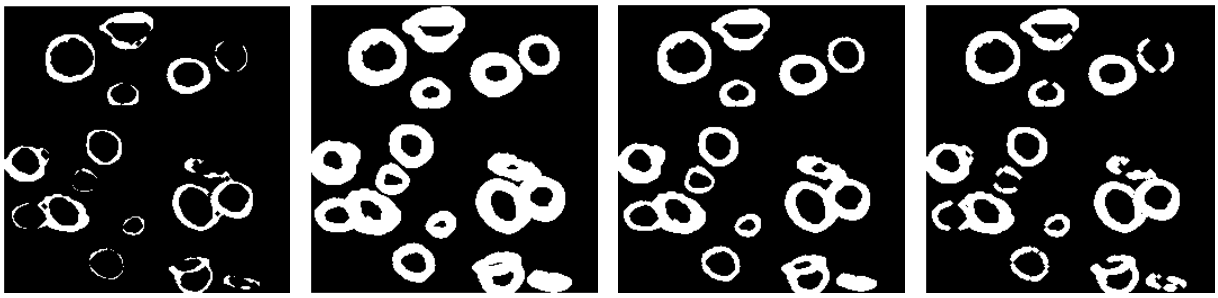

Figure 2.2-Morphological Filtering; (Left to Right) Erosion, Dilation, Closing, Opening. The original image can be seen in Figure 2.1.

1.2.3 Background subtraction

To separate the foreground of an image from the background, an intensity matrix of the background, I_b, (in the case of the present work, this would be an image of a glass tube with only liquid flowing) can be subtracted from I_0 (in the case of the present work, this would be an image of the glass tube with gas and liquid phases flowing). Not only does this highlight the image features which are generated by the gas phase, but it also removes background noise. This subtraction can be represented in the following equation:

$$I_1 = I_0 - I_b$$

Furthermore, the non-uniform illumination seen in the background due to the light source and lens vignetting can be corrected by subtracting the average image intensity of the background from the background itself. This average can be represented by the equation:

$$I_{avg} = \sum_{i=1}^{N}\sum_{j=1}^{M}\frac{I_b(x_i, y_j)}{NM} \tag{2.3}$$

Where N and M represent the height and width of the image, respectively. This average subtracted from the background produces the following intensity:

$$I_2 = I_b - I_{avg}$$

When these values are put together, we see the complete operation of background subtraction, producing the final intensity map I_3:

$$I_3 = I_1 - I_2 = (I_0 - I_b) - (I_b - I_{avg}) \tag{2.4}$$

With the first bracket representing a correction to background noise and the second correcting for non-uniform illumination and brightness within the image.

The process of background subtraction can result in chalky-looking regions with edges which are not clearly defined. A method of solving this problem is to increase the contrast within the image. This can be done by adjusting the intensity values across the output image of the subtraction so that only intensities describing important features of the image are preserved, with the lower intensity chalky blur being removed. *Figure 2.3* shows an original cropped image, an example of background subtraction, with this image corresponding to the final stage of subtraction described above, and an example of background subtraction with contrast correction. This comparison highlights the benefit of contrast correction, as the bubble edges are much more clearly defined in the last picture.

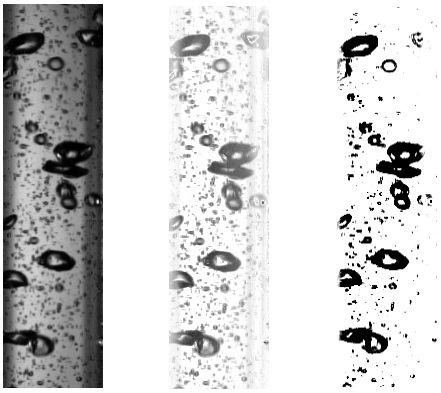

Figure 2.3-Background Subtraction; (Left) Original Image, (Centre) Background Subtraction, (Right) Background Subtraction with contrast correction

Background subtraction is also a means of separating the foreground and background of an image, and as such, can be used in the task at hand to separate individual bubbles and bubble clusters from the background of the image. The advantage of using background subtraction as opposed to binarization techniques is that the subtraction produces a grayscale image rather than a binary one, which contains additional textural information which can be used with proceeding image processing methods, while still effectively separating image foreground and background.

1.2.4 Image filtering

A variety of filters exist which can be used to denoise a given image [95]. This is an important step in pre-processing as the propagation of noise throughout proceeding algorithmic steps can produce sub-optimal results. These filters tend to be spatial low-pass filters, which produce a blurring effect on the image with the attenuation of high frequency signals. A commonly used filter is the median filter, which can be implemented to remove high-frequency noise within an image [62] [63]. The median filter can preserve edges while also removing noise within an image, leading to their wide use across digital image processing problems. This can be seen in *Figure 2.4*. On the left, the original image is shown, with tiny bubbles creating image noise, and poorly defined edges for the larger bubbles. Once this image has been processed using a median filter in the middle image, the edges of the larger bubbles are preserved and better separated from the background, while some of the small bubbles are removed from the image. Bubbles are treated as noise when the resolution with which the camera captures each bubble is not sufficient in representing the curvature of the bubble. This results in blocky bubbles with sharp edges, for which edge tracing algorithms will not be affective. Removing these bubbles effects any calculation of bubble size distribution, but these bubbles cumulatively make up a negligible amount of the gas fraction in the flow, and as such neglecting them in the calculation of void fraction has a negligible effect.

In addition to reducing image noise [95], Gaussian filters can be used to smooth the edges of shapes within an image, which can be effective in improving the results of boundary tracing algorithms [95]. This distortion of an object's shape, from the blurring introduced by this filter, can lead to ill-defined object edges, affecting the results produced by proceeding steps of edge detection and segmentation. The effects of applying a gaussian filter to an image can be seen in *Figure 2.4*. This example shows that some image noise is removed, and object edges become smoother due to a blurring effect. This blurring affects the uncertainty in the determination of the location of a given edge within an image. This increases the uncertainty in a measurement of a given bubbles geometry, as discussed further in *Part II, section 2.4.1*.

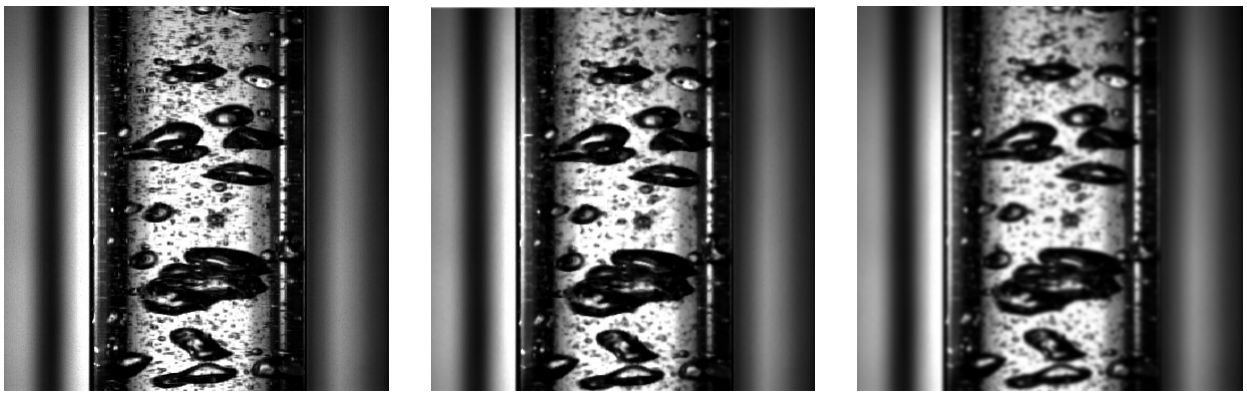

Figure 2.4-Image Filtering; (Left) Original Image, (Centre) Median Filter applied, (Right) Gaussian Filter applied

1.2.5 Distortion Correction and Camera Calibration

Camera calibration is an essential step when converting measurements from a 2D image into accurate real-world units, and for correcting distortion inherent in the lens of the camera. This calibration is achieved by estimating a set of parameters which can transform a set of coordinates in a 2D image into useful real-world coordinates. These parameters correspond directly to properties of the camera lens and image sensor of the camera [96] [97]. Calibrating the input image used in the present work will be essential in generating accurate real-world measurements.

Along with camera distortion correction, there are instances where there is additional distortion within the region of interest in an image. In the present case, distortion from the glass tube which houses the two-phase flow under investigation must be corrected for so that accurate measurements of the two-phase flow can be taken. This can be achieved by accounting for the refractive properties of the glass tube and fluid. Corrections are then made to counteract this refraction, by compressing and stretching regions of the image in order to directly counteract the distortion. The mathematical working used to calculate the degree to which specific regions within the image must be adjusted to counteract this distortion can be found in the work of Fu, et al. [59]

These distortion corrections [59] and camera calibrations [96] can be seen in *Figure 2.5*. While the effects of the camera calibration are small, correcting for the refraction caused by the glass tube produces the stretching effect seen in the second image. These distortion corrections were included when calculating the calibrating factor (22 pixels/mm) between pixels and metres.

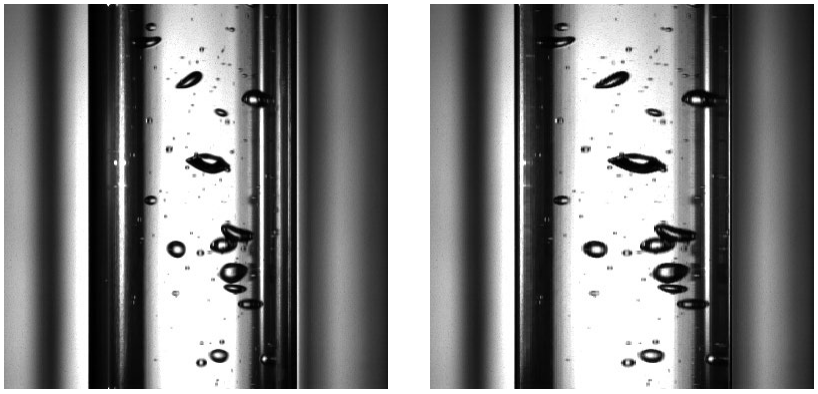

Figure 2.5-Distortion Correction; (Left) Original Image, (Right) Distortion Corrected Image

1.3 Boundary detection and segmentation algorithms

These techniques make use of the optimized data generated in the pre-processing stages as inputs. They produce outputs which can be used within a larger algorithm to achieve bubble detection, segmentation, and parameter extraction.

1.3.1 Watershed segmentation

Watershed segmentation is a technique used to separate two regions which have edges which are touching. This method works by treating a grayscale image like a topographical map, where the darker regions of the image correspond to a greater depth within the image's topography. Conversely, the brighter areas of the image correspond to a higher point within the image's topography, with a pure white pixel representing the highest point within the image. A benefit of using the watershed transform is that it preserves the outer boundary of an object, allowing for segmentation which preserves the outline of the cluster being segmented [98].

There are multiple algorithms used to perform the segmentation itself, with a popular one being "watershed-by-flooding" [98] [99]. This method can be thought of similarly to a real-world scenario where there are topographical basins geographically near to each other. These basins are flooded with water, until the point when the two bodies of water meet. The barrier formed by the meeting of the basins of water corresponds to the line of separation between the two basins. In an image, these basins are the objects being separated, and the line of separation corresponds to a set of pixels which share an intensity

level and are found between the two objects being separated. it is possible that the gradients and intensities of an image do not allow for effective segmentation through watershed transform. A solution to this problem involves generating the binary of the image, and then applying a distance transform to the binary image to create a gradient within the binary, which is then used to effectively segment the image with a watershed transform [100].

*Figure 2.*6 gives an example of this form of watershed segmentation.

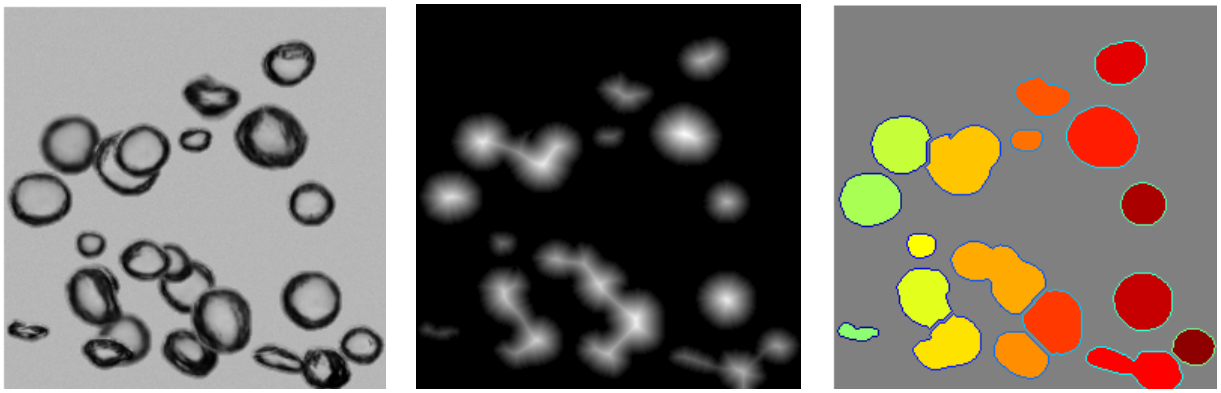

Figure 2.6-Watershed Segmentation; (Left) Original Image, (Centre) Distance Transform of Image, (Right) Watershed transform

1.3.2 Skeletonization

Skeletonization is a method which reduces objects within an image to a shape-centred coordinate axis, which can be used to separates distinct regions within the objects [101]. This is achieved using morphological operations on a binary image, which reduce objects in the image to a finite set of connected skeletons, which form the coordinate frame of the greater object [102]. This coordinate frame, which represents a connected medial location along an image binary, can be seen in *Figure 2.7.*

By performing a XOR operation between the image skeleton and the image binary, the image shape is preserved, while each binary object is effectively segmented by the object's skeleton. This operation is useful for bubble segmentation because these separated regions represent individual segmented bubbles from within a bubble cluster [70] [64], as is seen in *Figure 2.*7. Additional morphological filters are applied in generating these images to fix segmentation errors from the XOR operation.

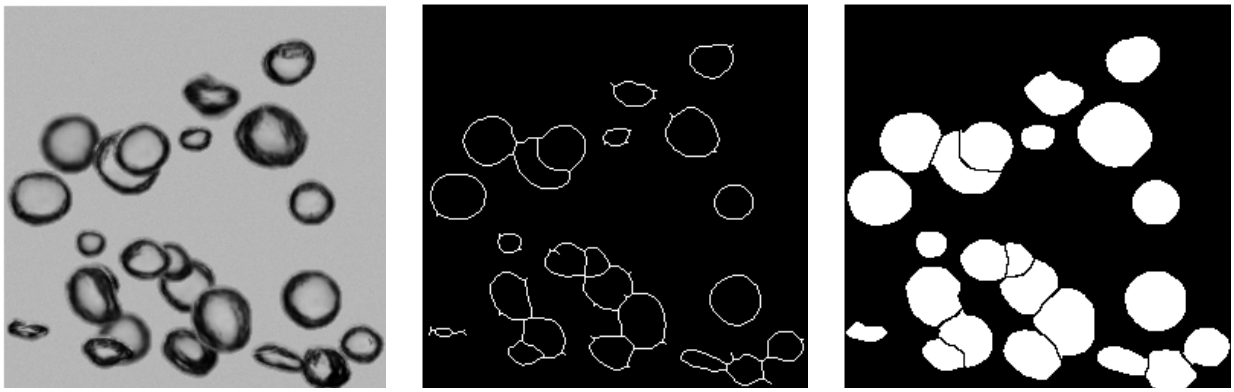

Figure 2.7- Skeletonization; (Left) Original Image, (Centre) Coordinate Frame/Image Skeleton, (Right) Segmented image using skeletonization

1.3.3 Edge detection

In segmentation it is often useful to obtain information about the edges and boundaries of an image. This information inherently segments the image into a set of separable boundaries, which can be treated and classified as individual segments within the image. Edge detection is also useful in the reduction of information to be processed in computer vision tasks.

Edges of an image can be highlighted through simple convolutional operations, such as the Sobel operator [103] or the Prewitt operator [104]. These individual operators can be used to detect vertical, horizontal and diagonal edges, but lack the fidelity to detect multiple edge orientations with the same operation.

A more complicated filter can be applied to the image to detect multiple types of edges with the same operation. One such operation is the Laplacian of Gaussian (LOG) edge detector. This method applies the Laplacian of a Gaussian mask to an image and using a specific algorithm, can produce complex edge detections within an image [105].

Edge detection is fundamentally related to the gradient information of an image. This image gradient can be used to describe both an edges direction and its magnitude. A popular edge detection technique which makes use of the image gradient is the Canny edge detector [106]. This operator applies non-max suppression (an algorithm for selecting one entity out of many over-lapping entities) to the magnitude of the gradient to preserve the strongest edges in any direction. The Canny edge detector also makes use of a threshold for the strength of the edges which are preserved by the algorithm. This is a customizable value, allowing for effective tuning of the Canny edge detector for a wide range of images. The Canny edge detector produces an image binary, which allows for the adjustment of its output using morphological filters. The Canny edge detector has been used in bubble detection at various stages across a range of methodologies [70] [74]. An example of the application of a Canny edge detector can be seen in *Figure 2.8*. This edge detection provides no segmentation for a given bubble cluster, but it preserves the boundary shapes of individual bubbles and bubble clusters well.

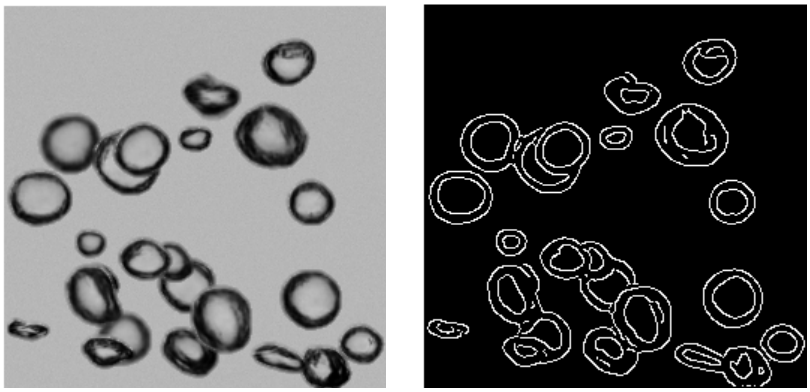

Figure 2.8-Canny Edge Detection; (Left) Original Image, (Right) Canny edge detection

From a binary image, edges can be extracted using line tracing algorithms to follow the outline of a given object. A commonly used boundary following algorithm is the Moore-Neighbourhood tracing algorithm. This method makes use of the Moore Neighbourhood, which relates a single point in a square lattice to its eight surrounding points [107]. The tracing algorithm makes use of this neighbourhood to move around the boundary of a shape until a stopping criterion has been met which ends the tracing process. An effective stopping criterion is the Jacob's stopping criterion [107], which ends the tracing process when

the starting pixel has been traced by the algorithm for a second time, in the same way it was passed through by the algorithm initially. This boundary tracing technique has been used to trace the boundaries of bubbles and their respective clusters in similar works [63].

1.3.4 Ellipse fitting

Due to the tendency for bubbles in vertical two-phase flow to form in ellipsoidal and spherical shapes, as a result of surface tension and hydrodynamic forces [15] [11], the ellipsoidal reconstructions of a given bubble can be used to investigate its shape and size, as well as define its outer perimeter. The fitting of an ellipse, described by a finite set of clearly defined parameters, to an object within an image, results in a massive reduction in the data used to describe the object, while preserving key information regarding the objects shape and size. Within a 2D image, this is achieved by applying one of a multitude of ellipse fitting algorithms to a bubble's boundary, extracted through the proceeding steps of a larger bubble detection and segmentation algorithm.

One such algorithm is the Circular Hough Transform (CHT) [108]. This transform uses "voting" in the Hough parameter space to extract the parameters of circles with an image. This transform is effective at the detection and reconstruction of circular (spherical in 3D space) bubbles but fails to reconstruct ellipsoidal bubbles. The Randomized Hough Transform (RHT) is a probabilistic variant of the Hough transform which can be used to fit ellipses as well as other curves within images [109]. While this method is effective, it is computationally expensive and requires a large amount of memory. The RHT has been optimized for efficiency and data storage in subsequent works [110] [111]. Another approach is to use least-square fitting to fit an ellipse to the shape of the boundary [72]. This method benefits from computational efficiency and robustness to noise.

While approximating a curve by minimizing least square error function has proven to be an effective curve fitting technique, this approximation evaluates the error as a distance along the abscissa along a cartesian plane, but ignores differences between points along the ordinate. Taubin's method [71] uses the exact geometric distance between a point and the first order approximation of the surface being fit to it when minimizing error, resulting in a more accurate and robust (due to the methods invariance to rotation) estimation of a 2D or 3D surface from a set of points. This fitting technique is known as the generalized eigenvector fit, and it is easily computed in real time due to its computational simplicity, while still achieving an effective minimization of the approximate mean-square error. The generalized eigenvector fit is also rotation invariant, making it an excellent curve fitting technique for image processing tasks. Ellipse fitting using Taubin's method can be seen in in figures throughout *Section 2.2.3* of this chapter.

Another method of direct ellipse fitting is proposed by Nayak and Stojmenovic [112] whereby the parameters of an ellipse are determined by estimating the optimal locations for the foci of the fitted ellipse in relation to the set of points the ellipse is being fit to. The location of the foci are found by minimizing the variance of the sum of the distances between each foci to each individual point being fit. The set of foci which have the least variance in their combined distance to each point being fit are selected as the foci of the ellipse, and from this information the orientation and axis lengths of the ellipse can be calculated directly.

1.3.5 Polygonal approximation

Polygonal approximation is the process of simplifying a shape within an image by reducing the number of points used to describe the shape, while still preserving the key features of the shape. This effectively

approximates the original, more complex shape, to a simplified polygon, made up of lines connecting the set of reduced points. The polygonization of a shape produces an approximation which has a reduced data size compared to the original shape, and which allows for the simplified analysis of an object's geometric features, such as the curvature between sections of the polygonal approximation, and information regarding the concavity or convexity of the approximated curves [113] [63].

For digital curves, there are two main approaches in achieving polygonal approximation. The first approach is to fit a sequence of line segments to the original shape by reducing a certain error criterion or optimality constraint [114] [115]. The second approach defines the polygon about a subset of dominant points which exist on the original shapes boundary [113] [116]. One such algorithm for polygonal approximation using dominant points is Poyato's method [113]. This method tends to include points of high curvature in the set of dominant points, because the algorithm makes use of the following formula, first used in the work of Teh and Chin [117]:

$$r_{ik} = \frac{d_{ik}}{l_{ik}} \tag{2.5}$$

This formula can be seen as a digital analogue to curvature [113]. For each point p_i, which belongs to a set of points describing a curve in an ordered sequence, l_{ik} represents the length of the chord between p_{i+k} and p_{i-k}, and d_{ik} is the shortest distance between p_i and that chord. A visual representation of these values can be seen in *Figure 2.9.*

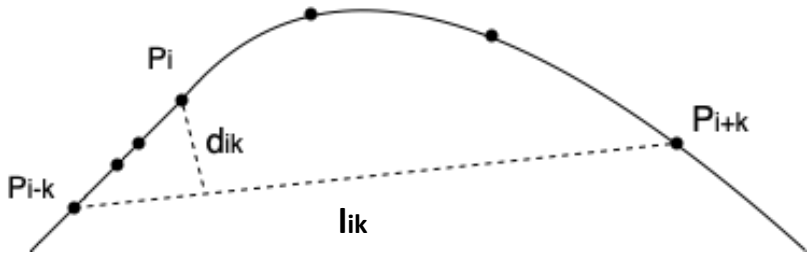

Figure 2.9- A visual representation of the values used when calculating The and Chin's curvature for point p_i.

Poyato's method works to delete all points along the curve for which r_{ik} is below a specific threshold, effectively keeping only points of high curvature, in terms of this metric. This algorithm has a stopping condition when the maximum r_{ik} value for a reduced set of point, r_{max}, is above a certain threshold. The ability of Poyato's method to adjust this stopping condition, being the maximum point curvature, makes it a more robust and customizable method of approximation. By increasing this threshold, fewer lines are used in generating the polygonal approximation of a set of points. *Figure 2.10* shows the effect of adjusting the stopping condition in Poyato's algorithm.

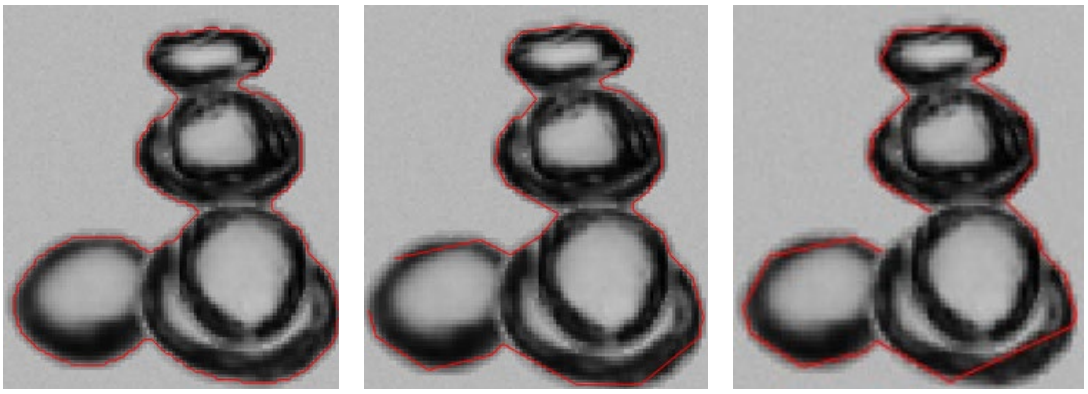

Figure 2.10- Poyato's Method; The stopping condition, r_{ik}, is increased from left to right (with values of 1,4, and 8 for each respective image).

For bubble segmentation, the polygonal approximation of a cluster of bubbles can be used to separate bubble line segments on the boarder of the cluster. This is achieved by separating the line segments about concave polygonal breakpoints, which represent the points where two bubbles touch [63]. The polygonal approximation also provides information on the curvature of a bubbles line section, as well as allow for simplified shape reconstruction by reducing the number of significant points included in the ellipse fitting process.

2 Development of a novel algorithm for Void Fraction Calculation in bubbly flow

The present section details the development of a novel non-intrusive methodology to calculate the void fraction of bubbly flow, using image processing techniques, and images from a single high-speed camera. This algorithm takes in a raw image of bubbly flow as an input, and outputs a void fraction calculation, making it an end-to-end approach for extracting void fraction information from an input image. The final algorithm also outputs an estimate of the range of uncertainty within which the final void fraction calculation lies. Code for the work presented here can be found by following the GitHub link in the appendix.

This algorithm can be executed in the following sequence:

1. Image pre-processing: The relevant pre-processing techniques are applied to the input image to extract the useful features needed for further processing, as well as for data-reduction purposes. Corrections to the input image are also applied to reduce uncertainty in later steps.

2. Bubble Segmentation and Parameter Estimation: The bubbles and bubble clusters are detected and segmented within the processed image. Additionally, the geometric parameters of the segmented bubbles are approximated in this step.

3. Bubble Volume calculation: The extracted parameters that describe the bubbles cross sections are used to calculate individual bubble volumes. These values are calculated with corresponding uncertainties.

4. Void Fraction Calculation: A final calculation of void fraction is found, including all uncertainties. This calculation is derived from the volumetric definition of void fraction, with the final value

corresponding to the sum of the bubble volumes (the gas fraction) divided by the calculated volume of the channel section being observed (the total volume).

The Bubble Segmentation and Parameter Estimation step will be detailed below using two different implementations. This step is the most expansive and challenging in the larger algorithm due to the challenge in segmenting a cluster of bubbles. All the above-mentioned steps are detailed and expanded on in the following sections. They are implemented using Matlab's image processing toolbox [88], unless otherwise stated.

2.1 Image Pre-Processing

There are some fundamental pre-processing steps applied to the input image that are required for each of the bubble segmentation techniques. These steps are applied to reduce the uncertainty in measurements taken from the image, by calibrating to correct for distortions, and by removing regions of high uncertainty. Foreground-background segmentation is also achieved in this stage of the greater algorithm. The image pre-processing steps are carried out in the following order:

First, the camera parameters are derived to calibrate the image in accounting for the physical properties of the camera lens and image sensor. These parameters are used to undistort all the images taken with the FastCam, through the correction of the distortions inherent to that camera [96] [97]. The extraction of these camera parameters, and their application in the relevant algorithms [96] [97], is done through Matlab's Camera Calibrator application [89]. In addition to the correction of the camera distortion, geometric equations are applied to correct for the warping of the fluid within the glass tube which comes about because of the refractive properties of the tube, with the required equations found in the work of Fu et al [59], and applied to an image using the OpenCV [90] Python library. With these distortions corrected for, the processed image represents a 2D, undistorted frontal view of the flow within the inner tube, as seen in *Figure 2.5*.

Next, the image is cropped to removed blacked out regions of the glass tube. This is done to remove those regions of the tube which are left without illumination due to the refractive properties of the tube. Although these regions might contain bubbles, their shapes cannot be determined with any certainty. The cropping of the image changes the volume of the tube section being observed. This fact must be accounted for in the final void fraction calculation carried out in the final step of the overarching algorithm.

Finally, a background subtraction and contrast correction are applied to the image, as described in *Part II, Section 1.1.3*. This requires a background image of the flow channel where there are no bubbles flowing. *Figure 2.11* shows a before and after comparison for an image in the pre-processing stage. This comparison shows how pre-processing effectively separates the larger bubbles from the background of

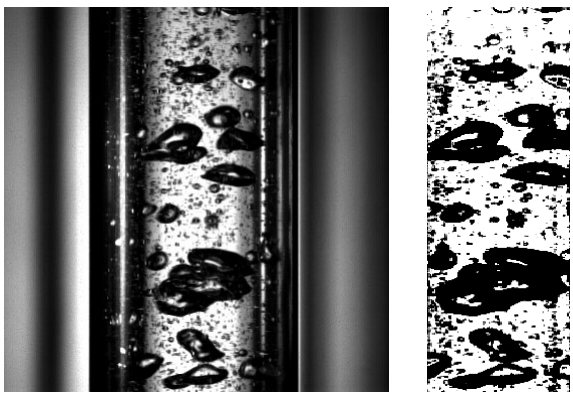

Figure 2.11-Fundamental Image Pre-Processing; (Left) Original Image, (Right) Processed Image

the image, while still preserving their shapes. Distortions from the tube and camera lens are corrected for, producing the wider bubbles seen in the image on the right. Additionally, dark regions, where any bubble shape estimation would have high uncertainty, have been cropped out.

2.2 Bubble segmentation and Parameter Estimation

An image outputted by the pre-processing step has a clear separation between foreground and background, yet any overlapping bubbles lead to amorphous bubble clusters in the foreground. These clusters make it difficult to determine the shape and volumes of the individual bubbles found within them. Effective bubble segmentation will separate the individual bubbles within the larger bubble cluster, allowing for more accurate volumetric calculations in the proceeding algorithmic steps.

As mentioned above, two different bubble segmentation techniques were explored: The use of BubCNN to detect and segment bubbles within an image, as well as a technique which makes use of breakpoints to achieve segmentation. These techniques also estimate bubble parameters by different methods, as discussed below. An integrated method, which made use of multiple segmentation techniques was explored, but was abandoned early for reasons described below.

2.2.1 Integrated method of bubble segmentation

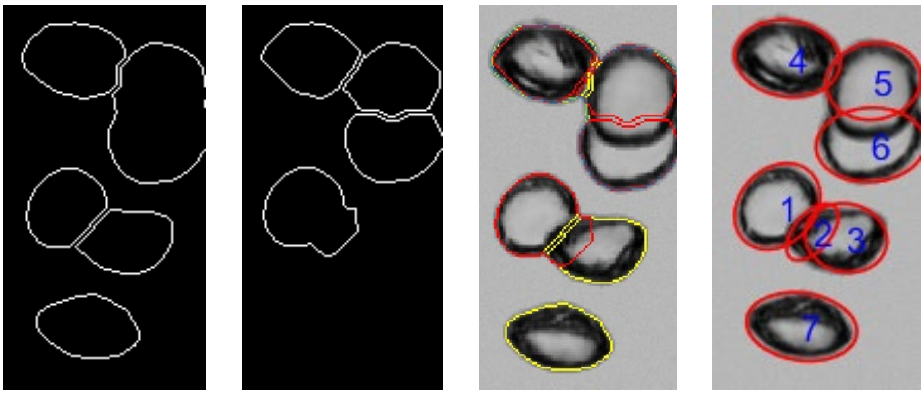

Figure 2.12-Integrated Method of Bubble Segmentation; (Left to Right) Watershed segmentation, Skeletonization, Overlaying of segmentation techniques, and Ellipses fit to segmentations

The first attempt in using image processing techniques to achieve bubble segmentation was based largely on the work of Fu et al. [70], whose work in turn was inspired by Lau [65] and Karn [64]. The idea of this technique is to use a variety of segmentation techniques, such as watershed segmentation, adaptive thresholding and skeletonization, to separate individual bubbles from a larger bubble cluster. The use of multiple segmentation techniques allows for a more robust segmentation algorithm, with the segmentations missed by one method being identified in another. A larger algorithm must be used to determine which segmentations are kept and which are ignored, as, depending on which image processing techniques are used, there can be an excessive number of segments being identified. In the attempt made to develop a similar algorithm, watershed segmentation and skeletonization were implemented to extract segment boudaries from an input image. *Figure 2.12* shows these segmentations, the result of overlaying these segmentations on the original input image, and the final ellipse estimations using Taubin's method [71].

Although this method is a promising option for achieving bubble segmentation, primarily for its ability to detect segmenting lines within a bubble cluster, it was abandoned due to the complexity inherent in

developing an algorithm to manage and select between bubble segmentations. An excessive number of false positive bubble detections occur when the two segmentations overlap to form a new, incorrect segmentation, as is seen in the image above, where the ellipse labelled “2” (between “1” and “3” in the image) represents a false positive.

Examples of this issue occuring on a real-world image can be seen in *Figure 2.13*, where an exessive number of bubble segments are detected for one large bubble, causing a high number of false positive bubble detections. While developing a novel algorithm which succesfully selects between these overlapping segments is possible [70], the prospect of developing such an algorithm in the current work was abandonned as the development of the Breakpoint algorithm (discussed in *Part II, Section 2.2.3*) continued in parallel without issue.

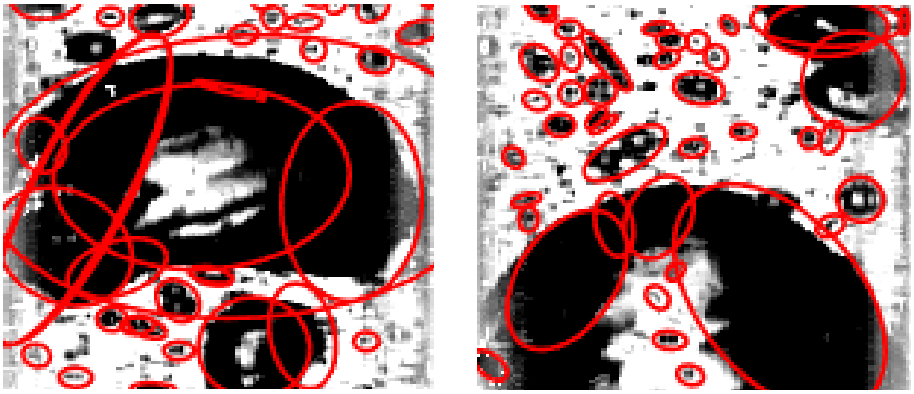

Figure 2.13-The Integrated approach applied to real world images

2.2.2 BubCNN for segmentation and parameter estimation

The BubCNN tool [76] represents a machine learning approach to bubble segmentation and parameter estimation. This network makes use of two cascaded CNNs. The first of these networks is an RCNN used to locate individual bubbles, and the second is a shape regression CNN used to estimate the parameters describing each bubbles shape.

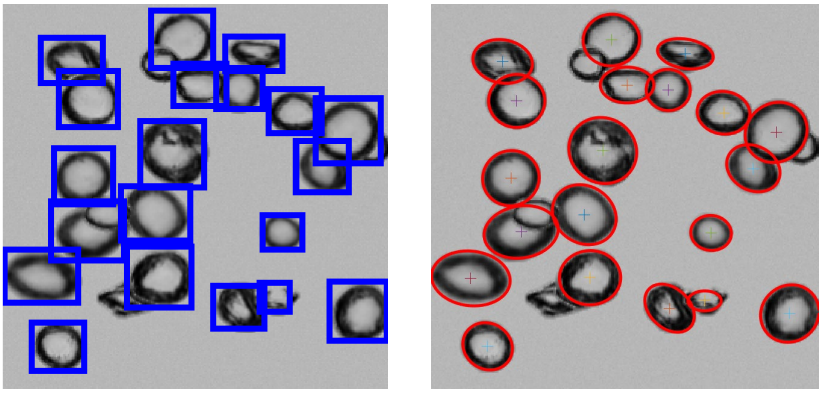

Figure 2.14- BubCNN applied to a BubGAN image; (Left) Bubbles detected by the RCNN network, (Right) Ellipse shapes found using the Regression CNN

The output of each of these networks can be seen in *Figure 2.14*. In this figure, it is evident that BubCNN has the capability to segment bubbles which exist within the same cluster. The transfer learning tool of BubCNN was used to expand the training data set of the network to include bubble images and parameters from the testing environment of the current work. This will improve the performance of

BubCNN on images generated by this testing environment. 100 varied images of bubbly flow were added to BubCNNs training set, through the manual labelling of individual bubbles within the image set using the BubCNN transfer learning module.

2.2.3 Breakpoint segmentation and Parameter Estimation

The third approach developed to achieve bubble segmentation makes use of breakpoint detection and polygonal approximation to achieve bubble segmentation. This algorithm is based upon the work of Zhang et al. [63] with novel additions being made to that work. This Breakpoint Segmentation algorithm can be executed in the following sequence:

1. Pre-Processing: Additional Pre-Processing is applied to the image to highlight the concave points along the cluster outlines and remove concavities which come about because of noise in the image.
2. Contour Segmentation: Sets of points making up individual contours are segmented along the bubble cluster-outline. This step is achieved by deriving a polygonal approximation of the cluster outline and separating the contours of this approximation about the concave points of the outline.
3. Segment grouping: The separated contours are then grouped together based on a set of simple conditions with the aim to group contours belonging to the same bubble found within the bubble cluster together.
4. Estimation of occluded contours: The contour segments grouped on the perimeter of the bubble cluster do not represent the section of the bubbles which exist within the bounds of the cluster's outer perimeter. Therefore, an estimation must be made using these outer segments to predict the shape of the bubble within the bubble cluster.
5. Parameter Estimation: From the grouped bubble segments, the parameters of the ellipse which approximate the individual bubbles can be derived through ellipse fitting techniques.

These steps are elaborated upon below.

1. Pre-Processing:

The pre-processing stage is applied to the input image with the aim to remove concavities along bubble cluster outlines which are a result of noise, and to accentuate the true concave points along the cluster outline. Additionally, further foreground-background segmentation is implemented, as well as a small degree of bubble cluster segmentation.

First, the input image is blurred using a Gaussian filter. This blurring smooths the edges of the bubble clusters, removing small concavities which may exist along the cluster outline because of image noise, or as a result of gradient inconsistencies produced by the background subtraction implemented earlier. The standard deviation of the kernel used to apply the Gaussian filter was set to a value of 0.5, producing only a slight image blur. This slight blurring of the image preserves the structure of the bubble clusters, while smoothing their edges.

Next, foreground-background segmentation is achieved using Otsu's Method [92], producing a binary image where the bubble clusters appear as silhouettes of the bubbles that they represent, with only the

outline of the clusters being preserved. An inner bubble may appear after binarization, manifesting as a hole within the cluster outline. These holes are filled using morphological filtering.

Finally, a distance-based watershed transform [100] is applied to the binary image to achieve a level of segmentation within the binary, separating a small set of bubbles from the bubble clusters. The watershed transform was used over skeletonization techniques because the latter tended to drastically over-segment real world data.

After this stage of processing, the image data has been reduced to an image binary, with noisy concavities removed and larger concavities maintained, and with a small level of segmentation having already being achieved. The results of this pre-processing step can be seen below in *Figure 2.15* and *Figure 2.16*, where computer generated BubGAN data and real-world images are processed respectively.

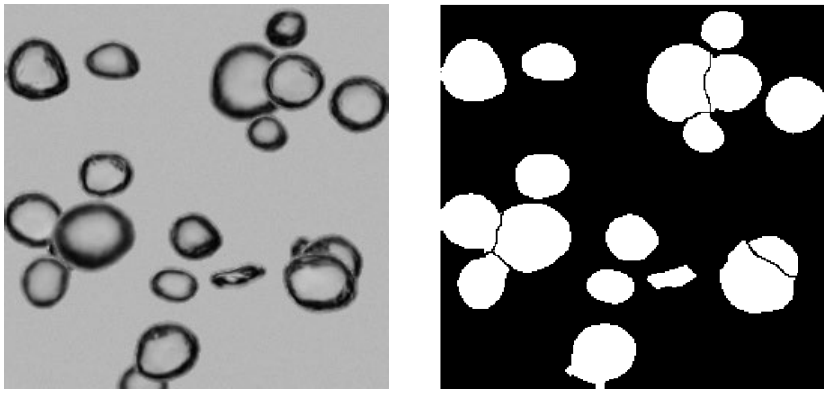

Figure 2.15- Pre-processing for the Breakpoint algorithm (BubGAN images); (Left) Original Image, (Right) Processed Image

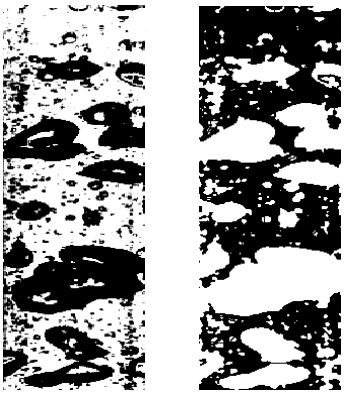

Figure 2.16- Pre-processing for the Breakpoint algorithm (Real-world images); (Left) Original Image, (Right) Processed Image

2. Contour Segmentation

In this step, the contours around a bubble cluster outline are segmented about the concave points which lie on the boundary outline. It is assumed that these concave points are the meeting points of two different bubbles, and therefore each contour connected to such a point belongs to a separate bubble.

The first stage of this contour segmentation is the identification of the various bubble cluster boundaries within the binary image. A Moore-Neighbourhood tracing algorithm [107] is used to find a set of

boundaries within the image. Each of these boundaries represents either an individual bubble or a bubble cluster. This tracing algorithm stores the boundary information as a set of coordinates, with each pixel location along the boundary perimeter being stored as an individual coordinate. This can lead to an excessive number of data points being used to representing the cluster boundary. Additionally, it is more difficult to extract points of high concavity along such a curve, because these regions are represented by a set of vertices with small curvature magnitudes, rather than one distinct vertex with a large curvature value. To solve these problems, a polygonal approximation of the bubble cluster is performed.

For this polygonal approximation, Poyato's method [113] was implemented, with this algorithm being selected for its ability to preserves points of high curvature. By preserving points of high curvature, points of high concavity are included in the final approximation. This is required for the next step of the segmentation algorithm. Additionally, the ability to adjust the stopping condition of Poyato's method allowed for useful customization during the development of the larger segmentation algorithm. For its final implementation within the segmentation algorithm, Poyato's method was implemented with a stopping condition of $r_{max} > 1$. r_{ik} values less than 1 are found for points on curve segments which have a very low convexity, and thus, this threshold removes points which are redundant in describing the shape of the larger bubble cluster, such as those existing in the middle of a straight line. This threshold therefore achieves effective data reduction of the original contour while still maintaining an acceptable resolution when approximating a cluster outline.

The reduced set of points must then be arranged in a counter-clockwise order for the proceeding concave point detection. To determine whether a breakpoint p_i is concave or convex, the cross product of the vectors to and from p_i is taken, and if this cross product is negative, the breakpoint is considered concave. This condition is reliant on the counter-clockwise ordering of points, and can be expressed as follows:

$$\overrightarrow{p_{\iota-1}p_{\iota}} \times \overrightarrow{p_{\iota}p_{\iota+1}} < 0 \tag{2.16}$$

With *Figure 2.17* providing a visual representation of this condition.

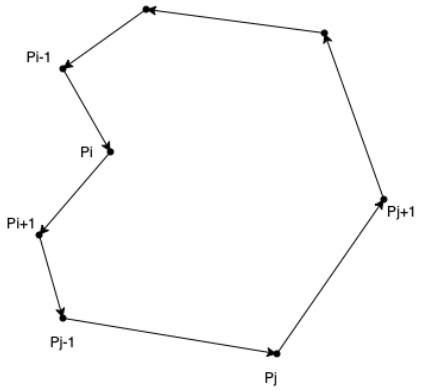

Figure 2.17- Visual representation of the condition used to detect concavities; With $\overrightarrow{p_{\iota-1}p_{\iota}} \times \overrightarrow{p_{\iota}p_{\iota+1}}$ being the only negative cross product between vectors along this set of points, it can be seen that vectors describing concavities for a set of points in a counter-clockwise order have a negative cross product, while all other cross products will be positive.

The larger cluster outline is then segmented into a set of curves separated by these concave points. In *Figure 2.18*, the concave points are circled in yellow, with segments being separated about these points. This figure shows that bubble curve segments can successfully be segmented from one another by taking segments around these breakpoints.

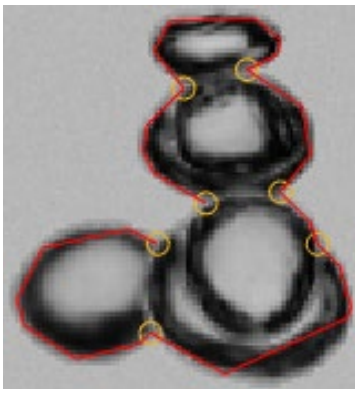

Figure 2.18-Contour segmentation using Breakpoints (circled in yellow)

3. Segment Grouping

The method by which individual segments are grouped together is based upon the method implemented by Zhang et al. [63] except where Zhang uses the *Average Distance Deviation* as a measure of how well an ellipse fits a set of points, the present algorithm makes use of the final root mean square error (RMSE) after Taubin's method [71] has been applied to fit an ellipsoidal shape to a set of points. Taubin's method was used to calculate the RMSE of an ellipsoidal approximation to a set of points for its approximation accuracy, computational speed, and rotation invariance.

Each curve segmented along the bubble cluster outline is fit by an ellipse using Taubin's method, and the subsequent RMSE is taken from the approximation, and assigned to its respective curve segment, S_l, as $RMSE_l$. In determining whether to combine S_l with another segment, the following steps are executed: The points in segment S_l are combined with the points from each other segment along the cluster outline individually, with the segment being combined with S_l assigned the label S_k, and the combination of the two segments being assigned S_{lk}. These two other sets of points are approximated using Taubin's method, generating $RMSE_k$ and $RMSE_{lk}$. The condition under which two segments S_k and S_l are grouped together as S_{lk} is as follows:

$$RMSE_{lk} \leq \frac{RMSE_l + RMSE_k}{2} \tag{2.17}$$

If this condition is true, the two segments are grouped together, and are assumed to belong to the same ellipse. This condition is true if the combined segment is better approximated by an ellipse (as measured by its RMSE) than the two separate segments individually. This condition is effective for most cases, but it must be mentioned that it has a bias towards not grouping curve segments together in cases where one or both curve segments have only a very few number of points describing them. This is because Taubin's method is able to fit an ellipse to a smaller set of points with a lower RMSE than a larger set of points, so in the case where one or both curve segments have only a very few points describing them, the above condition is unlikely to be true. That said, it was observed that for the majority of curve segments, this was not a problem, with both the left and right side of the above condition being of very similar magnitude, no matter if the condition is true or false. This indicates that in general, the curve segments being grouped from the data used here are represented by a sufficient number of points for this not to greatly impact the performance of the segment grouping methodology.

This process is followed for each segment, and once segments have been grouped, the entire grouping process can be executed again to potentially group 3 of the original segments together. The effectiveness of segment grouping can be seen in *Figure 2.19*, where two separate segments from the same bubble are successfully grouped together.

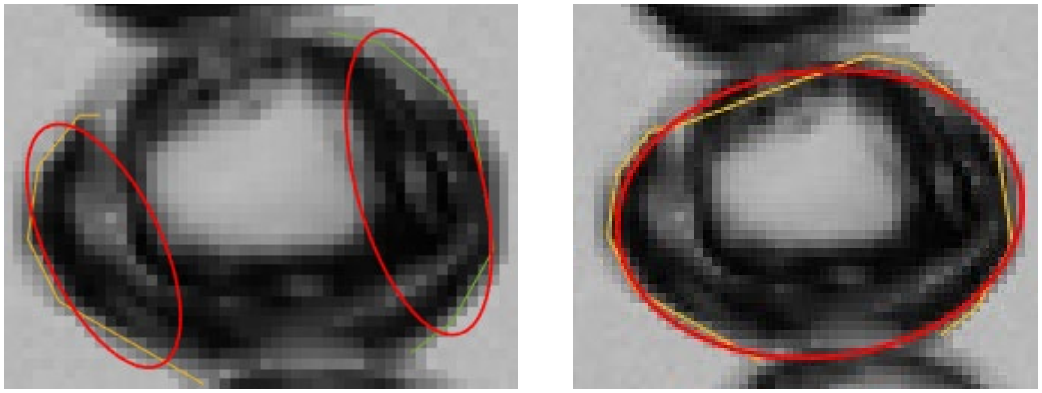

Figure 2.19-Segment grouping; (Left) Two contour segments belonging to the same bubble are incorrectly approximated by separate bubbles, (Right) The two segments are grouped together, allowing for a more accurate, single bubble, approximation

4. Estimation of occluded contour and Ellipse Parameter Estimation

In certain cases, a large portion of an individual bubble's outer perimeter can be represented by the grouped segments identified in the previous steps. However, it is common for only a small portion of a bubble's outer perimeter to be represented by these segments, with most of this curve existing within the borders of the initial bubble cluster outline. Because the bubble cluster was identified from a foreground-background binary segmentation, the interior of the bubble cluster contains no useful information for bubble shape prediction. As such, an estimation must be made to determine the shape of the occluded contour within the bubble cluster outline.

In some cases, using Taubin's [71] method of curve fitting can produce well fit ellipses to even a small curve segment from a larger bubble, with the ellipse being fit to the segment accurately predicting the shape of the occluded section of the bubble. In these cases, the estimate produced by Taubin's method is used as the final ellipse parameter estimate, and the algorithm ends here, where a set of ellipse parameters are produced to describe the bubble by Taubin's method. However, a supplementary algorithm is introduced to improve the performance of Taubin's method when the best fit ellipse to a segment has an RMSE above a certain threshold.

It was observed that Taubin's method can achieve an RMSE less than 1 for the vast majority of cases when fitting an ellipse to a set of points along a curve segment when the curve segment does not have any major concavities or long straight lines. This means that Taubin's method can fit an ellipse to points on a round, consistently convex curve segment with an average error less than one pixel in terms of the location of each point. Concavities and long straight lines are both common characteristics of bubble curve segments which have occluded sections, and as such, using this metric is an effective way of identifying when a curve segment has an occluded section. A threshold of $RMSE > 1\ pixel$ was therefore found to be an effective trigger for when to apply the algorithm described below, as the algorithm is triggered in cases where there is likely an occluded section of the bubble.

This new algorithm works to estimate the location of, and consequently add, a new point, which lies within the bubble cluster, to a segment group in such a way that when Taubin's method is applied to the new segment group, the RMSE value for the new segment group is lower than the RMSE belonging to the group before the new point was added.

An example of this point placement can be seen in *Figure 2.20* below. In the image on the left, Taubin's method fails to effectively estimate the shape of the occluded bubble segment within the bubble cluster, resulting in a shape estimation with a high RMSE. The image on the right shows the improvement to this estimation, achieved by adding a new point (circled in blue) to the original set of points, by using the point placement algorithm detailed below. Once the point placement algorithm has been executed, and a new point has been added to the set of points describing a bubble, Taubin's method is applied to extract the final ellipse parameters of the bubble.

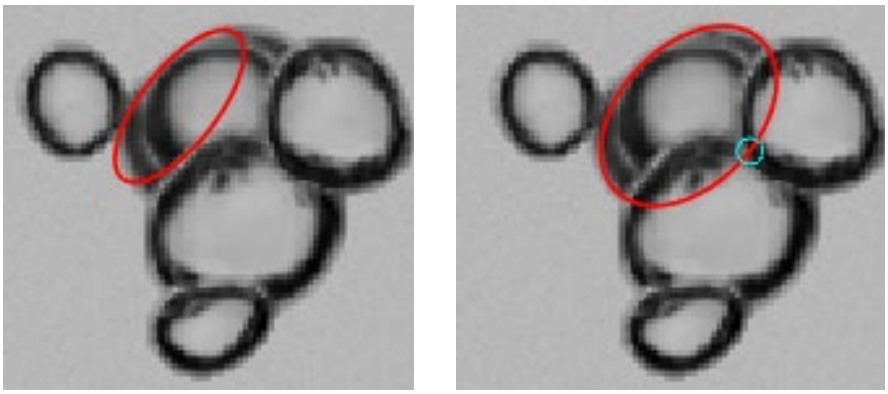

Figure 2.20-Effects of Point Placement; (Left) Taubin's method poorly approximates the shape of a mostly occluded bubble, (Right) The new point introduced by the Point Placement algorithm (shown in blue) improves the approximation produced by Taubin's method

2.2.4 Point placement algorithm

As mentioned, this algorithm is introduced to supplement the performance of Taubin's method in predicting the shape of the occluded contour of an ellipse. This algorithm is only executed if Taubin's method produces an RMSE estimate above a certain threshold ($RMSE_{taubin} > 1$), and its results are only utilized, by placing a new segment point within the occluded section, in cases where this point placement decreases the RMSE of the bubble segment.

This algorithm is based on a similar principle to the work of Stojmenovic and Nayak [112] in that it proposes ellipse parameters through the determination of the locations of the ellipse's foci. Another similarity between Stojmenovic and Nayak's method and the method detailed in the current work can be found in the criteria used for the selection of an ellipse's foci. The algorithm detailed in the current section differs from prior work in its determination of the ellipse orientation, and in its practical implementation of the foci search.

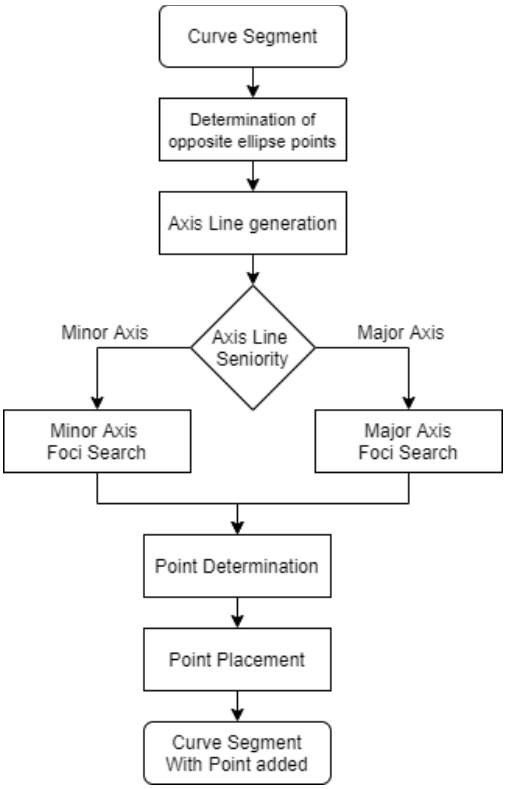

Figure 2.21-The Point Placement Algorithm

This new algorithm can be represented by *Figure 2.21*, with the various blocks being detailed as follows:

1. Determination of opposite ellipse points: Two points are found along the segment which represent reflections of each other about either the major or minor axis of the ellipse.
2. Axis line generation: The equation of the axis line about which the previously found points are reflected is determined. This axis line is used to determine the orientation of the ellipse.
3. Axis Line Seniority Decision: It is determined whether the axis line found is the major or minor axis of the ellipse.
4. Foci Search: Depending on which axis the line belongs to, a respective algorithm is executed in order to determine the foci locations, and thus the parameters of the ellipse to which the segment belongs.
5. Point Determination: The coordinates of the new point being added are found according to the ellipse parameters derived in the previous step.
6. Point placement: The new point is added to the segment.

The methods used to execute these algorithmic steps are detailed below:

1. Determination of opposite ellipse points

The first two steps of this algorithm work towards determining the equation of a straight line, which represents either a major or minor axis of the ellipse under investigation. The first step is then to find two points which lie on the ellipse segment, and which are reflections of each other about this axis line.

It is assumed that the point along the ellipse segment which intersects with the axis line is neither the first nor last point of the segment (when ordering the points in a counter-clockwise orientation). This is a fair assumption because of the convex nature of most segments protruding from a bubble cluster. From this assumption, either the first or last point along the segment (with the points ordered in a counter-clockwise orientation) is taken as one of the reflecting points. Whether the first or last point is taken is dependent upon which of those points has a reflection within the segment.

In most cases, the segment will be truncated in such a way that the sections of the segment divided by the axis line are not of equal length. As such, only the shorter section will have a reflection. This can be seen in *Figure 2.22*, where the segment representing a contour existing on a cluster boundary is shown in purple, with the occluded segment shown in blue. In this figure, the two sections of the purple contour are divided by the axis line into unequal lengths. Only for the shorter section does every point have a reflection point about the axis line (the major axis) which is found on the other section. To determine whether the first or last point of the contour will have a reflection point, the median location of the set of points making up the curve is found, and whichever of the first or last point is closer to the median is selected as a reflecting point. This median point is shown as a green circle in *Figure 2.22*. It was observed that the median of the set of points lies closer to whichever of the start or end points of the segment has a reflection within the set, and the selection process was designed accordingly.

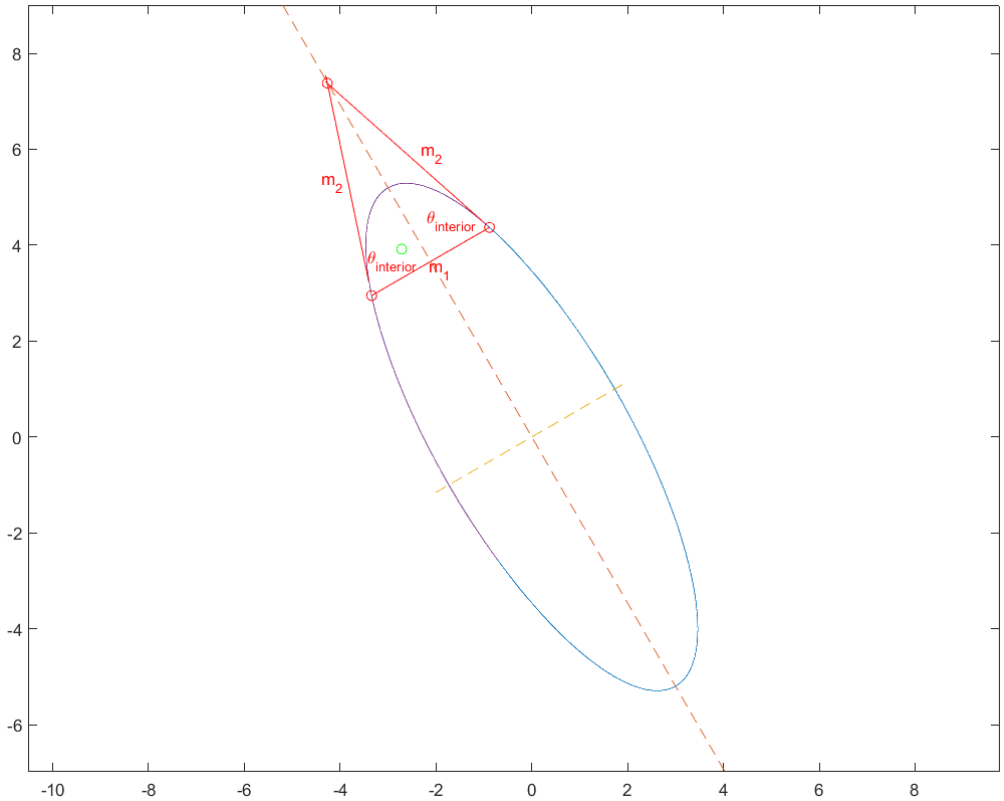

Figure 2.22-Determining reflection points; The shorter section (as separated by the dotted orange line) of the non-occluded ellipse segment (shown in purple) is found with the aid of the segment's median point (circled in green). Reflection points are then determined by finding an isosceles triangle (shown in red) with points lying on both segment sections.

With the first reflection point found, it is observed that two points along an ellipse are reflections of each other if the interior angles made between the respective tangent lines of each reflection point, and the chord connecting the two points, is equal. In other words, two points are reflections of each other if the points form an isosceles triangle, with the two reflection points as two of the triangles' vertices, and the intersecting point between the two reflection points tangent lines, projected into space, as the third vertex. The proceeding task is then to find a second reflection point which when paired with the initial reflection point satisfies the above conditions. This is achieved by sequentially calculating the interior angle between each candidate reflection point (with every other point besides the initial reflection point qualifying as a candidate point) and the chord between that point and the initial reflection point, as well as calculating the angle between the initial reflection point and the chord. The candidate reflection point with the interior angle most similar to the corresponding interior angle made with the initial reflection point is selected as the second reflection point. The isosceles triangle can be seen in red in *Figure 2.22*, with the two reflection points lying on the ellipse segmented circled in red, and the equal interior angles labelled as such.

The interior angle between the meeting of two straight lines can be calculated using the following equation:

$$\theta_{interior} = \tan^{-1}\left(\frac{m_1 - m_2}{1 + m_1 m_2}\right) \tag{2.18}$$

With m_1 and m_2 representing the respective gradients of the two lines. m_1 is assigned the value of the gradient of the chord between the initial reflection point and the candidate point, and m_2 is assigned the gradient value of the tangent of either the initial or candidate reflection point, depending on which interior angle is being calculated. The lines described by these gradients can be seen in *Figure 2.22*.

With the derivation of two points which are reflections of each other about the axis line under investigation, the next step is to determine the linear equation of this axis line.

2. Axis Line Determination

With the two reflection points having been determined, the axis line can be found easily by calculating the equation of a line which bisects the chord between the two reflection points at a right angle. This line is represented by the dotted orange line in *Figure 2.22*. This axis line is a crucial part of the following step of estimating the foci locations of the ellipse.

3. Axis Decision

The search methodology executed is dependent on whether the axis line found in previous steps corresponds to a major or minor axis. This decision is made based on the digital curvature value [117] found using *Equation 2.5*. This curvature is calculated for a parabolic arc taken between the reflection points found in Step 1. If this curvature value is above a certain threshold, the axis line is determined to correspond to the major axis of the ellipse, and if it is below that threshold, it is a minor axis. In this application, the threshold is set to 0.1 units. This low threshold ensures that the axis line is only determined to be a minor axis line if it passes through a curve segment of relatively low convexity. This decision criterion is practically effective in determining which axis of the ellipse the axis line corresponds to.

4-5. Foci Search and Point Determination

The methodology used to find the foci is dependent on the decision made in Step 3. Although different methods are used to search for the foci, the same foci selection criteria is applied regardless of the search method.

Foci Selection Criterion

This criterion is based on the ellipse property that the sum of the distances between each of the ellipse's foci and an arbitrary point along the ellipse is constant across all points along the ellipse. This can be defined mathematically as follows:

$$\sqrt{\left(x_i - x_{f_1}\right)^2 + \left(y_i - y_{f_1}\right)^2} + \sqrt{\left(x_i - x_{f_2}\right)^2 + \left(y_i - y_{f_2}\right)^2} = 2a = D_i \tag{2.19}$$

Where (x_i, y_i) is the location of some point along the ellipse and $\left(x_{f_1}, y_{f_1}\right)$ and $\left(x_{f_2}, y_{f_2}\right)$ are the respective locations of foci f_1 and f_2. $2a$ represents the constant summed distances between the foci and an arbitrary point, with a representing the length of the major axis. This property is used in selecting the optimal foci locations as follows: A set, labelled S, of the summed distances between both candidate foci and every point along the ellipse segment is created. This set is therefore made up of distance values corresponding to each point (x_i, y_i) along the ellipse segment, with a point's corresponding distance value assigned the form D_i. The candidate set of foci which creates the set with the lowest variance are selected as the ellipse foci. In other words, the candidate foci pair which produce the most constant value when their distances are summed across the entire set of points along the ellipse segment are selected as the final ellipse foci. The equation to minimized is therefore the variance in D values across the set of points, expressed as follows:

$$Var(S) = \frac{1}{N}\sum_{i=1}^{N} D_i^2 - \left(\frac{1}{N}\sum_{i=1}^{N} D_i\right)^2 \tag{2.20}$$

Where $S = \{D_i | i = 1,2,3 \ldots, N\}$. Stojmenovic and Nayak [112] show that it is not straightforward to minimize this function in terms of the foci locations, and resort to finding candidate foci through a linear search across a set of pixels. The following methods work in a similar way, yet the search algorithms applied here are novel.

Major Axis Foci Search and feature extraction

Because the Foci of an ellipse lie on the major axis, the pixel search occurs along the derived axis line, with each candidate foci pair existing along this line. The pixel search occurs along the major axis line (shown in green in *Figure 2.23*), between its intercept with the ellipse, and a reasonably far distance away from that intercept, within the interior of the ellipse. Every possible pairing of points along this line is considered a candidate foci pair, with a set S being generated for every candidate foci pair. A candidate foci pair is shown in red in the figure below. As described above, the candidate foci pair which produces a set S with the lowest variance amongst its data values is selected as the final foci pair.

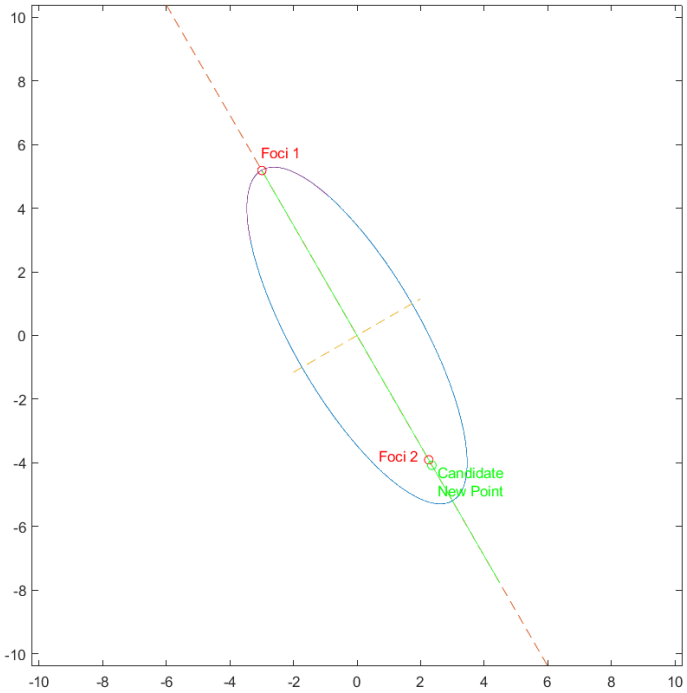

Figure 2.23- Major Axis Foci Search; A candidate foci pair (each circled in red) found along the major axis line (shown in green) is used to determine the location of the new point to be added (circled in green) to be added to the observable ellipse segment (shown in purple)

With the final ellipse foci having been determined, the remaining ellipse parameters can be calculated directly. A key ellipse parameter is the linear eccentricity, c, corresponding to the Euclidean distance between the two ellipse foci, and expressed as follows:

$$c = \frac{\|f_1 - f_2\|_2}{2} \tag{2.21}$$

This calculation can be performed once the final foci pair, made up of f_1 and f_2, has been found. With the final set s_{final} being made up of the summed distances between each point along the ellipse and the final foci pair, and since s_{final} was selected based on its low variance, the mean of s_{final} will be very similar to each point within the set and will correspond to the ellipse constant $2a$. Therefore, the semi-major axis length of the ellipse can be calculated as follows:

$$a = \frac{mean(s_{final})}{2} \tag{2.22}$$

From these two values, the eccentricity, e, and semi-minor axis lengths, b, can be calculated directly using the following ellipse equations:

$$e = \frac{c}{a} = \sqrt{1 - \left(\frac{b}{a}\right)^2} \tag{24}$$

Minor Axis Foci Search and feature extraction

Since the foci lie along the major axis, when the derived axis line is determined to be the minor axis of the ellipse, there is an additional level of complexity added to the search. Although the major axis can be defined once the length and angle of the minor axis are known, deriving the length of the minor axis for a sparse set of noisy points is non-trivial. To find the foci, a set of perpendicular lines bisected by a projection of a large minor axis is generated, with each line corresponding to the pixel which bisects it along the minor axis line. These perpendicular lines are generated along this large minor axis line between its intercept with the ellipse, and a reasonably far distance away from that intercept, within the interior of the ellipse. In *Figure 2.24*, the purple segment represents the ellipse segment under investigation, and the minor axis line from which perpendicular lines are generated is shown in green, with the perpendicular lines represented in red. Candidate foci pairs are generated along each perpendicular line, with each candidate focus in a pair being equidistant from the minor axis line. This aspect of the search ensures that the foci are an equal distance from the ellipse centre, as is true for all ellipse foci. A set S is generated for each candidate foci pair, and the candidate foci pair which produces such a set with the lowest variance amongst its data values is selected as the final foci pair, f_1 and f_2. Using these locations, the formulas defined above can be used to derive the ellipse features.

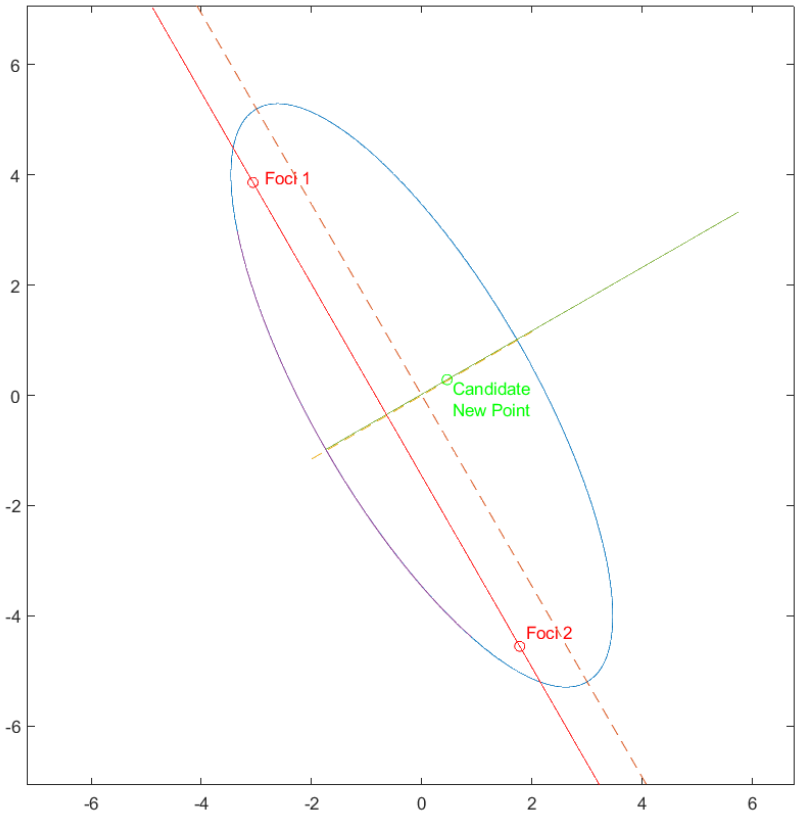

Figure 2.24- Minor Axis Foci Search; A candidate foci pair (each circled in red) found along a perpendicular line (shown in red), which is bisected by the minor axis (shown in green) is used to determine the location of the new point (circled in green) to be added to the observable ellipse segment (shown in purple)

6. Final Point Placement

Although the current algorithm can be used to completely define the shape of an ellipse, given the features extracted in previous steps, it was found that combining the estimations made by this algorithm with those produced by Taubin's method, by inserting a new point determined by the point placement algorithm into the original set of points describing the bubble segment, and then applying Taubin's method to fit an ellipse to the new set, produced much better results in terms of bubble shape approximation on real world images. By defining an ellipse entirely by the values derived by the current algorithm, some of the original points defining the bubble segment can be poorly approximated in the final ellipse estimate for real world bubble images, whereas Taubin's method ensures a better fit to the original points.

The pixel location of the new point added to the set is found opposite to the existing axis intercept, along the axis line found in *Step 2* of this algorithm, at a distance from it equal to either the major or minor axis lengths derived by this algorithm, with this distance corresponding to the axis line's definition. *Figure 2.23* and *Figure 2.24* show how new points are placed during the execution of the algorithm.

Performance evaluation of Point Placement algorithm

Testing

Before the point placement algorithm can be evaluated in terms of its contributions to the results of the larger bubble segmentation algorithm, it must be evaluated on its own on an ideal dataset. Such a dataset is produced by randomly truncating ellipse segments to form a dataset of parabolic curves derived from a set of larger ellipses. The algorithm is then used to estimate the parameters of the larger ellipse through analysis of the given curve segment. The evaluation of this algorithm will consider its merits as a method for ellipse parameter estimation, as well as its ability to place a new point successfully along the perimeter of the greater ellipse that the curve segment belongs to.

When creating an "ideal" dataset, it must be considered that this algorithm makes use of mathematical principles which are true along a continuous Euclidean plane. Because the implementation of the algorithm is digital, the ellipses are defined by a set of ordered points, rather than a continuous locus. Likewise for all line segments considered by the algorithm. As such, the "ideal" digital dataset is one with ellipse segments made up of points with a resolution high enough for the digital nature of the curve to have little to no effect on the results produced by the algorithm. The digital ellipse points were generated with an interval of 5E-3 along the axis of the Euclidean representation of the ellipse, producing a set of points which represent a high-resolution digital ellipse. "Units" is the metric used to describe spatial measurements along this Euclidean plane.

The randomly truncated ellipse segments have a length equal to a quarter of the total ellipse perimeter length. The dataset was generated using ellipses with eccentricities of 0.745, 0.866, 0.917, 0.943 and 0.968. These eccentricities were produced by maintaining a minor axis length of two units, and generating ellipses with major axis lengths of 3,4,5,6 and 8. Six segments were generated for each of these ellipses every $15°$ of rotation, between angles of $0°$ and $90°$, producing a dataset made up of 180 ellipse segments. Three examples of ellipse segments used for testing the point placement algorithm can be seen below, with the segment of the ellipse upon which the algorithm is applied shown in purple.

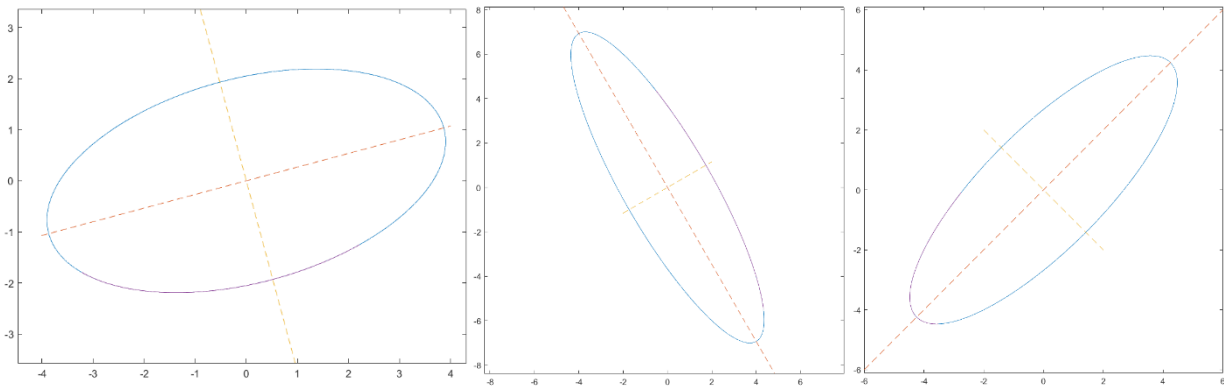

Figure 2.25-Examples of curve segments (shown in purple) taken from the ideal-ellipse dataset

Results

Table 2.1 presents the various errors in the extraction of ellipse parameters across the dataset. The RMSE in eccentricity can be thought of as a measure of the algorithms ability to predict the shape of an ellipse. The RMSE in rotation angle represents the algorithms' ability to estimate the angle of rotation of the ellipse. Area contained within an ellipse is given by:

$$A_{ellipse} = \pi ab \tag{2.25}$$

This value is used to calculate the error in the algorithm's estimation of the ellipse's size. Finally, the new point location error magnitude represents the Euclidean distance between the new point placed by the algorithm and the ideal location of this point.

Table 2.1- Point Placement Test Results

Major Axis length	True Eccentricity	RMSE Eccentricity	True Area $[units^2]$	RMPSE Area [%]	RMSE angle of rotation [rad]	New Point Location Error magnitude [units]	
						Mean	Std
3	745E-3	8.4E-3	18.8	3.5E-2	3.9E-3	71.6E-3	90.7E-3
4	866E-3	3.5E-3	25.1	2.4E-2	3.9E-3	80.7E-3	97.6E-3
5	917E-3	3.0E-3	31.4	3.0E-2	7.2E-3	116.3E-3	184.8E-3
6	943E-3	3.0E-3	37.7	3.4E-2	1.1E-2	173.7E-3	254.8E-3
8	968E-3	2.6E-3	50.3	5.0E-2	1.7E-2	277.3E-3	494.0E-3

Discussion

The RMSE in eccentricity remains small across all the ellipses, given that the eccentricity of an ellipse can be values between 0 and 1. It can then be claimed that the point placement algorithm performs very well in estimating the shape of an ellipse in ideal circumstances.

The range of possible values for the Absolute Angle of rotation error are between zero and $\pi/2$. This means the errors produced for this metric are once again extremely small, showing that the algorithm performs very well when predicting the angle of rotation for an ellipse. The fact that the RMSE in angle of rotation decreases with increasing eccentricity is to be expected, as with greater eccentricity comes a greater likelihood of the point placement algorithm correctly determining the major axis of the ellipse, due to the increased number of points found along the major axis, and from this, correctly calculate the ellipses angle of rotation.

The Root Mean Square Percentage Error (RMSPE) is taken for area measurements. This value is taken instead of RMSE because it is expected that error in area predictions will increase with size (with bubble sizes increasing with eccentricity in this dataset), so it is more appropriate to use a metric, like the RMSPE, which is scale invariant. This metric reveals that the percentage error in ellipse area is consistent and relatively small across bubble sizes and eccentricities, with this value being less than 0.05% of the true bubble size across all size brackets. This indicates that the point placement algorithm predicts the area ellipses from the ideal dataset with high accuracy, across size brackets.

A similar trend of increasing mean and standard deviation values with increasing eccentricity can be seen in the new point location error magnitudes. This is expected, as the point placement step makes use of the estimated major and minor axis lengths derived by the algorithm, as does the calculation of area, so the trend of errors seen in area estimations is like the trend in the point placement location error. As such, the causes of such errors would be the same as those mentioned in the previous paragraph. The point placement also relies on the algorithm's angle of rotation estimation, but because these estimations are very good, they do not have a great effect on the point placement error. The mean point placement error across eccentricities is small, given that these errors are multiple orders of magnitude smaller than the axis lengths, so the points placed are close to their desired locations. When comparing the ratios of the new point location error magnitudes with the lengths of the major axes belonging to their respective ellipses, this ratio again increases with eccentricity, showing that not only is the error larger, but this error with respect to the axis length also increases. This is in line with the cause of error mentioned above, with the error increasing with the number of candidate foci pairs. This ratio remains relatively small, (never more than 34.7E-3) and the location error itself never exceeds 0.3 units, with an error of zero falling within one standard deviation of the mean in all cases.

The success of this algorithm is only reduced by its digital implementation, with the core ideas, based in Euclidean geometry, being sound. Further analysis and testing of this point placement algorithm and its contribution to the overall performance of the bubble segmentation algorithm can be found in the next section.

2.3 Evaluation of Segmentation and Parameter Estimation Algorithms

2.3.1 Testing

The breakpoint algorithm is evaluated against two distinct datasets: A set of 1000 images of bubbly flow generated using BubGAN [79], and a set of 100 real bubbly flow images from the experimental environment described in *Part I, Section 4*. BubGAN images represent a close to ideal case, where bubble edges are clearly defined, the images are noise free, and bubbles have borders that are well approximated by an ellipse. Additionally, when using the BubGAN dataset, the ground truth parameters describing the bubbles, and used for comparison against the segmentation algorithms estimations, are known, as they are computer generated, resulting in a more accurate set of ellipse parameters. Conversely, the real

bubble dataset is made up of a set of ground truth bubble parameters, derived from ellipses that were manually fitted to each bubble in the image dataset, using BubCNN's transfer learning application [76]. This results in a dataset which is likely to have biases due to human error. Another issue with this method of extracting ground truth bubble parameters from real world data, and the segmentation algorithms described here in general, is that non-ellipsoidal bubbles have their shapes estimated using ellipsoids, resulting in further errors in the extraction of bubble shapes. With these issues acknowledged, given the time required for a user to manually extract the shape of each bubble, simplifying this process, by estimating each bubble's shape as an ellipsoid, is an acceptable solution. These "ground truth" datasets of bubble parameters are generated to use as benchmarks for comparison across all segmentation algorithms, on both datasets.

For each dataset, the breakpoint algorithm will be compared to BubCNN [76] across all performance metrics. This comparison is useful because BubCNN represents a high performing benchmark in bubble feature extraction, on real world and BubGAN images. This comparison is also interesting as a measure of performance between classical computer vision techniques, based in image processing algorithms, and more modern techniques which make use of deep learning and CNNs. The breakpoint algorithm will also be tested with and without the point placement algorithm, to confirm if its successful performance on an ideal ellipse dataset translates to more complex datasets.

2.3.2 Results and Discussion

The same performance metrics for measuring an algorithm's ability to extract bubble features (related to bubble size and shape) as those used for the point placement algorithm are applied here, except for the "New Point Location Error" (for obvious reasons) and the "Absolute angle of rotation error". It was found that Taubin's method of ellipse parameter estimation often predicted highly accurate ellipse parameters (axis lengths and ellipse centroid location) with a highly inaccurate angle of rotation estimation. This is most likely due to the circularity of many of the ellipses, resulting in ill-defined angles of rotation. These circular bubbles could also result in Taubin's method misclassifying the semi axis labels assigned to a measurement, while still having a low error in terms of the axis length estimates. This misclassifying of semi axis labels for a measurement would result in large, but unimportant, angle of rotation errors in cases of near spherical bubbles. As such, this metric is not included, as it has little relation to the true performance of the break-point algorithm in its ability to predict the shape and size of bubbles within an image.

New performance metrics are also introduced to measure the algorithm's performance in terms of bubble detection and segmentation. Detection accuracy represents the number of correct bubble detections made by an algorithm within the entire dataset, divided by the total number of bubbles in the dataset, as a percent. False Positive Rate (FPR) shows the percent of all bubble detections which represent an incorrect detection. Group Detection Rate (GDR) represents the percent of true positive detections that include the centre location of more than one bubble within the perimeter of the predicted ellipse. This is a measure of an algorithm's ability to segment individual bubbles from a cluster of bubbles. The results of the break-point algorithm and BubCNN on the two datasets can be seen below in *Table 2.2* and *Table 2.3*, with *Figure 2.26* and *Figure 2.32* revealing information related to the algorithms performance across bubble sizes, and *Figure 2.28* and *Figure 2.31* showing the Probability Density Functions of each algorithm in terms of their bubble size predictions for each respective datasets.

Figure 2.29 and *Figure 2.30* allow for the qualitative comparison between methods for each dataset, by presenting a graphical representation for the ellipses detected and estimated by each algorithm in comparison to the ideal case. For each of these images, the red ellipses represent the ground-truth ellipse for each bubble, with the green ellipses representing the BubCNN prediction, and the blue ellipses representing the breakpoint algorithm (with point placement enabled) prediction. The red ellipses with dashed lines represent those bubbles which were not detected by either algorithm, with the dashed lines in the other two colours representing false positives detections by each colour's respective algorithm. It is noticeable that some very small bubbles are not included in the ground truth dataset. This is because neither algorithm accurately detected these small bubbles, and the manual labelling of each of them would be extremely time consuming, with their contribution to the overall void fraction being minor. As such, some very small bubbles were not included in the ground truth dataset.

BubGAN Dataset

Table 2.2-Segmentation Test Results (BubGAN)

Algorithm	RMSE eccentricity	RMSE area $[pixels^2]$	FPR [%]	GDR [%]	Detection Accuracy [%]
BubCNN	7.1E-2	127.8	0.5	13.0	70.1
Breakpoint (without point-placement)	1.0E-1	135.4	5.2	17.5	87.5
Breakpoint (with point-placement)	8.4E-2	132.0	5.2	17.0	87.5

On average, the breakpoint algorithm performs better in terms of bubble shape and size estimation when it makes use of the point placement algorithm, as is evident by the improved RMSE in eccentricity and area seen in *Table 2.2*. The cause for these improved results can be found in *Figure 2.26*. Because the point placement algorithm is not triggered for most bubbles, the distributions of absolute errors for each size bracket (for both eccentricity and area errors) are very similar for the breakpoint algorithm, with and without point placement. The reduction in the area and eccentricity RMSE values is then attributed to the point placement algorithm reducing the amount of and magnitude of outlying absolute errors. This is also the cause for the small differences in the error distributions across size brackets. *Figure 2.27* shows an example of how these outlying errors in eccentricity and size are corrected for by the point placement algorithm: Initially, Taubin's method estimates the shape of the bubble with large errors in eccentricity and size due to the incorrect estimation of the occluded bubble shape within the bubble cluster, but with the aid of the point placement method, the shape of the bubble within the cluster is better approximated, hence removing the large outlying error. This effect is especially prevalent in size brackets up to 2040 $pixels^2$, with these sizes containing most of the bubbles in the dataset, as can be seen in *Figure 2.28*.

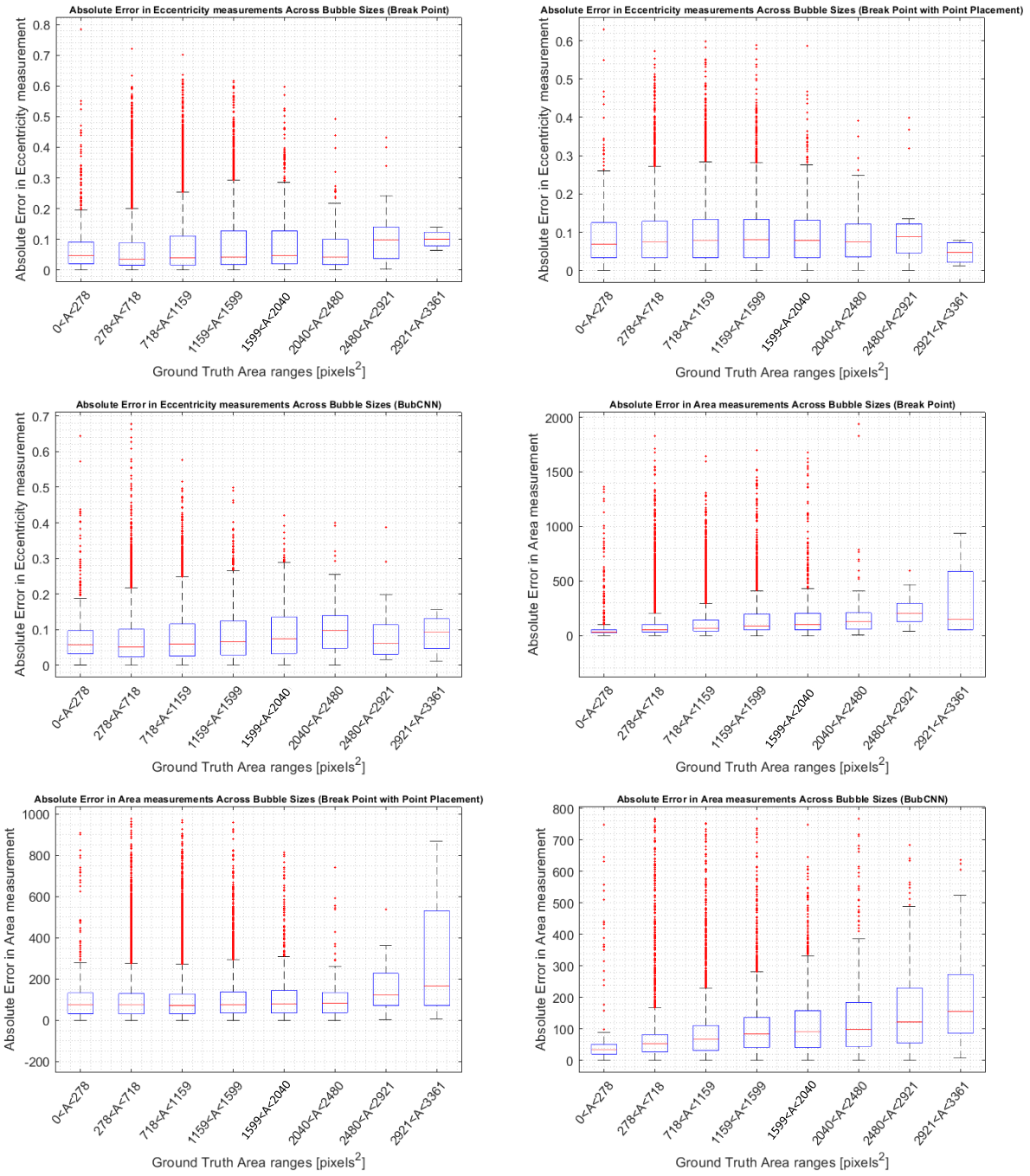

Figure 2.26-Algorithm performance across bubble size (BubGAN)

For larger bubbles, many of the outliers are consistent whether the point placement algorithm is enabled or not. This is most likely a case of the point placement algorithm failing to be triggered for these large bubbles, because if the algorithm were triggered, there would at least be some change to the errors of these outliers. This activation error could be the result of larger bubbles having a greater number of points describing them, and hence, the $RMSE_{taubin}$ which triggers the activation of the breakpoint algorithm could be generally lower for these bubbles, as Taubin's method can approximate an ellipse to these larger sets of points with higher confidence.

In terms of bubble size distribution (BSD), *Figure 2.28* reveals that the breakpoint algorithm tends to overpredict bubble sizes, as is shown by this algorithms PDF being shifted to the right, with the point placement algorithm slightly improving this overprediction. This improvement can be attributed to the reduced error in bubble size estimation provided by the point placement algorithm. The general cause for this overprediction in size by the breakpoint algorithm is the distortion to bubble outlines from the Gaussian filter applied in the pre-processing phase. While the application of this filter is required for the segmentation algorithm, it causes ill-defined bubble edges due to the blurring effect, resulting in the overprediction of bubble size.

It is encouraging that the point placement algorithm reduces the GDR of the breakpoint algorithm, as this means that in some cases, the point placement algorithm improves the bubble shape estimate, so that rather than including multiple bubbles within the same estimate, the occluded section of an exposed bubble segment is correctly included within the shape estimation, while the otherwise incorrectly included bubble segments are cut out. *Figure 2.27* shows an example of this effect. Considering this improved result, and the point placement algorithms corrections to large eccentricity and area errors made by the breakpoint algorithm, the point placement algorithm is to be included in the overall breakpoint algorithm for future work. In the proceeding paragraphs, when referring to the breakpoint algorithm, the point placement algorithm is included.

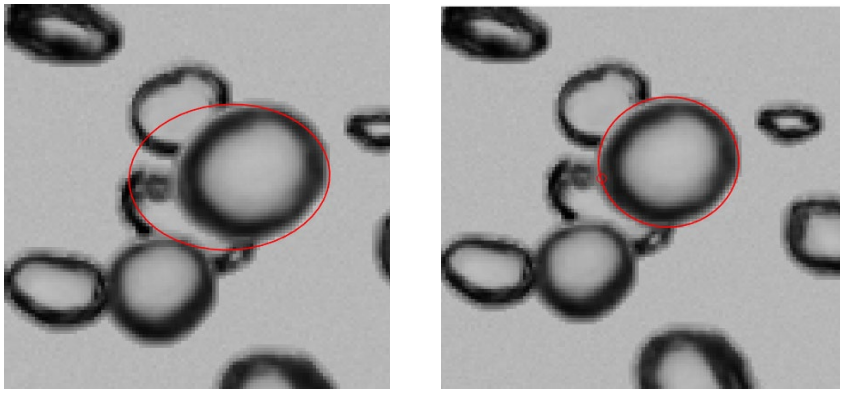

Figure 2.27- The Point Placement algorithm's effect on group detections; (Left) Original image, where multiple bubbles are grouped within the same ellipse, (Right) Point Placement algorithm (new point shown in red) improves the bubble shape estimation within the bubble cluster, correcting for the group detection.

On average, BubCNN outperforms the breakpoint algorithm in all metrics used to measure performance in both bubble shape and size estimations, as is seen in *Table 2.2*. *Figure 2.26* shows that BubCNN predicts area and eccentricity of a bubble with very similar median values to the breakpoint algorithm, but BubCNN has far fewer, and lower magnitude, outlying error values within the bubble size ranges which contain most bubbles (areas between 278 and 1588). This can be attributed to the algorithmic approach used by the breakpoint method causing errors to be propagated from one step of that algorithm to the next, resulting in massive errors in bubble shape and size estimation at its output. This error propagation does not occur for BubCNN, as this method considers multiple high-level spatial features when making a prediction of bubble shape and size, rather than the algorithmic use of low-level image features. This use of high-level features means that BubCNN will only detect a bubble if certain high-level features are present, leading to fewer detections (as seen in BubCNNs detection accuracy), but also fewer high outlying errors in the estimation of a bubbles shape and size.

As with the breakpoint algorithm, the rightward shift of BubCNNs PDF in *Figure 2.28* reveals that this algorithm tends to overpredict bubble size. This is an issue in BubCNN's shape regression network, used to estimate the shapes of bubbles. This result might be improved by increasing the size of the training set for that network.

With the mean ground truth bubble size being 872 $pixels^2$, the RMSE values for both algorithms' area predictions are around 15% of the mean bubble size. This error is due to both algorithms tendencies to overpredict the size of a bubble, as is evident by the BSD PDFs of the two algorithms appearing as translations of the ground truth PDF. As such, these errors can be corrected for by adjusting semi-axis length predictions made by the two algorithms to counteract these errors. Although the breakpoint algorithm performs worse than BubCNN in bubble shape and size estimation, this difference in performance is very small, as is evident in the error values shown in *Table 2.2*.

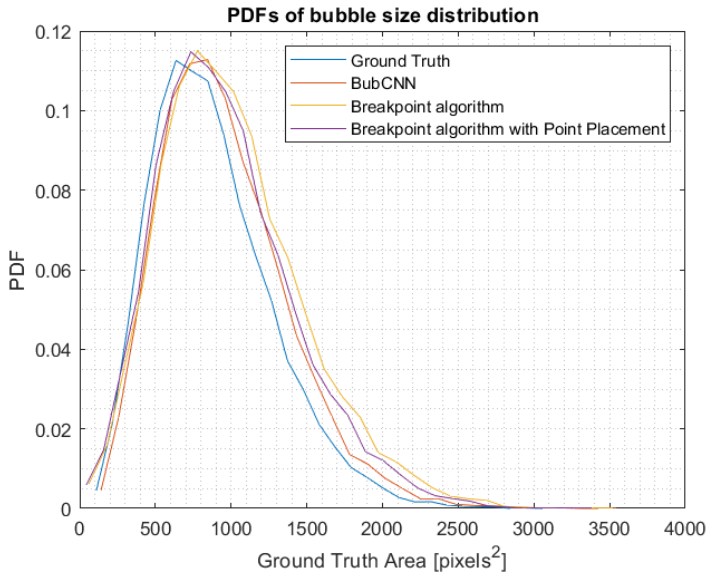

Figure 2.28-PDF of bubble size distribution (BubGAN)

For both methods, large errors in bubble size estimations are seen for bigger bubbles, while the eccentricity errors for these bubbles remain relatively low. These surprising results are due to the very limited number of datapoints within these size ranges. There are less than 10 bubbles for each of the largest two size brackets, resulting in biased data. The cause for the high area error in some of these cases is that the large bubble's area is mostly cropped out of the image.

Eccentricity error remains relatively low for these larger bubbles because eccentricity relies on a ratio of semi axis lengths, so this error does not scale with the larger bubbles, resulting in a relatively small errors even if the size estimate has a large error. On average, both methods predict eccentricity with high accuracy, as revealed by the eccentricity RMSE values shown in *Table 2.2*, given that eccentricity values are between zero and one.

BubCNN drastically outperforms the breakpoint algorithm in terms of its false positive rate. This highlights the sensitivity to noise and variations in low level image characteristics found in traditional segmentation techniques, with the algorithmic approach incorrectly detecting artifacts within an image as bubble segments, leading to incorrect ellipse estimations. Examples of these errors can be observed in *Figure*

2.29. The use of higher-level image features to detect bubbles by BubCNN results in far fewer false positives, as this method is not as sensitive to minor changes in low level image characteristics. *Figure 2.29* also reveals that the breakpoint algorithm fails around the edges of images, leading to a disproportionate number of false positives in these regions. This is because of the bubble sections that are cropped out of the image, resulting in the inner bubble not being filled in during the pre-processing phase of the algorithm. This is because the algorithm cannot distinguish between the inner and outer bubble without closed bubble edges. This results in the algorithm being applied to a "U" shape describing edge bubbles, rather than an elliptical one, leading to the algorithms failure.

In terms of GDR, BubCNN also performs better than the breakpoint algorithm. This shows that the use of high-level image features to detect bubbles results in better bubble segmentation and bubble shape estimation within a bubble cluster. Group detections by the breakpoint algorithm can be seen in *Figure 2.29*, where the algorithm fails to detect certain breakpoints, leading to the incorrect grouping of bubble segments. These missed breakpoints could be a result of the image being blurred too much in the pre-processing phase of the algorithm. Another reason for BubCNNs excellent segmentation performance is that this algorithm can identify features within the outline of a bubble cluster, whereas the breakpoint algorithm uses only this perimeter for segmentation. This results in BubCNN being better at estimating the shape of bubbles which have curve sections within a bubble cluster. Unlike the breakpoint algorithm, BubCNN can estimate the shape of a bubble which has its perimeter entirely within a larger bubble cluster, as is seen in *Figure 2.29 (Centre)*. With bubble segmentation and detection being among the main functions of the breakpoint algorithm, it is recommended that BubCNN be used for these tasks instead of the breakpoint algorithm due to its superior performance in these processes.

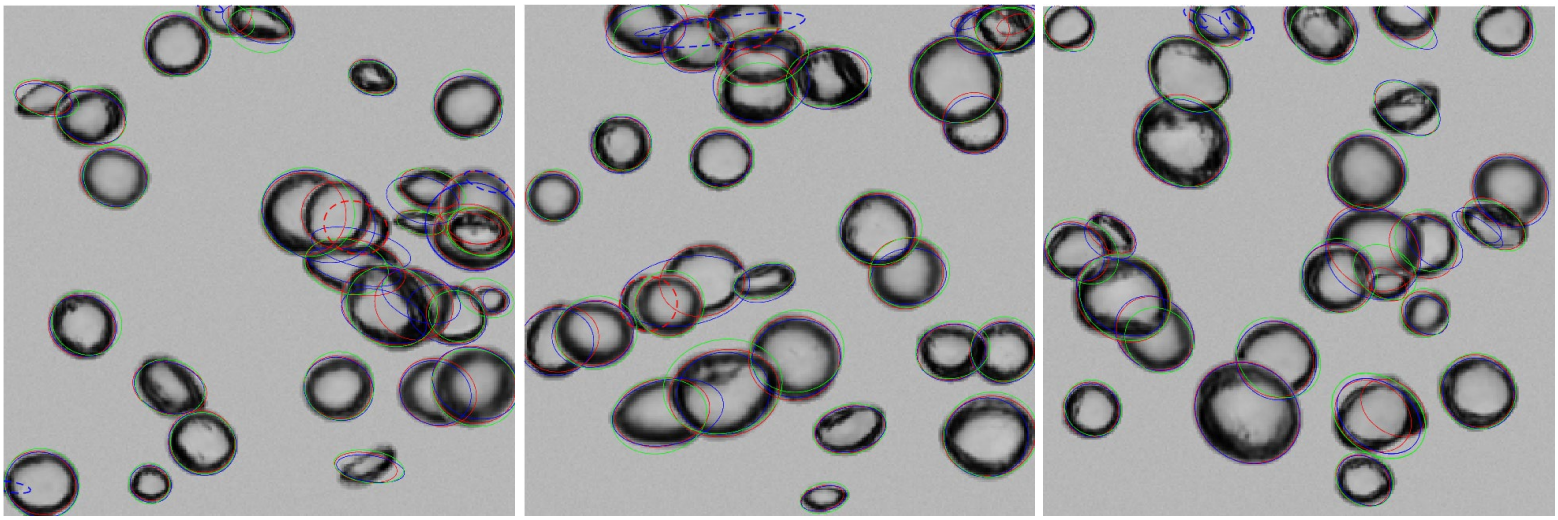

Figure 2.29-Qualitative Segmentation Results (BubGAN); The bubble segmentations achieved by the breakpoint algorithm with point placement (Blue) and BubCNN (Green) can be compared to the ground truth ellipses (Red). For the algorithms, the ellipses with dashed lines represent false positives, and for the ground truth data, they represent ellipses missed by both algorithms.

The breakpoint algorithm outperforms BubCNN in detection accuracy. This is because the breakpoint algorithm's use of lower-level image features to detect bubbles results in a lower threshold in terms of the requirements need for the algorithm to detect a bubble, whether that detection is correct or incorrect. This is also the cause of the breakpoint algorithms high false positive rate. BubCNN requires more certainty of the presence of certain high-level bubble features to trigger a detection, and as such, misses

some detections which do not meet these requirements. It is worth noting that some bubbles generated by BubGAN are completely occluded by other bubbles, making it impossible for either algorithm (or a human observer) to detect them. This can be observed in *Figure 2.29 (Right)* and will result in a small decrease in detection accuracy for both algorithms, even though the detection of these bubbles is impossible.

From these results, BubCNN performs better, in tasks of bubble segmentation and shape estimation, than the breakpoint algorithm, on the ideal images produced by BubGAN. BubCNN was originally trained using BubGAN data, so it is unsurprising that it performs exceptionally well on such data. The higher detection accuracy of the breakpoint algorithm comes at the cost of far more false positives, and a lower GDR score than BubCNN, for reasons stated above. Although BubCNN outperforms the breakpoint algorithm for the current task, the results between the two methods are very close for this ideal dataset, as is confirmed in *Table 2.2*. This is a positive result for the breakpoint algorithm, as BubCNN represents a high benchmark in terms of bubble detection and shape estimation [75]. The similarity in performance between the two methods encourages the further development of the breakpoint algorithm, as a major advantage this algorithm has over BubCNN is that the errors made by this algorithm can be tracked, explained, and debugged, while BubCNN operates as a black box system, with it being difficult to analyse and fix the causes of its errors, due to the nature of deep neural networks.

Real World Dataset

Table 2.3-Segmentation Test Results (Real World)

Algorithm	RMSE eccentricity	RMSE area $[pixels^2]$	FPR [%]	GDR [%]	Detection Accuracy [%]	Complexity
BubCNN	1.3E-1	61.6	7.8	23.9	36.3	O(n)
Breakpoint (without point-placement)	2.4E-1	82.2	14.2	31.0	62.3	O(n)
Breakpoint (with point-placement)	1.8E-1	78.0	14.2	30.9	62.3	$O(b^2)$

When comparing *Table 2.2* and *Table 2.3*, it is observed that there is a decrease in performance across all metrics, for each algorithm, when tested on real-world data. With the mean bubble size for this dataset being 209 $pixels^2$, the RMSE area values seen in *Table 2.3* are lower in magnitude, but represent a larger percent of the mean value (35% on average across methods). It is notable that the performance of BubCNN decreases to a lesser degree than the breakpoint algorithm in all metrics besides detection accuracy. This is due to BubCNNs reliance on high-level image features in its processes, resulting in a reduced sensitivity to noise, and therefore a smaller dip in performance when the noise present in the real-world images is introduced to the algorithm. The breakpoint algorithm is particularly sensitive to noise because concavities in bubble outlines, which can arise due to noise around the edges of bubbles, are incorrectly detected as breakpoints. This leads to the incorrect grouping of bubble segments, or the splitting of bubble segments which should be represented as a single contour. These effects are detrimental to the breakpoints algorithm's ability to group bubble segments together, and accurately estimate the shape and size of an individual bubble, as is confirmed by comparing the two tables. The

introduction of image noise is also likely to be the cause for the increased FPR rate across algorithms, as noisy image artifacts could lead to the incorrect detection of bubbles by either algorithm. It is interesting that BubCNN experiences a large increase in its FPR, despite its increased robustness to noise. This increase could be a result of BubCNN incorrectly identifying bubble image features in noisy regions. The breakdown of the breakpoint algorithm due to image noise can be seen for some bubbles in *Figure 2.30*, particularly large ones. Noise around these large bubbles edges leads to an excessive number of segments, leading to grouping errors being made by the breakpoint algorithm. These images also depict many false positives for both methods, due to the reasons stated above.

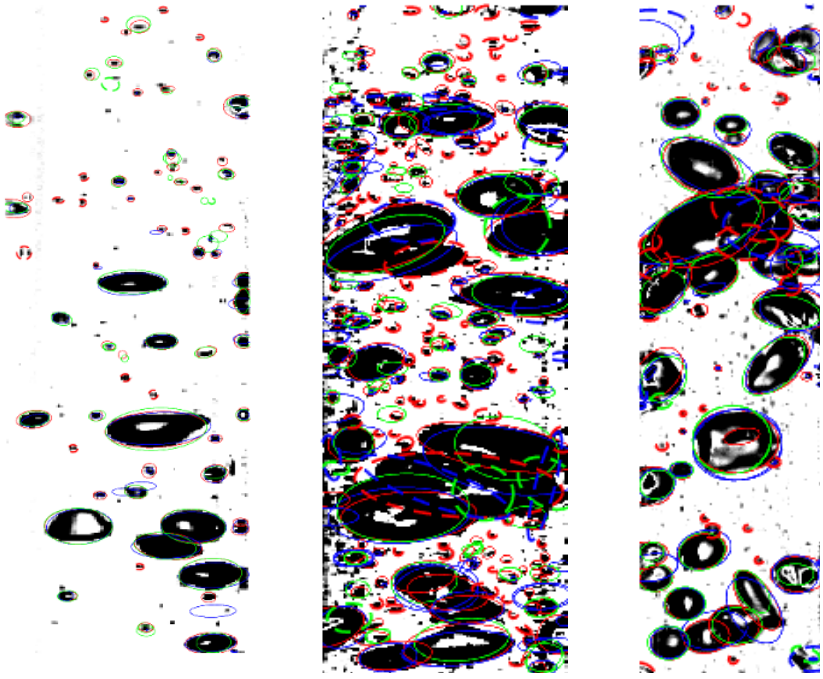

Figure 2.30- Qualitative Segmentation Results (Real World); The bubble segmentations achieved by the breakpoint algorithm with point placement (Blue) and BubCNN (Green) can be compared to the ground truth ellipses (Red). For the algorithms, the ellipses with dashed lines represent false positives, and for the ground truth data, they represent ellipses missed by both algorithms.

Another cause for the performance dip when testing the algorithms on real-world images is that real world bubbles can appear in non-elliptical shapes. This issue affects the breakpoint algorithm, as concavities which do not represent the meeting point between two bubbles, and non-smooth bubble edges, can lead to false breakpoint detections. For BubCNN, some of these non-elliptical bubbles often fail to be detected. This is because BubCNN was initially trained using BubGAN images, and then trained by using transfer learning with ground truth bubble data from a real-world dataset. Bubbles produced by BubGAN generally tend to be circular or elliptical in shape [78]. Because far more BubGAN data was used to train BubCNN than real-world data, the network has a bias to detect mostly elliptical bubbles, with non-elliptical shapes often being missed in the detection phase of the algorithm. This issue could be amended by training BubCNN on a more diverse dataset, containing more real-world data. The effect of non-elliptical bubbles is especially noticeable in *Figure 2.30*, where it is evident that most bubbles cannot be described well by an ellipse, leading to white space within a bubble's ellipse, or bubble edges going outside an ellipse's bounds.

In terms of the increased GDR of both algorithms on the new dataset, different explanations are required for each. For BubCNN, the GDR increases because of the black, featureless interior of bubble clusters found in the real-world dataset, as is seen in *Figure 2.30*, as opposed to the well illuminated and feature dense interior seen in the BubGAN dataset. These dark regions lead BubCNN to identify whole clusters as a single bubble, or falsely detect bubbles within the region (as seen in *Figure 2.30 (Centre)*) as it is these interior features which allow the algorithm to accurately segment bubbles within a cluster. The increase in GDR for the breakpoint algorithm is again due to the increased noise level in the images, resulting in the incorrect segmentation and separation of bubbles within clusters.

The cause for the drop in detection accuracy when testing the algorithms on real world data is the presence of tiny bubbles in the images, far smaller than the bubbles seen in the BubGAN dataset. These bubbles appear as noisy dots, or tiny black spheres within the image. This increased number of small bubbles found in the images is evident when analysing *Figure 2.31*. The PDFs are shifted much farther left than those seen in *Figure 2.28*, indicating that the average bubble size (being 209 $pixels^2$ for the ground truth dataset) is smaller in the real-world dataset than in the BubGAN dataset (872 $pixels^2$). These small bubbles are often missed by the breakpoint algorithm, as they are removed from the image when the Gaussian filter is applied in the pre-processing phase. BubCNN also performs particularly badly in detecting tiny bubbles, as these noisy, featureless bubbles are completely different to the bubbles used to train the network from BubGAN, leading to the severe drop in detection accuracy achieved by this algorithm on the new dataset. The failure of these methods to detect smaller bubbles has led to their PDFs of BSD, seen in *Figure 2.31*, to be shifted to the right of the ground truth function, with the much smaller bubbles (which occur with the highest frequency in the ground truth dataset) having a probability of zero in terms of being detected by these methods. Some of these errors can once again be corrected for by adjusting semi-axis length predictions made by the two algorithms to counteract these systematic errors. *Figure 2.30* confirm the failing of these two algorithms to detect tiny bubbles, with an excessive number of small bubbles being missed by both algorithms in each image. Although in *Figure 2.32* it appears that the methods under investigation perform well on these tiny bubbles, this is because these results include only detected bubbles. This means that the methods under investigation perform well in terms of bubble shape and size estimation for the small bubbles that they do detect, but still, they fail to detect many of them.

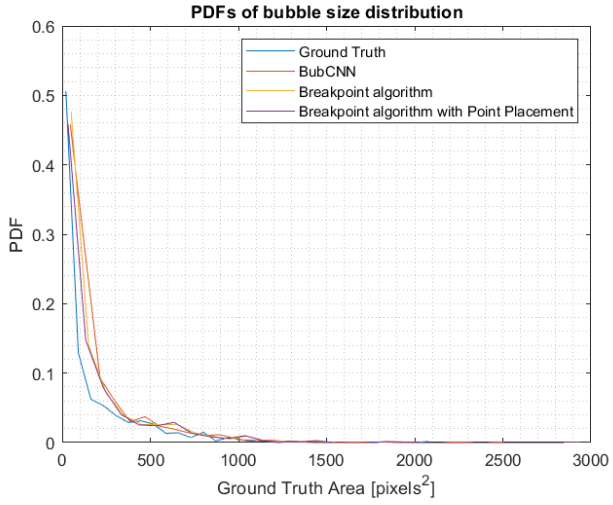

Figure 2.31-PDF of bubble size distribution (Real World)

The inclusion of the point placement algorithm has a positive effect on the performance of the breakpoint algorithm when applied to this dataset. The point placement algorithm has a positive effect on bubble size estimation, with a 5% increase in performance in terms of area RMSE values. This increased performance is due to the same kinds of corrections made to breakpoint algorithm predictions, which have outlying absolute errors, as those mentioned for the BubGAN dataset. While this is beneficial to the breakpoint algorithm, the point placement algorithm has a generally negligible effect on the majority of bubble shape estimations, as is seen in *Figure 2.32*, with only a few outlying bubble shape errors being corrected for. A similar result is seen in this figure, in terms of the corrections made to outlying eccentricity errors by the point placement algorithm, resulting in the lower eccentricity RMSE when point placement is enabled. In terms of GDR, the point placement algorithm has a small but positive effect, pointing to an improvement in the shape estimation of occluded bubble segments. With these small improvements considered, it is recommended that the point placement algorithm be included in the overall breakpoint algorithm when applied to real-world images.

When considering the performance difference between BubCNN and the breakpoint algorithm, once again, BubCNN outperforms the breakpoint algorithm in every metric besides that of detection accuracy. As mentioned, and accounted for, BubCNNs performance is better in tasks of bubble shape and size estimation across bubble sizes. This can be confirmed by *Figure 2.32*. These figures show that the breakpoint algorithm performs similarly to BubCNN on most bubbles, as can be confirmed by comparing the results seen in the first area range (within which most bubbles fall) for both eccentricity and area errors. It is for larger bubbles, which are far less numerous, as is shown in *Figure 2.31*, that BubCNN significantly outperforms the breakpoint algorithm in bubble shape and size estimation (except for the poor results achieved by BubCNN on the two largest size brackets, where there is only one bubble in each bracket, leading to biased results). For void fraction estimation, this is significant, as these larger bubbles contribute more to the volumetric gas content observed in an image. BubCNN also outperforms the breakpoint algorithm in terms of GDR and FPR. This is due to the breakpoint algorithms sensitivity to noise, which causes a high number of false positives and false breakpoint detections, as discussed above.

The better performance achieved by BubCNN over the breakpoint algorithm is also qualitatively evident in *Figure 2.30*, where the high number of false positives and poor bubble shape and size estimation achieved by the breakpoint algorithm can be seen. The performance of the two algorithms is most similar in *Figure 2.30 (Left)*, where there is only a small number of bubbles, with reduced image noise. In this image, and in cases in the other two figures where the bubble outlines are simple and separate from tiny bubbles, the breakpoint algorithm performs well in tasks of bubble segmentation, with performance appearing like what is achieved on the BubGAN dataset. However, for many cases in the other two images, the increased number of tiny bubbles and image noise results in the failing of the breakpoint algorithm to correctly group bubble segments, leading to bubble shape estimations with large errors. By comparing *Figure 2.30 (Left and Centre)* to *Figure 2.30 (Right),* it is evident that BubCNN also suffers from a performance drop with the introduction of many tiny bubbles to an image, although this performance drop is still not as drastic as the breakpoint algorithms.

An essential factor to consider when considering the deployment of these algorithms is their complexity. BubCNN and the Breakpoint algorithm (without point placement) each have complexity O(n), with n representing the number of bubbles in the image. This complexity is due to the fact that the same algorithmic steps are followed for each bubble. On the other hand, when the point placement algorithm is included, the complexity increases to $O(b^2)$, with b representing the number of pixels representing

bubble contours. This exponential complexity factor, and the fact that b is far greater than n, means that the point placement algorithm has a far longer run time than the other two, and as such is problematic for real-time deployment.

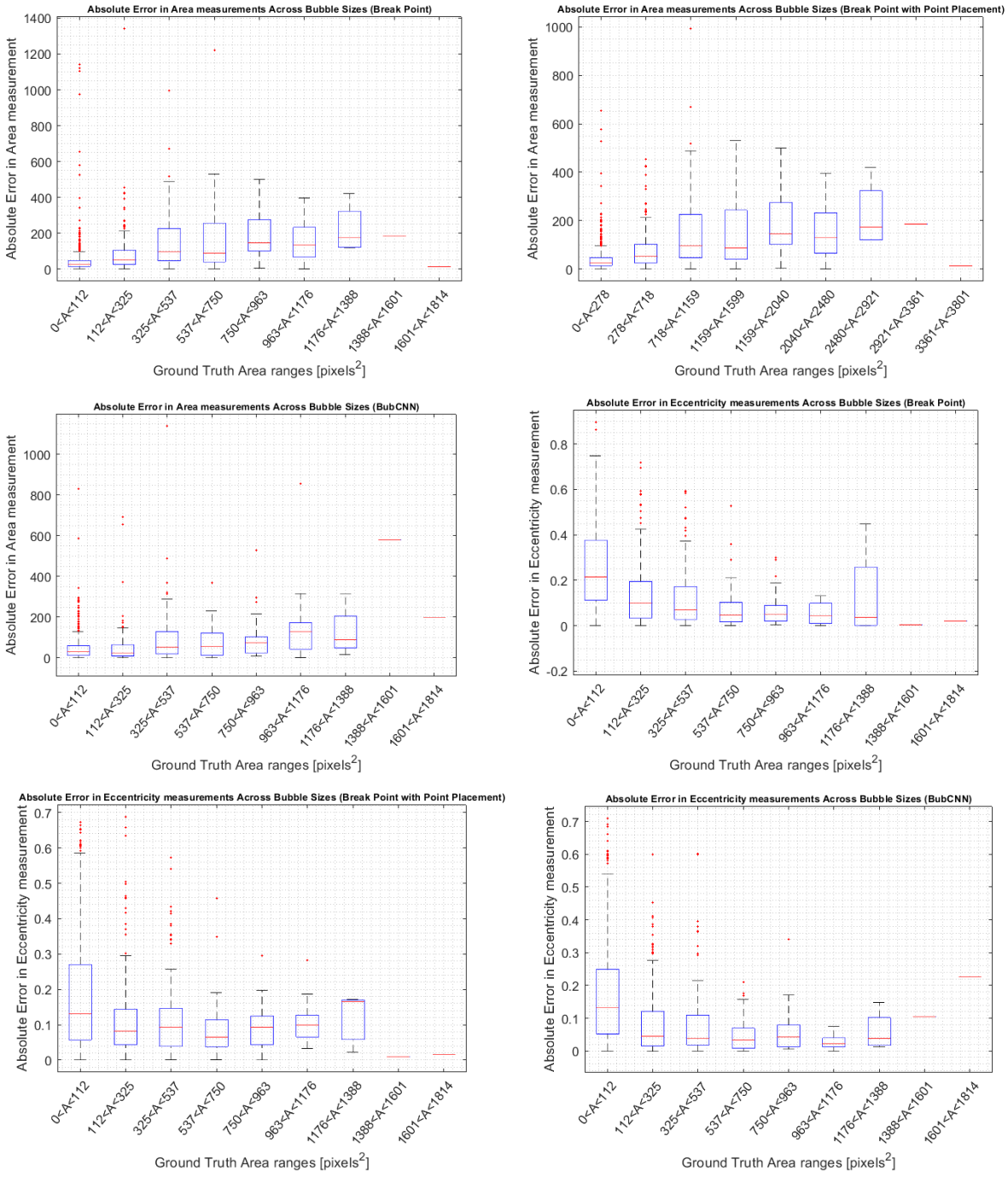

Figure 2.32- Algorithm performance across bubble size (Real World)

2.3.3 Conclusion

On both the real world and BubGAN datasets, the inclusion of the point placement algorithm in the breakpoint algorithm increased the performance of the latter. It is therefore recommended that the point placement algorithm should be included in the overall breakpoint algorithm. It is notable though, that the point placement algorithm drastically increases the interpolation time of the breakpoint algorithm (as is evident by the added complexity from the algorithm), so if runtime is important to the user when executing the breakpoint algorithm, it is better to exclude the point placement algorithm.

The breakpoint algorithm's similar performance, to that of BubCNN, on the BubGAN dataset, is encouraging for this method's further development. If the breakpoint algorithm and BubCNN had similar performance on a given dataset, it would be preferable to use the breakpoint algorithm, as the customizability of this algorithm allows for debugging, whereas BubCNN's use of deep neural networks means that its errors are difficult to interpret and debug.

While the performance of the breakpoint algorithm is similar to that of BubCNN on the ideal BubGAN dataset, the breakpoint algorithms performance is far below that of BubCNN on the real-world dataset. This shows that the use of lower-level image features, used in methods like the breakpoint algorithm, leaves an algorithm more sensitive to image noise and variations in image characteristics, leading to reduced performance in bubble shape and size estimation, and an excessive amount of false positive detections. On the other hand, the use of higher-level image features like those used by BubCNN produces an algorithm more robust to the above, and one which achieves better results. It is possible that the performance of the breakpoint algorithm would be closer to that of BubCNN in a more optimal test environment (as is the case with the BubGAN dataset), but BubCNN's greater robustness to sub-optimal environmental conditions makes it a better choice for analysing real world data.

The biggest failing of BubCNN on the real-world dataset is its poor results in the detection of tiny bubbles. Although both algorithms perform worse on smaller bubbles, these bubbles also contribute less to the overall volume of gas present in the fluid, and as such, have a smaller impact on the void fraction estimates. It is therefore recommended that in implementing either algorithm to extract bubble parameters for the purpose of calculating void fraction, that the smaller bubbles should be removed from the image through pre-processing methods to reduce the noise present in an image, and to sharpen the edges of the larger bubbles, so that BubCNN and the breakpoint algorithm can better extract the features of larger bubbles.

For this task, BubCNN is a better choice, as this algorithm provides more accurate shape and size estimation for medium and large bubbles, which contribute the most to the calculated void fraction value. Because the current task is to calculate void fraction for a given image, BubCNN will be used as the algorithm for bubble detection, segmentation, and parameter extraction for all further work.

BubCNN can be improved further through updating its internal parameters through training with a larger dataset. This is another advantage for BubCNN, and it is therefore recommended for further work on this topic that a larger dataset for BubCNN be developed and used in training to further improve performance.

2.4 Bubble Volume calculation with uncertainty

With a set of ellipse parameters describing each segmented bubble having been extracted from an image, the next step in the greater algorithm for calculating the observed void fraction in an image is to estimate the gas volume contained by each bubble, within a range of uncertainty. To calculate the uncertainty in

an estimate of volume for a given bubble, all sources of uncertainty found within this calculation must be identified and accounted for. These uncertainties arise in the estimation of ellipse parameters which are used to calculate volume, being the major and minor axis lengths.

Although the uncertainty in the calculation of the ellipse parameters for a given bubble are calculated here, two other sources of uncertainty exist: First, there are some bubbles which are not well approximated by ellipses. These bubbles have their volumes calculated with a relatively large error when using this method. A new algorithm is required to address this problem, and it is out of the scope of the current work. Second, there is a small amount of gas volume from the tiny bubbles missed by the algorithm which is not accounted for. This gas volume is assumed to be negligible, and is ignored.

2.4.1 Sources of uncertainty in semi-axis lengths

Near/Far effect

Without stereovision, all bubbles within an image appear as if they are on the same flat plane. This is of course not the case, with the section of flow being observed containing depth information inaccessible to a single camera. As such, the optical effect that an object's apparent size changes in relation to how far it is to an observer is a source of uncertainty. In addition to this observation, it is also noted the degree of uncertainty in bubble size because of this effect varies with a bubble's location along the horizontal image plane. This is because of the cylindrical shape of the flow channel: the depth of the channel in relation to an observer (and therefore the uncertainty in measured axis lengths) is at a maximum at the centre of the tube, and at a minimum on the edges of the horizontal plane. The uncertainty in bubble size therefore is directly related to the depth of the channel in relation to an observer, which is related to a bubble's location along the horizontal plane. The following calculations are performed after image distortions from the glass observation window and from camera extrinsics have been corrected for.

Statistical methods are employed to calculate the bounds of uncertainty for axis length measurements. Before uncertainties can be calculated, a key assumption must be stated: It is assumed that bubbles are distributed uniformly in terms of the depth of the tube in relation to a viewer. Following this assumption, the mean location for a bubble's centre in terms of the depth of the tube is at the tube's centre, and the measurements of axis length taken from a given image, $\bar{a}$ and $\bar{b}$, are assumed to be taken from a bubble at this mean depth. Also following this assumption of uniformity, the standard deviation in the measurement of an axis length can be calculated as follows:

$$\Delta = \frac{(a_{max} - a_{min})}{2} \tag{2.26}$$

$$\sigma_{a,near/far} = \frac{\Delta}{\sqrt{3}} \tag{2.27}$$

A factor is needed to calculate the maximum and minimum values of an axis length. This factor must adjust the measured axis length value in accordance with the bubbles position along the horizontal axis. *Figure 2.33* shows that it is logical to relate this factor to an angle θ_n, as this value decreases as a bubble moves further away from a viewer, with an increasing value of d_n.

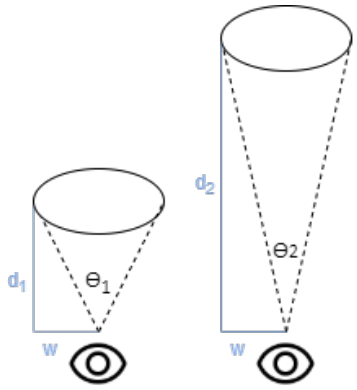

Figure 2.33- Derivation of the near/far scale factor

This factor, ϕ, can be calculated and applied to an axis length a as follows, where $D(x)$ is a function used to calculate a bubbles distance from the camera (and therefore its depth within the flow channel) based on its position along the horizontal axis:

$$\underset{max}{\phi_{min}} = \mathrm{A} \cdot \arctan\left(\frac{w}{d \pm D(x)}\right) = \mathrm{A} \cdot \frac{\theta}{2} \tag{2.28}$$

$$\underset{max}{a_{min}} = a \cdot \underset{max}{\phi_{min}} \tag{2.29}$$

d, the distance from the camera to the centre of the viewing section, is 310 mm, as can be understood from *Part I, Section 4*. *w* is equal to 4 mm, as it represents the radius of the flow channel. A constant *A* is introduced to the formula so that the factor ϕ will be equal to one at the edges of the tube, where there is no uncertainty in the true size of a bubble in terms of the near far effect, and where $D(x) = 0\ mm$. This constant is calculated to be 77.5, for the above requirement to be met.

As mentioned, $D(x)$ represents the potential error in the depth of a bubble when assuming its depth is at the centre of the tube, with x representing the bubble location along the images horizontal axis. *Figure 2.34* represents a cross section of the flow channel, and shows how $D(x)$ can be derived using the geometry of a circle:

$$y^2 + (x - r)^2 = r^2 \tag{2.30}$$

$$D(x) = y = \pm\sqrt{r^2 - (x - r)^2} \tag{2.31}$$

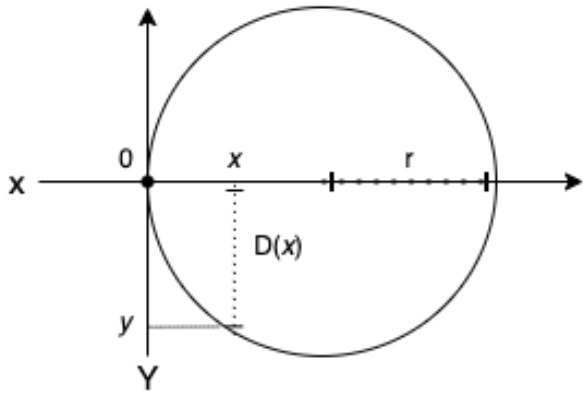

Figure 2.34-Axis used in calculating near/far uncertainty

With the internal diameter of the flow channel equalling 8 mm, *r* then is equal to 4 mm, and *x* must be a value between 0 mm and 8 mm. The camera setup was calibrated such that a 1 mm line on the plane at the centre of the tube, in terms of depth, corresponds to 22 pixels in an image of the tube for which refractive distortion and camera extrinsics have been corrected for. This calibration is used to scale the coordinates of a bubble's centre to find its real-world horizontal location in terms of *x,* along an axis like that seen in *Figure 2.34.* This calibration is also used to convert the final uncertainty value back into units of pixels. With the maximum and minimum values found for each axis length, the uncertainty in these calculations can be derived using the Equations *2.26* and *2.27*.

Blurred bubble edges

Due to limitations of image resolution, and the effect of motion blur, the edges of individual bubbles are not defined with absolute certainty. *Figure 2.35* shows that for a bubble's edge, there are three layers of pixels which could be used to describe the outer boundary of the bubble shape, with two pixels between the maximum and minimum boundary locations. It is difficult to say with certainty which of these layers represents the true bubble boundary, and which layers are a product of motion blur.

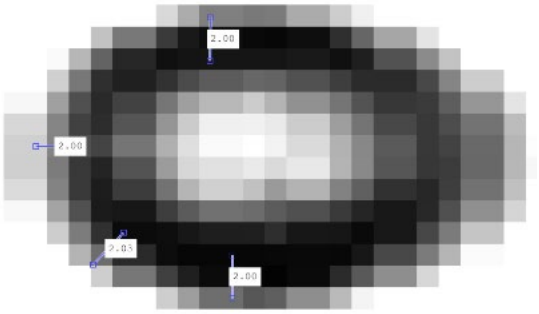

Figure 2.35-Poorly defined bubble edges; The lines and pixel distances shown represent the possible edge locations of the bubble

Herein lies a source of uncertainty in calculating the lengths of a bubbles major and minor axis. An assumption is made, that the probability of one of these three layers being identified as the bubble edge is uniform. Following this assumption, the layer detected as the bubble edge, on average, is the central layer. Following this assumption of uniformity, the standard deviation, and uncertainty in axis length measurements due to blurred bubble edges, is found as follows:

$$\Delta = \frac{(a_{max} - a_{min})}{2} = \frac{2}{2} = 1\ pixel \tag{2.32}$$

$$\sigma_{a,blur} = \frac{1}{\sqrt{3}} \tag{2.33}$$

To calculate the total uncertainty in a measurement of a bubble's semi-axes, the combined uncertainty from the above two sources must be found using the following formula:

$$\sigma_a = \sqrt{\sigma_{a,near/far}^2 + \sigma_{a,blur}^2} \tag{2.34}$$

2.4.2 Final bubble volume calculation with uncertainty

To calculate the uncertainty in each bubble's volume, the formula for the propagation of error [117] is used:

$$\sigma_V = \sqrt{\left(\frac{\partial V}{\partial a}\right)^2 \sigma_a^2 + \left(\frac{\partial V}{\partial b}\right)^2 \sigma_b^2 + \left(\frac{\partial V}{\partial \varphi}\right)^2 \sigma_\varphi^2} \tag{2.35}$$

A function of volume, $V(a, b, \varphi)$, is therefore required, where φ is the angle of the axis by which the ellipse is rotated about when calculating volume, as shown in *Figure 2.36*. To find this function of volume, a function describing a rotated ellipse in terms of its *2D x and y* coordinates is required:

$$\frac{\big(xcos(\varphi) + ysin(\varphi)\big)^2}{a^2} + \frac{\big(xsin(\varphi) - ycos(\varphi)\big)^2}{b^2} = 1 \tag{2.36}$$

This function is solved in terms of x to find the two solutions of x corresponding to each y value:

$$x_{\substack{1\\2}} = \frac{\pm ab(b^2\cos^2(\varphi) + a^2\sin^2(\varphi) - y^2\cos^4(\varphi) - y^2\sin^4(\varphi) - 2y^2\cos^2(\varphi)\sin^2(\varphi))^{\frac{1}{2}} \mp a^2 y\cos(\varphi)\sin(\varphi) \pm b^2 y\cos(\varphi)\sin(\varphi)}{a^2\sin^2(\varphi) + b^2\cos^2(\varphi)} \tag{2.37}$$

The volume of the ellipse is calculated by treating the distance between x_1 and x_2 as the diameter of a circle describing a cross sectional slice of a 3D ellipsoid at each corresponding y value, and then summing the areas of each of these slices. This is achieved through the following integration, where A represents the cross-sectional area of the ellipse at vertical location y:

$$V(a, b, \varphi) = \int_{y_{min}}^{y_{max}} A\ (y)\ dy = \int_{y_{min}}^{y_{max}} \pi \left(\frac{x_1(y) - x_2(y)}{2}\right)^2 dy \tag{2.38}$$

The variable φ, the bubble rotation angle in the direction of the plane, affects the depth calculated for a given bubble, as is shown in *Figure 2.36*. This Figure shows that for different values of φ between 0 and π/2, different volumes will be calculated for bubbles with the same major and minor axis length, as the depth of each horizontal slice corresponds to the length of the red lines within the ellipses, which are evidently unequal across rotation angles.

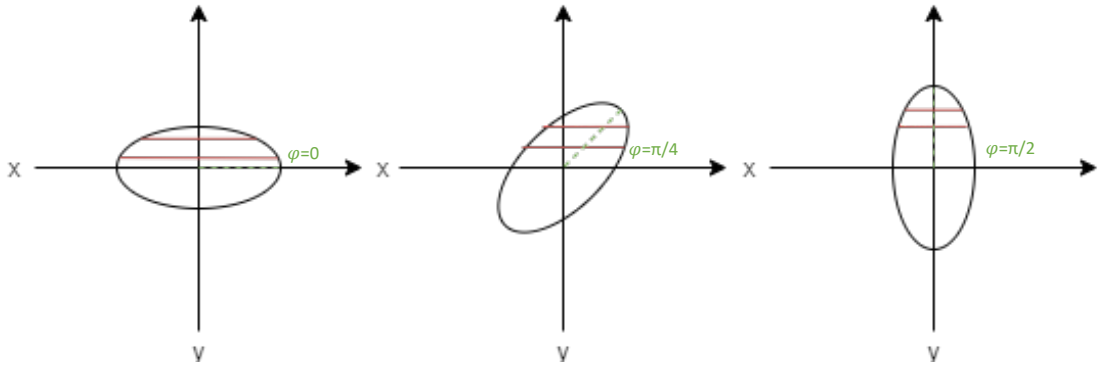

Figure 2.36-The effect of rotation angle on bubble depth

The only known information regarding the value of φ is that it is between 0 and π/2, as these angles encapsulate all possible rotations due to symmetry. An assumption is then made, that φ is described by a uniformly distributed probability density function between values of 0 and π/2. The expected value and variance of such a function are calculated as follows:

$$\bar{\varphi} = E[\varphi] = \int_0^{\frac{\pi}{2}} \varphi \frac{2}{\pi}\, d\varphi = \frac{\pi}{4} \tag{39}$$

$$\sigma_\varphi^2 = E[\varphi^2] - E[\varphi]^2 = \frac{\pi^2}{48} \tag{40}$$

With $V(a, b, \varphi), \bar{a}, \bar{b}, \bar{\varphi}, \sigma_a, \sigma_b$ and σ_φ defined, a user can calculate the volume of a given bubble, with uncertainty, as follows:

$$V(a, b, \varphi) = V\left(\bar{a}, \bar{b}, \bar{\varphi}\right) \pm \sigma_V \tag{41}$$

2.5 Final Void Fraction Calculation

The final volumetric void fraction is calculated by summing the volume of all detected bubbles, while still considering uncertainties, and then dividing this value by the volume of the observed tube section, in units of pixels. Because void fraction is a ratio, and measurements in pixels are directly proportional to the calibrated real-world measurements, the use of pixels instead of units of meters will not change the final value. The final void fraction for a given image, with uncertainty, is calculated as follows:

$$\alpha = \frac{V_{gas} \pm \sigma_{V_{gas}}}{V_{total}} = \frac{\sum_{i=1}^{N} V_i \pm \sqrt{\sum_{i=1}^{N} \sigma_{V_i}^2}}{A_{cropped} I_{height}} \tag{2.42}$$

Where *N* is the total number of bubbles, I_{height} is the vertical size of the image being analysed, and $A_{cropped}$ is the cross-sectional area of the tube, when accounting for the cropped-out regions within the

image (from the pre-processing described in *Part II, Section 2.1*). The calculation of this value is done as follows, with *Figure 2.37* for reference:

$$A_{cropped} = 2\int_{-(r-c_1)}^{r-c_2} \sqrt{r^2 - x^2}\, dx \tag{2.43}$$

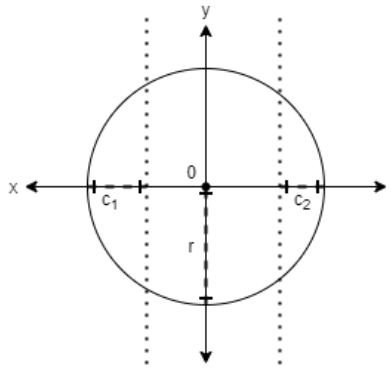

Figure 2.37-Axis used in calculating $A_{cropped}$

Where c_1 and c_2 represent the number of pixels cropped out from each side of the tube section within the image, and *r* can be calculated in pixels by using a calibrating factor. The total volume present in the viewing section is then found by calculating the cropped cylindrical volume, described by the multiplication $A_{cropped}I_{height}$, in units of pixels.

In calculating the total uncertainty in gas volume, $\sigma_{V_{gas}}$, the equation for combining standard uncertainties is used to sum the uncertainty in volume for each bubble i, σ_{V_i} (as found in the previous step):

$$\sigma_{V_{gas}} = \sqrt{\sum_{i=1}^{N} \sigma_{V_i}^2} \tag{2.44}$$

2.6 Testing

With the above algorithm and its various components described in their entirety, testing of the larger system could commence. This testing involves the evaluation of the algorithm as a method to calculate void fraction using image processing techniques. To evaluate the predictions made by this algorithm, they will be compared to existing models used to predict void fraction under similar test conditions. Through this comparison, it will be determined if the predictions produced by this algorithm agree with commonly used methods to calculate void fraction. Conversely, by comparing the results produced by this algorithm against multiple existing models, the performance of these models for CO_2 vertical up flow can also be examined. In future work, it is recommended that the algorithm here be applied to a ground truth dataset of void fraction values, with corresponding images to more accurately qualify it's performance.

The bubbly flow under investigation can be described as a gas phase dispersed within a continuous liquid phase. Because of this, and the generally excellent results [18] [31] of drift flux models for vertical flow, this model type represents the most appropriate choice to validate the results of the algorithm described in the current work against. Of the existing drift flux models, those of Bhagwat and Ghajar [28], Toshiba [27] and Rouhani and Axelsson [26] represent the best choices to model the current system, as these

model have been shown to produces accurate correlations for vertical up flow, at lower void fraction values [10] [18] [31]. The $K\alpha_h$ model of Armand [24] and Massena [25] will also be used to describe the current system due to its good performance on vertical flow [18]. The equations representing these models can be found in *Table 1.1*.

The Homogeneous and Slip Ratio models will also be included for comparison, as these models represents old and widely used methods for void fraction estimation. The slip ratio model requires values for the cross-sectional averaged velocities of the gas (u_g) and liquid (u_l) phases, with the governing equations (*Equations 1.4 and 1.5*) for this model being shown in *Part I, section 2.2*. Measuring the gas velocity is achieved as follows: for a sequence of frames extracted from a video, individual bubbles are tracked between frames, with their vertical displacements between frames being used to calculate the gas velocity. Dividing this displacement by the time interval between frames (0.735 ms) and multiplying it by a calibrating factor for pixels and meters (22 pixels/mm) allow for the calculation of how far a bubble travels between frames, and therefore, how far over a definite time interval, with this value representing velocity. Each bubble is tracked using a Median Flow Tracker [119] implemented with the OpenCV [90] Python library. For a given video, multiple bubbles are tracked across frames so that an average gas velocity can be calculated by considering a set of velocity values. *Figure 2.38* shows how individual bubbles are tracked between frames. The vertical displacements between bounding boxes describing the same bubble (with these boxes sharing a number across frames) are calculated between frames, with the average displacement across bounding boxes being used to calculate the overall gas velocity, u_g, as described above.

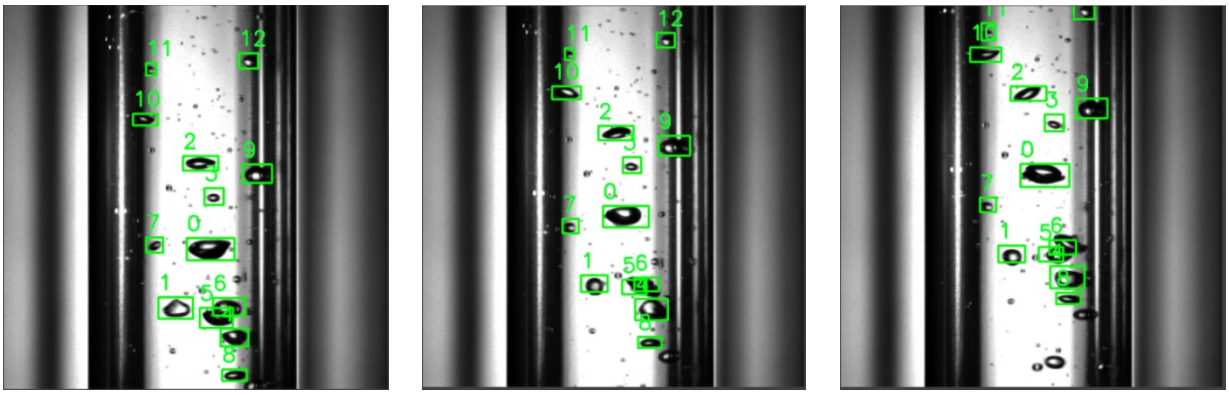

Figure 2.38-Calculating Gas Velocity; The vertical displacements of bubbles, derived by tracking bubbles across frames, are used to calculate the velocity at which the gas phase flows

To calculate the superficial liquid velocity (defined in *Part I, Section 2.4*), the following equation is used:

$$U_l = \frac{G(1-x)}{\rho_l} \tag{2.45}$$

With the mass flux, vapour quality and density of liquid being known measured quantities. To calculate the liquid velocity, as is required for the slip ratio, the following equation is used:

$$u_l = \frac{U_l}{1-\alpha} \tag{2.46}$$

When this equation is substituted in for u_l in the slip ratio model equation (defined by *Equations 1.4* and *1.5*), the void fraction term appears on both sides of the equation. To find the void fraction predicted by

the slip ratio model, by subbing in for the slip ratio using u_l and u_g, as described above, Matlab's [87] symbolic toolkit is used to solve for α.

One hundred images were used in generating the void fraction data points representing the results of the algorithm developed in this work. Flow characteristics associated with each image, such as vapour quality, mass flux, mass flow rate, fluid density, viscosity, gas velocity and liquid velocity are used in calculating the void fraction values predicted by the above-mentioned models. Uncertainty is introduced by the vapour quality term when calculating void fraction, due to the inherent uncertainty in the experimental methods used to derive this value [14]. When vapour quality is included in the calculation of void fraction by an existing model (*Figure 2.39*), and when plotting the results of the algorithm developed in the previous sections across vapour qualities (*Figure 2.40*), a conservative uncertainty bound of ±0.01 in the vapour quality value is included [14].

2.7 Results

Table 2.4-Relationship between Bubble Sum and a selection of existing models

Model	Predictions which fall within uncertainty bounds [%]	RMSE in Void Fraction Prediction
Bhagwat and Ghajar [28]	95	3.6 E-2
Slip Ratio Model	86	5.4 E-2
Rouhani and Axelsson [26]	95	4.2 E-2
Toshiba [27]	80	3.8 E-2
Homogeneous Model	89	7.4 E-2
Armand [24] -Massena [25]	94	5.8 E-2

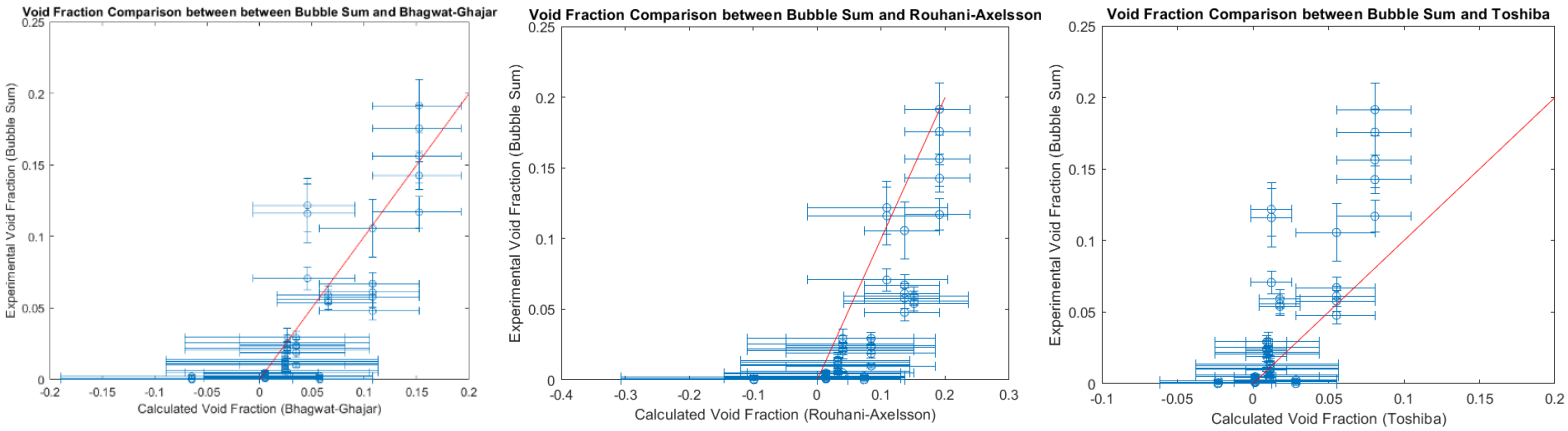

Figure 2.39-Void Fraction comparison between Bubble Sum and various high performing models. horizontal and vertical error bars represent uncertainty in both dependent and independent variables, as discussed below.

In *Figures 2.39* and *2.40*, the label "Bubble Sum" relates to the algorithm detailed in the current work. *Figure 2.39* plots the void fractions predicted by the Bubble Sum algorithm, with uncertainty, against the

void fractions predicted by Bhagwat and Ghajar's, Rouhani and Axelsson's, and Toshiba's models. In these figures, the vertical uncertainty bars are derived using the methods described in *Part II, sections 2.4,* and the horizontal uncertainty bars are a result of the effect of uncertainty in vapour quality values when calculating the void fraction predicted by an existing model. These figure shows the degree to which the two methods are correlated, with data which is accurately correlated between the methods falling on the red line. A summary of these results can be seen in *Table 2.4*. Included in this table is the percentage of calculated void fraction values, found with Bubble Sum, which have uncertainty bounds that include the predicted void fraction for each respective model. This metric can be interpreted in *Figure 2.39* as the percent of data points that have uncertainty bounds overlapping with the red line in each figure. This table also includes the RMSE value between the data produced by Bubble Sum and the predictions made by each model.

Figure 2.40 shows the void fraction predictions made by each model across vapour quality values. Included in this figure are the void fractions predicted by Bubble Sum, with uncertainty.

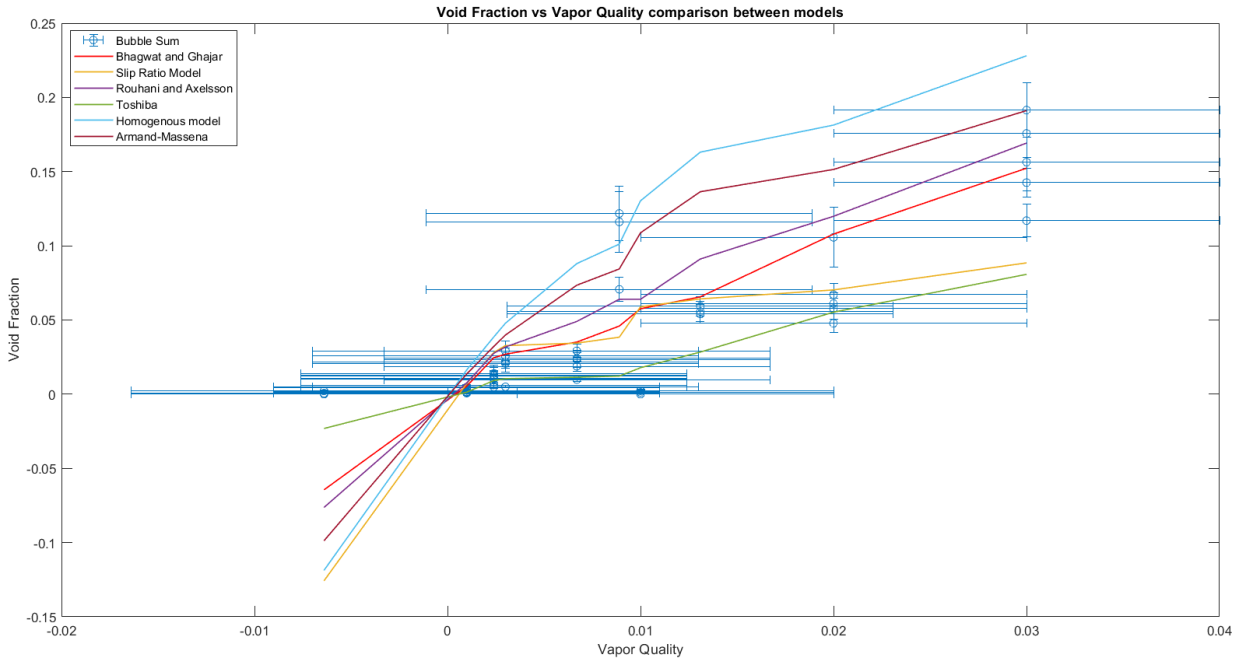

Figure 2.40-Void Fraction Vs Vapour Quality; Bubble Sum (shown with uncertainty bounds) can be compared to existing models

2.8 Discussion

The models which correspond to the results of Bubble Sum most closely are the three drift flux models used for testing. This could be expected from the results seen in the existing model comparisons for vertical flow [18] [31] [10], where drift flux models consistently achieve the highest performance for void fraction estimation in vertical bubbly flow. *Table 2.4* reveals that these models produce significantly lower RMSE values than the homogeneous, $K\alpha_h$ and slip ratio models when compared to the Bubble Sum data. Although the drift flux models outperform the others, *Figure 2.40* reveals that all the models being tested appear to follow a similar trend to the data produced by Bubble Sum. This is confirmed in *Table 2.4*, where it can be seen that the majority of predictions made by each model fall within the uncertainty bounds of the corresponding datapoints produced by Bubble Sum.

The generally high performance seen across models in predicting the results of Bubble Sum are a result of the narrow range of vapour qualities under observation. At very low vapour qualities, the models predict similar values, with greater deviations between models seen at higher vapour qualities. The fact that there is a disproportionate number of very low vapour quality (below 0.01) datapoints, coupled with the relatively large uncertainty bounds produced by the uncertainty in vapour quality values, with respect to the range of values under observation, ensures similarly high performance across models, as the large uncertainty bounds for a given datapoint almost always include all the model predictions. The above-mentioned factors produce the deceptively positive results seen in *Table 2.4*. To compare models more accurately, a much larger and less biased spectrum of vapour quality values is required.

The Bubble Sum data therefore can be said to correlate strongly with the existing models when including uncertainties, over this narrow and low range of vapour qualities, yet if a new model were to be developed using data generated by Bubble Sum, it is recommended that the uncertainty in vapour quality be reduced as much as possible, and that a much larger range of vapour qualities should be investigated. This recommendation reveals another issue regarding the use of Bubble Sum to develop new correlations: this method can calculate void fraction only for bubbly flow, which only occurs for very small values of vapour quality. Bubble Sum, then, could only be used to make correlations for very low vapour quality bubbly flow, restricting its usefulness as a measurement tool. That being said, the performance of Bubble Sum can still be evaluated against existing models for this small range of vapour qualities.

Correlations between the high performing drift flux models and the bubble sum data can be seen in *Figure 2.39*. Although Toshiba's model produces the second lowest RMSE value in *Table 2.4*, this result can be attributed to the disproportionate number of low vapour quality datapoints under investigation. Toshiba's method tends to drastically under-predict data produced at higher vapour qualities, as is evident in the above figures, and in the comparatively small number of data points falling within uncertainty bounds of this model, as seen in *Table 2.4*. This model has been shown to underpredict void fraction in other investigations [31].

When comparing the results produced by Bhagwat-Ghajar to those produced by Rouhani-Axelsson, *Table 2.4* reveals that although the same amount of datapoints are predicted by these models within the uncertainty bounds produced by Bubble Sum, Bhagwat-Ghajar achieves a lower RMSE, indicating that the data produced by Bubble Sum is best modelled by this method. *Figure 2.39* reveals that Rouhani-Axelsson tends to over predict void fraction, especially at higher vapour qualities. A particularly encouraging result produced by Bhagwat-Ghajar can be seen in *Figure 2.39*, at the highest set of void fraction predictions. These predictions all come from different images taken under the same experimental conditions. With the average of these five datapoints being almost perfectly predicted by Bhagwat-Ghajar, this model might predict the results of Bubble Sum even better when longer term averages are taken from images generated under the constant conditions.

2.9 Conclusion

The algorithm described above represents a method for which traditional image processing techniques are employed to calculate the void fraction directly from a given image of bubbly flow, with the various components of the algorithm being tested in earlier sections. The preceding section shows that the results produced by this method are in agreement with void fraction models developed for similar experimental conditions.

Although these results are positive, it must be noted that this algorithm works only for an extremely small range of vapour quality values (as is seen in *Figure 2.40*), within which bubbly flow is generated. An assumption required for this algorithm, that bubbles tend to be ellipsoidal in shape, is false for higher vapour qualities, where the interface between vapour and liquid phases becomes harder to define. As such, it is not obvious as to how the algorithm developed here could be extended to calculate void fraction for higher vapour quality flow images. The Bubble Sum algorithm therefore represents a tool which works only under very specific experimental conditions.

The agreement between predictions made by Bubble Sum and those produced by Bhagwat-Ghajar is encouraging for Bubble Sum's efficacy as a measurement tool for void fraction in low vapour quality bubbly flow. Bhagwat and Ghajar [28] generated void fraction data by using quick-closing-valves, which have been shown to produce very accurate results [45]. The fact that the Bubble Sum predictions, for very low vapour qualities, and when including uncertainties, are so similar to those produced by a model developed using such a method as QCV, points towards the conclusion that Bubble Sum predicts void fraction under the above stated conditions with relative accuracy.

To analyse the properties of two-phase CO_2 flow for higher vapour quality values, more modern computer vision techniques must be employed. In *Part III*, machine learning techniques are applied to extract flow properties from images and videos of high vapour quality flow.

Part III – Deep Learning based Computer Vision

This chapter details the research and development undertaken to produce a set of tools, for analysing two-phase CO_2 flow, based on deep learning computer vision techniques. The first section details an investigation into the practical applications of Convolutional Neural Networks, for both video and image analysis. Following this, the development of a set of computer vision based tools is described.

1 Convolutional Neural Networks

In recent years, Deep Learning has revolutionised many fields within the applied sciences [120] [121]. This boom in the application of deep learning techniques is the result of the ever decreasing hardware costs associated with these methods, which in turn leads to further research within this domain [120] [121]. Like so many other fields of engineering and the applied sciences in recent years, more and more traditional image processing tasks are being performed with the use of artificial neural networks (ANNs) and deep learning instead of by algorithmic methods [120] [121]. This is because within the field of computer vision, the essential task of feature extraction can be automated with the use of deep learning. This automated feature extraction is often performed using Convolutional Neural Networks (CNNs) and deep learning.

1.1 General Applications of CNNs

A CNN is a type of ANN which is commonly used in computer vision tasks [122] [123] [121]. These networks can learn which features within an image are relevant for achieving a specific task, removing the difficult manual task of feature extraction [123] [121]. Traditionally, algorithms and feature extraction methods would have to be manually tuned by researchers to extract the desired features. The task of manual feature selection requires deep analysis of the problem at hand, and manual feature extraction methods require time-consuming fine-tuning [121] . Conversely, with the use of deep learning, these complex algorithms can be replaced by CNNs, which are “trained” to extract the desired features from an image. These features are graphically encoded in the convolutional layers of the CNN and are learned through the training of the network for a specific task [77], with the features identified by a CNN being objective and optimized for that specific task. This can lead to feature extraction and identification which is counter-intuitive to a human user, yet well suited for the task [80] [81]. In this way, CNNs can automatically produce well suited features for a specific task, replacing the need for complex image feature engineering. Modern CNN architectures have produced ground-breaking results in tasks related to image feature extraction [122] [124] [125].

While these feature maps are present in all CNNs, the output of the network is dependent on the type of learning problem the network has been designed for. For instance, CNNs designed for image classification problems output a set of values corresponding to a predefined set of classes, with the analysis of these values revealing which class the network determines the input image to be. These values are found in the final “Classification layer” of the network, where output neurons corresponding to each class are activated to varying degrees depending on the input. The final classification is taken from the corresponding output neuron which is activated with the largest magnitude [80] [81] [82].

On the other hand, CNNs designed for detection output the likelihood of a specific object being detected within each region of the image [76] [77]. Regression CNNs output continuous values which represent specific characteristics found within an image. In related work, regression models are used to extract the specific geometrical parameters of bubbles detected in earlier stages of a network [77] [76]. Regression

networks can be created by implementing a single output neuron in the final layer of the neural network, which outputs the final estimated value. The various output types made available by CNNs make them a robust tool for completing computer vision tasks related to multi-phase flow, as is evident in the expansive works related to the task at hand [74] [75] [76] [77] [78] [80] [81].

CNNs analyze regions within an image by applying a sliding window across an image, and generating task-specific, probabalistic results pertaining to each region. Regional Convolutional Neural Networks (RCNNs), on the other hand, make use of region proposals to process only the useful regions within an image [126]. These regions are found by using a small neural network to detect features within the image which correspond to regions where an object of interest is likely to be found [126]. This reduces the computational cost and run time when using the network, as only specific, optimal images patches are fully processed for a given task, rather than the whole image.

These different utilizations of the image features extracted in previous convolutional layers are implemented in the final output layers of the networks. This allows for the customization and modification of pretrained models for a wide range of problems, because the learnt features encoded in the convolutional layer of these networks may be relevent for more than one type of network output [121]. It is possible to train a network by freezing the weights assigned to these convolutional layers, while re-learning the weights in the final layers to produce a different output. In fact, the entire structure of the final few layers used to produce a specific output, by making use of the features produced by the convolutional layers of the network, can be customized, so that image features learnt for one task can be utilized for a different one [121] [127]. This process is know as transfer learning, and it allows for the development of new networks from pretrained existing ones [127]. Apart from its usefulness in allowing for the customization of a networks output, transfer learning is useful in cases when a limited dataset is available to the user, as one can access the network weights, representing features, which were learnt by a network with the same architecture, but by using a larger dataset, and apply them to their own network. This is useful if similar image features are found between the larger and smaller datasets. When using these pretrained weights as initial values for a network, the further training of the network, using new data, adjusts the existing features extracted by the network to be optimized for the new data [127].

Network architectures which perform well on a broad set of computer vision problems can be applied to novel tasks by training them on new datasets. This allows for a user to make use of high performing architectures without having to design such a network from scratch. To determine the general applicability of these architectures, they are often tested on multiple datasets in their developent, with networks that perform well on a variety of datasets being said to generalize well [122] [123] [124].

1.2 Video analysis using CNNs

If an image is a 2D signal made up of intensities along axes of height and width, then a video is a 3D signal, with its intensities being defined along a height, width, and temporal dimension. There are a variety of ways to extend the techniques used in the analysis of single frames for the analysis of sequences of frames, with many of these methods making use of CNNs within their structures. This section details a selection of commonly used techniques for video analysis in the field of computer vision.

1.2.1 Fusion models

Karpathy et al. [128] present the concept of fusion model categories, whereby a model is defined by the point in the flow of data through an architecture at which temporal information is incorporated. An Early

Fusion model is one which incorporates the temporal information in a sequenced set of image frames at the beginning of the network. This can be done by "stacking" the images into a single image tensor, and using this stack as the input to a single network [128] [129]. Early Fusion allows the network to detect local motion features between frames, such as direction and speed of the optical flow between pixels across frames.

The idea of a Late Fusion model is that the temporal information is incorporated at the end of the architecture, with individual frames being passed through separate network paths, and the temporal information being incorporated by passing the ordered outputs from the individual frames into another network [128]. This model can be used to extract averaged motion characteristics across frames. Temporal Feature Pooling networks make use of late pooling [130], as do archictectures which make use of Recurrent Neural Networks (RNNs) for temporal information fusion [131]. Two-Stream networks implement late fusion in the fusion of the feature outputs from the two different network streams [129].

For both early and late fusion, the length of the set of frames used as inputs to the network must be predefined; Videos vary in length, yet network architectures must be designed for inputs of a fixed size. Thus, these fusion models are designed to classify video clips of predefined length from the larger video. To achieve this, before a video is passed into these architectures, it must be truncated into clips of the required number of frames, with predictions being made on each clip [128].

1.2.2 Single-Frame models

A Single-Frame model of video classification is one by which the classification of an entire video is found by classifying a single frame from the video using standard CNN architectures [128]. This structure does not account for temporal information in the video, and hense has been considered a naïve approach to video classification in other works [130]. This model is most commonly used as a baseline for comparison to other methods of video classification [130] [128] [129] [132]. Karpathy et al. [128] show that their attempts at integrating temporal information into video classification only marginally improved upon the performance of a single frame model. This does not point to the irrelevence of temporal information in video classification, but rather to the inherent difficulty of its succesful incorporation into a neural network.

1.2.3 Temporal Feature Pooling models

Temporal Feature Pooling is a late fusion method which considers non-sequential temporal information [130]. This method is an extension of the single frame model, as it works by passing a sequence of individual frames from a video through separate identical single frame networks (CNN networks), and then performs a pooling operation on a set of features produced by each parallel network. The sequence of the frames is lost during this pooling operation. This pooling operation can take the form of mean or max pooling, and the point within the model architecture at which feature-pooling occurs is used to define that architecture [130]. These pooled features are passed into a set of fully connected layers in order to generate an output for the input data.

One such architecture is known as late temporal feature pooling. In this architecture, a sequence of images are passed through identical parallel networks, with these networks including both convolutional and fully connected layers. The pooling operation then takes place at the output of the fully connected layers, so that a final classification can be made based on a set of pooled, high-level image features [130]. These high-level image features do not contain spatial information describing the features within an

image, but rather they represent high-level abstractions describing semantic characteristics of the image [130]. Therefore, late pooling does not preserve spatial information information during its pooling operation.

A high performing architecture for temporal feature pooling is that of the convolutional pooling model [130]. This architecture pools features taken from the outputs of the final convolutional layers found in each network from a set of parallel networks. In this way, features are pooled across a sequence of images. These pooled features are then passed into a series of fully connected layers to make the final classification [130]. These pooled features represent high-level image features, describing spatial information within the image. The pooling operation for convolutional pooling therefore preserves spatial information across the time domain [130].

Feature pooling models make use of the spatial information across a set of frames, but do not account for the temporal information between them, as the frame order is lost in the pooling layers. This model is commonly used as a baseline for comparison for other video classification methods [129] [132] [130], but it has also been shown to perform relatively well for video classification tasks in its own right [130] [132].

1.2.4 3D Convolutional Neural Networks

With an input represented by the 3D tensor of stacked images created by implenting early fusion to a set of frames, using 2D convolutional filters allows for the network to learn relationships between spatial features across the set of frames, yet the network can not learn the temporal features directly, as these features exist in a third dimension. 2D convolutional layers therefore lose temporal information with every convolutional operation performed on the 3D input, as these temporal features are not preserved by a 2D convolutional filter [133]. For a network to learn the temporal features, 3D convolutional kernels must be used in the convolutional layers of the network, along with 3D pooling layers in place of 2D pooling [134]. Karpathy et al. [128] made a direct comparison between early fusion, late fusion, single frame models, and 3D CNNs (in the form of their Slow Fusion Network), with the 3D CNN consistently performing the best in video classification accuracy across multiple datasets. 3D CNNs have many more learnable parameters than 2D CNNs due to the increased dimensionality of the network, and as such, real time applications of such networks are extremely limited, with no succesful examples of such a network found at the time of writing. Additionally, the training of this increased amount of network parameters requires more memory for training, and leads to 3D CNNs having a tendancy to overfit to training data when training with smaller datasets [135].

1.2.5 Two-Stream Networks

Two-Stream architectures contain two parallel network paths, with one trained to extract temporal features, and the other to extract spatial features. Late fusion is then implemented to make use of features from both network paths in making a final classification regarding the input data [129].

The temporal network is trained using optical flow data to encode the temporal information describing the patterns of motion associated with objects between frames [130] [129]. Optical flow describes the motion of related pixels between image frames. There are multiple methods for generating this data, with optical flow estimation representing an ongoing problem in the field of computer vision [136] [137] [129] [138]. While the spatial stream is trained using 3-channel input images (with each channel respectively correspnding to a red, green and blue intensity), optical flow data can be used to train the temporal stream, by applying a 2-channel optical flow image at the input, with the two channels representing the

horizontal and vertical components of optical flow respectively [129]. *Figure 3.1* shows a representation of the optical flow between two frames using the high performing Farneback algorithm [138]. This representation reveals small changes in pixel intensity between images, with this information potentially being valuable in the classification of flow regime.

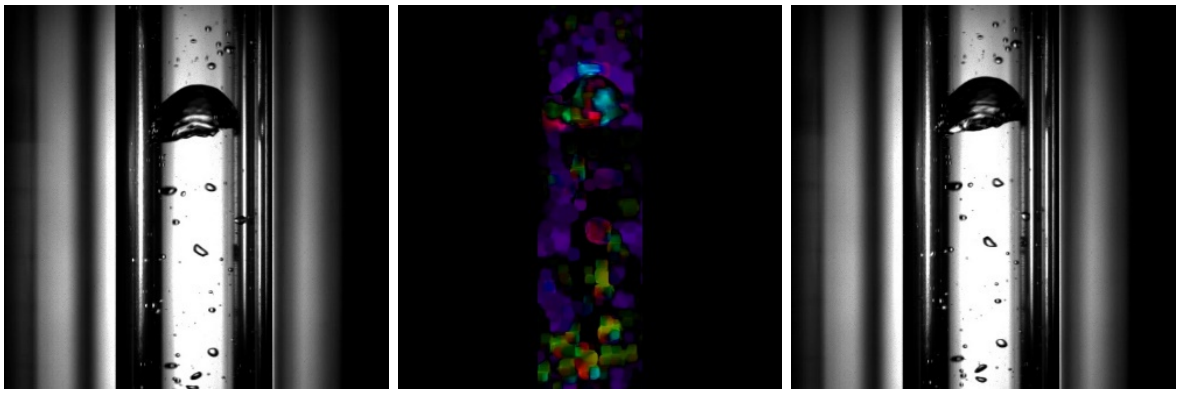

Figure 3.1- Optical Flow between frames; The image of optical flow (Centre) is extracted using the other two frames. Two of the three RGB colour channels (red and blue) shown represent the horizontal and vertical magnitudes of optical flow respectively, with the third channel (green) used to represent the direction of flow

Early fusion has also been used to stack optical flow images at the input of the temporal network in order to extract features related to motion in the video over a sequence of frames, rather than just between two frames [129]. Single frame optical flow models have been implemented for comparsion [129] [130], with single frame optical flow models consistently outperforming single frame spatial models in video classification tasks [129] [130]. Two-stream models have also been extended to incorporate RNN networks at their output to improve results through the integration of additional temporal information [130].

Two-stream networks have been shown to achieve state of the art performance in video classification using benchmark datasets [129] [130], yet real-time application of this network type using optical flow has yet to be achieved. This is because of the high memory footprint associated utilizing two parallel networks for classification, as well as the additional processing tasks required for extracting optical flow data. However, work has been done on using motion vectors instead of optical flow to represent the temporal stream of a two-stream network in order to move towards real-time performance [139].

1.2.6 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) represent a type of network architecture used in the classification and detection of patterns in a sequenced dataset. RNNs are equipped to analyze sequences due to the information transfer operations within these networks. While standard neural networks pass information forward through the network in one direction, RNNs transmit data forward and recursively, with the previous hidden state (a unique component of RNNs) of a network being passed back into itself for the next pass of input data through the network [131]. RNN therefore have a form of internal memory, provided by the persistence of the hidden state throughout the passing of sequenced data through the network. Because the output is fed back into the network for each data element in the sequence, all future outputs dependent on previous computations.

Traditional RNNs are susceptible to the common issue in back propogation, of exploding and vanishing gradients [131]. In cases of exploding gradient, the gradient used to update the internal parameters of a network increases with every time step, resulting in weight changes which are too big to reach precise and desirable weight values. Conversely, in the case of vanishing gradient, this gradient decreases with every time step, resulting in a situation where internal weights are updated with insignificant changes [140] [131]. Long Short-Term Memory Units (LSTMs) [141] are introduced within RNN networks specifically to solve these issues relating to gradient, allowing RNN networks to learn over a much greater number of timesteps. While the internal state of an RNN unit is defined purely by its hidden state, LSTM units make use of two different components to define the current state: A hidden state, corresponding to the short-term memory recursively passed through hidden units, and the internal cell state, corresponding to the long-term memory of the unit [141]. Instead of updating these states purely through backpropogation and gradient descent, LSTMs make use of gated cell structures to determine how these states are updated with each time step, removing the issues of exploding and vanishing gradient [142] [141].

The types of sequenced data to which an RNN, and by extension, LSTM network, can be applied are varied, with common applications including natural language processing and speech recognition [143], text recognition and sentiment analysis [144], and time series forecasting [142]. These network types have been applied to the feature outputs from CNN networks, to complete various tasks which make use of sequenced image data. The application of natural language processing and text analysis can be extended by performing these tasks on text extracted from images containing text using CNNs [145]. Image captioning has also been achieved by classifying a set of objects and actions in a given image using CNNs, and then using the natural language processing capabilities of RNNs to create a sentence describing the image using these classifications [146]. LSTMs have also been used in identifying transition events in image features extracted from a sequence of input images using CNNs [147]. Superb results in video classification have been achieved using LSTMs, by implementing a method where a sequence of image features extracted by CNN networks is passed into an LSTM network, and then using the outputs of these LSTM networks, a classification can be made regarding the set of input frames [130] [132] [133] [148] [149]. CNN to LSTM networks have online capability, with this architecture being applied to the computer vision problems of object detection and tracking in real-time [150] [151]. There has also been work done in reducing the computational requirements of LSTM and RNN networks, which could be applied to a CNN + LSTM network to move towards real time video classification [152].

1.3 Optimization and Loss functions

The learning process, whereby a CNN network is trained to identify a set of image features, requires a labelled training dataset, with these labels representing target outputs by the network, for their corresponding input data, relevant to the task at hand. The networks are trained using gradient descent and backpropagation over time. This training updates the weights of the network with the aim to generate network parameters which produce a desired output. The degree to which the weights are updated is dependent on the error, or loss, produced between the output of the network for a given input and its target output value.

A variety of loss functions exist, with different functions being suitable for different network configurations and learning tasks [153]. A popular loss function used for regression tasks is the Mean Squared Error (MSE) function. This value represents the average squared difference between every

prediction and its corresponding target value, across the entire dataset. This loss function can be expressed as follows, with $\hat{y}$ representing the predicted output and y representing the target output:

$$MSE = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 \tag{3.1}$$

By optimizing a network to minimize the MSE, the network is trained to produce an output with a value approaching that of the target value. This works well for regression problems, where the aim is for the network to produce a continuous output value identical to the target value.

Different loss functions are suitable for classification problems, where the target value (a class label) is not related directly to the activation values at the output of the network. One such loss function is the Cross-Entropy Loss, which works to maximize the activation of the desired output neuron and minimize all other activation outputs. In this way, the Cross-Entropy Loss optimizes the network to differentiate clearly between classes [154]. This function achieves this by calculating the loss based on the difference between the target probability distribution function (PDF) and the PDF produced at the output of the network. The target PDF is such that the desired class label has a probability of 1, and all other labels have a probability of 0. By optimizing the network to achieve this kind of PDF across the firing of its output neuron's, it produces the result described above. Cross-Entropy loss functions have also proven to train faster, and with better generalization, than other loss functions for classification tasks [155].The equation used to calculate the Cross-Entropy Loss can be seen below:

$$L = \sum_{i=1}^{n} -y_i \log_2(\hat{y}_i), \qquad for\ n\ classes \tag{3.2}$$

In this function, y represents the target probability (being a value of 1 for the desired output class and 0 otherwise) of a given class n, and $\hat{y}$ being the Softmax probability (being a value between 0 and 1) of that class, as predicted by the network. This means that the networks prediction of the target output class is the only prediction that contributes to the overall loss. The function is logistic in nature, so that the loss is larger when the prediction for the desired output has a probability closer to zero, and smaller when its probability is closer to one. This function is based on the idea of entropy, as it relates to information theory, where entropy refers to the level of uncertainty in the output when processing an input variable [156].

A function which utilizes the loss value to update the internal parameters of a network is known as an optimization function. These functions use gradient descent algorithms to minimize the overall loss of the network through the updating of the network's internal parameters (weights). A commonly used optimization function is Stochastic Gradient Descent (SGD), which has proven to generalize well, and converge relatively quickly to a minimum loss value [157]. Another popular optimization function is Adaptive Moment Estimation (Adam) [158]. This function makes use of momentum, which considers past gradients from the gradient descent algorithm when updating network parameters.

1.4 Other Machine Learning Techniques Used in Computer Vision Tasks

The approaches described above relate to Deep Learning, a subfield within the larger field of Machine Learning. Two other machine learning approaches which might be useful in the study of two-phase flow are those of the Support Vector Machine (SVM) and the K-means clustering algorithm.

SVMs discriminate between classes by finding an optimal hyperplane for separating the multi-dimensional input data belonging to the various classes [159]. This method can be used as an end-to-end approach for differentiating between labelled objects within an image [160], although the CNN based deep learning approach for image feature extraction and classification has been shown to perform better in this task [161]. SVMs do not perform as well as CNNs in terms of image feature extraction, but they have been used successfully to classify high-level image features extracted using CNN layers [129] [162]. SVMs have been used to classify flow regime in terms of low-level textural features [81].

While the techniques described above fall within the Machine Learning sub-category of supervised learning techniques, K-means clustering is used for unsupervised learning. This means that target labels are not used when training the algorithm for classification tasks, as the separation of input data into distinct classes is achieved through the application of the unsupervised learning algorithm [163]. K-means clustering achieves this by partitioning the input data into *K* different classes, which are grouped in such a way that the Euclidean distance between datapoints within the same class, and the mean location of the datapoints within that class, is minimized [163]. There are many methods of selecting the optimal number of clusters [164]. An old, yet still popular, method for selecting *K* is known as the Elbow method [164]. This method considers the distances between cluster mean locations, and selects the number of clusters at an "Elbow point", where partitioning the data into more clusters does not significantly affect the average distances between cluster means [164]. Within the field of computer vision, K-means clustering has been used for tasks of image segmentation, by clustering pixels which have similar intensity values, and which are nearby each other [165].

2 Flow Regime Classification using Computer Vision Techniques

The goal of this section is to produce a tool which can accurately classify the flow regime of vertically up flowing two-phase CO_2 based on image data captured using the test rig described in *Part I, Section 4*. This is useful in generating flow pattern maps, as the transitions between flow regimes can be automatically tracked and associated with changing flow parameters. The development of this tool was an iterative process, with four unique implementations being developed in total. First, a standard CNN was used to classify the flow [122]. This implementation, and its subsequent failure, revealed that temporal information would be crucial in the classification of flow regime. The single frame model was then expanded upon to include temporal feature pooling. Following this, a two-stream network [129] was implemented, which also presented a set of issues. From this, the final implementation was developed: a CNN is used to extract low level image features for each image in a sequence of images taken from a video clip. These low-level image features are then passed into a deep LSTM network [166] which is used to produce the final flow regime classification. This architecture is also extended to be able to predict the average vapour quality for a set of input images. This process of development is detailed in the proceeding sections.

The practical implementation of all work described in this chapter was done in Python [88], with all deep learning tasks making use of the PyTorch Python library [91].

2.1 Flow Regime classes under investigation

The following attempts at classifying images of two-phase flow within flow regime classes make use of eight distinct classes, based on those found in the literature for vertical up-flow [11]. They are produced over the entire range of possible vapour qualities (between zero and one) ensuring that data for all major

flow regimes is generated. The flow regime classes used, in order of ascending vapour quality, for fixed flow conditions, are as follows:

1. Liquid-Bubbly: A few tiny bubbles are dispersed within a continuous liquid phase.
2. Bubbly: Many small bubbles are dispersed within a continuous liquid phase.
3. Bubbly-Slug: Many small to medium sized bubbles flow within a continuous liquid phase, with a few large bubbles appearing.
4. Slug: Large Taylor bubbles flow amongst many small to large bubbles, in a continuous liquid phase.
5. Slug-Churn: Larger Taylor bubbles (long semi-cylindrical bubbles) are observed, with their tails appearing as churn flow, where gas and liquid phases are turbulently mixed.
6. Churn: Gas and liquid phases are turbulently mixed, with no clear interface between the two
7. Annular: A gas core forms at the center of the flow channel, with liquid appearing at the walls of the channel.
8. Mist Flow: A continuous gas phase flows with a few tiny liquid bubbles being observed.

A typical image from each of these regimes can be seen in *Figure 3.2*.

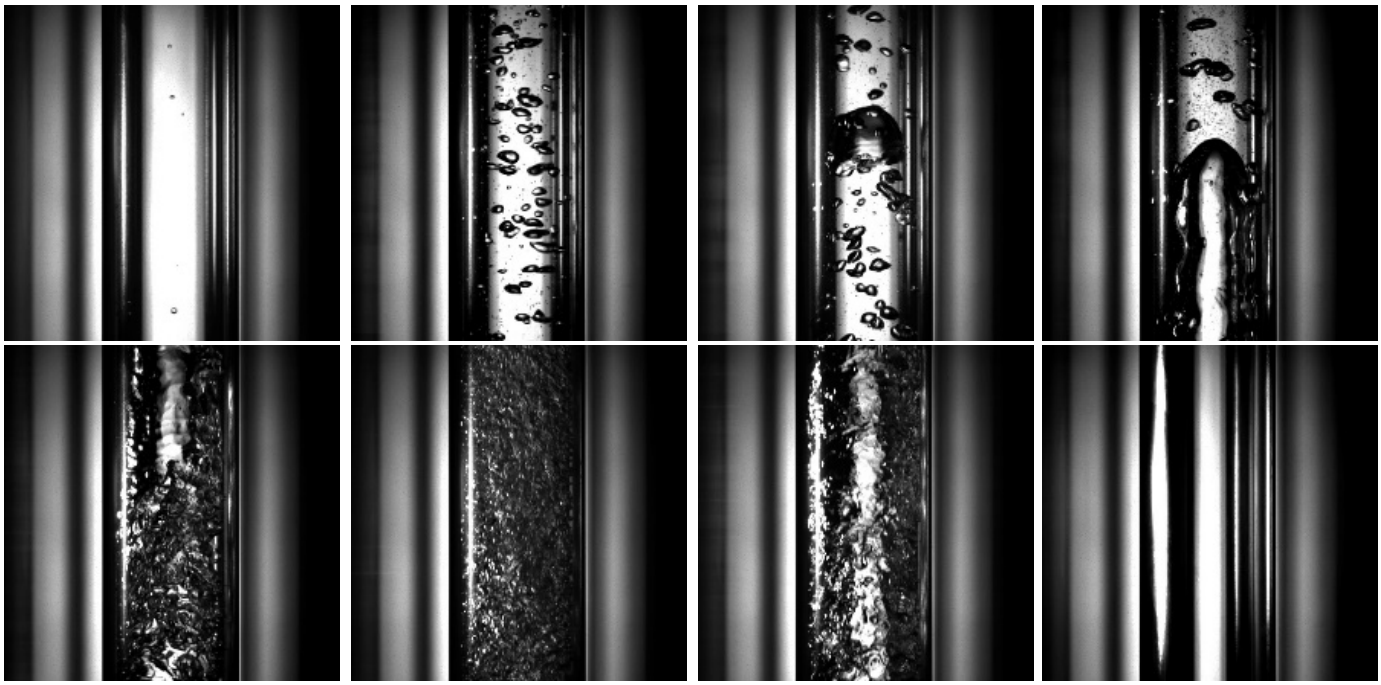

Figure 3.2-Flow Regimes under investigation; (Top row, left to right) Liquid-Bubbly, Bubbly, Bubbly-Slug, Slug, (Bottom row, left to right) Slug-Churn, Churn, Annular, Mist

2.2 Single Frame Model implementation

The first attempt made towards achieving flow regime classification using computer vision techniques was to develop a CNN based image classifier, trained to classify images of CO_2 flow within one of the above specified classes. This implementation was achieved using ResNet101 architecture [122], with this network being selected for its state-of-the-art performance in image classification.

31900 images were used for training and 7975 images were used for validating. This split of the data was random to avoid any biases in the datasets, and all flow image classes were balanced. A cross-entropy loss function was used for training, with this loss function selected because of its excellent performance in classification tasks [155], and a Stochastic Gradient Descent (SGD) optimizer was employed for its good

generalization [167]. The optimizer was defined with a momentum of 0.9 and a learning rate of 1E-3. A batch size of 10 frames was used, with the network being trained over 30 epochs, taking about four hours to train on the digital workstation described in *Part I, Section 4*. All the above hyperparameters were selected by using a grid parameter search (with parameter ranges of 1E-5 to 1E-2 and 5 to 15 for learning rate and batch size respectively).

The development of this model was not extensive due to the relatively poor accuracy seen in its results. The model was not able to achieve a classification accuracy of more than 69%. This poor result can be linked to the confusion matrix seen in *Figure 3.3*. Used to interpret results of classification networks, confusion matrices show the distribution of predicted output labels (along the horizontal axis) for each input class (shown along the vertical axis). They are revealing in that they show which classes a network might get confused between, and which classes a network classifies consistently well.

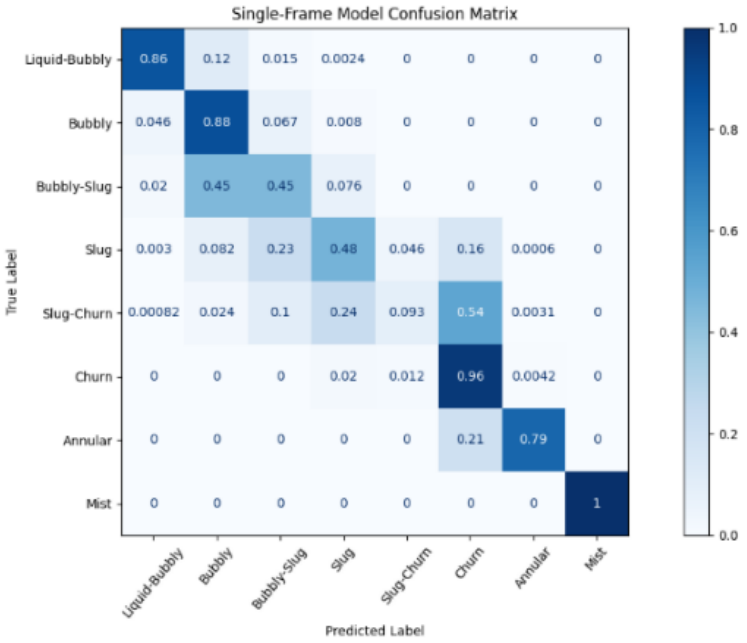

Figure 3.3- Normalized Single-Frame Model confusion Matrix

This figure shows that the network struggles to accurately classify flow regimes which have highly transient visual features. The flow regimes with very low (Liquid-Bubbly, Bubbly) or very high (Churn, Annular and Mist) vapour qualities present consistent image features across images within their class. This results in the networks good performance on these more consistent classes, and the large amount of confusion for the more transient classes. Transience in the image features of a given class causes this confusion, as it leads to images having shared features between classes, resulting in the network misclassifying those images. This effect can be seen in *Figure 2.36*, where the top three images come from the Annular class and the bottom three come from the Slug-Churn class. The top three images have common image features and are classified correctly by the network. In contrast, the Slug-Churn images are each classified in different and incorrect classes, with the first image being classified as bubbly flow, the second as slug, and the third as churn.

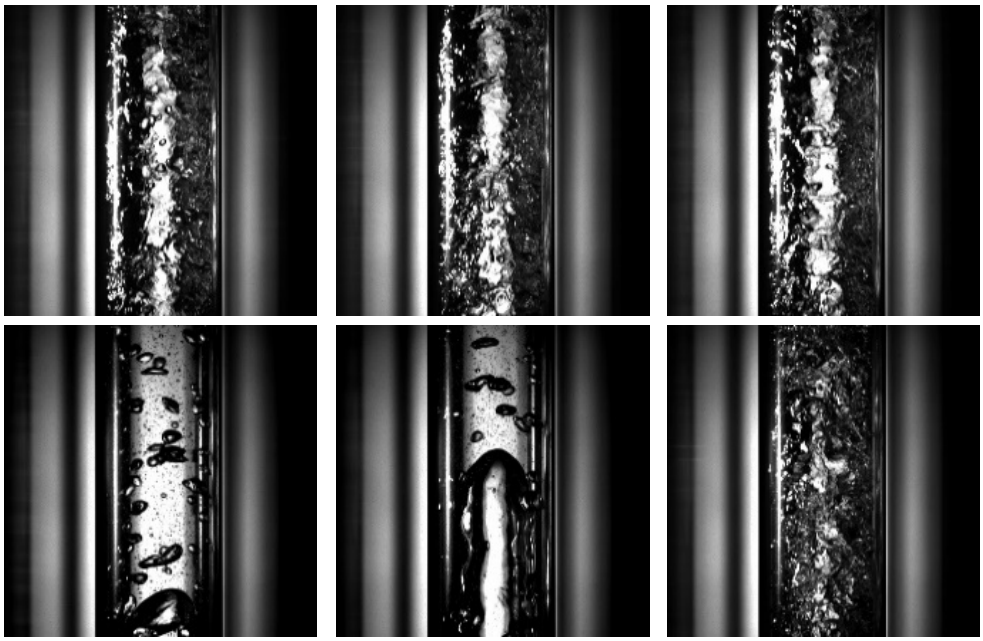

Figure 3.4- Flow Regime Misclassification by the Single Frame Model; Similar Annular images (Top row) are well classified, but more transient Slug-Churn images (Bottom row) are misclassified into Bubbly (Left), Slug (Centre) and Churn (Right) flows.

Following these results, and upon reflection, it was observed that an individual's perception and definition of flow regime is not only characterized by the spatial distribution of the gas and liquid within the two-phase flow, but also the way in which the unsteady flow moves and changes over time. For instance, a transitional flow regime might appear distinctly as one flow regime in a single frame, but it is defined as transitional based upon how the flow behaves across multiple frames. This means that temporal information must be key for image-based flow regime classification. From this observation, the proceeding implementations are developed to consider temporal information in their classification of flow regime.

2.3 Temporal Feature Pooling Implementation

The single frame model was expanded upon with the aim to achieve flow regime classification with the use of temporal feature pooling. As mentioned, this method uses information from multiple frames from a video clip but does not account for the ordering of these frames. The same dataset and training methods used in the development of the single frame model are applied here.

Both late feature pooling and convolutional feature pooling was implemented using features produced by the single frame model, with results shown below. Average and max pooling were both implemented, with average pooling producing better results in all test cases.

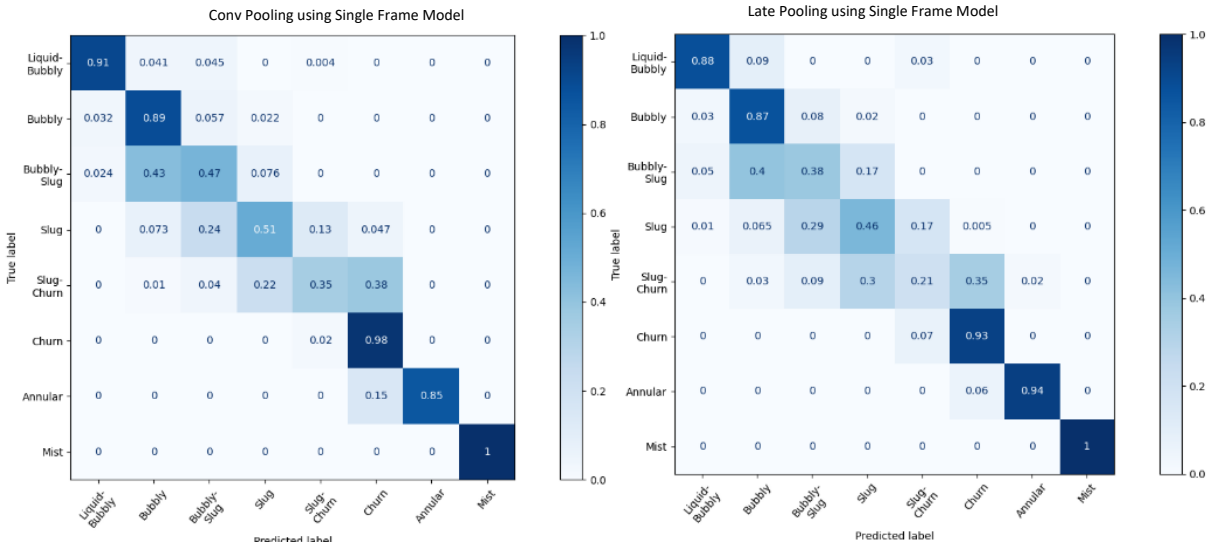

Figure 3.5-Feature pooling normalized confusion matrices (Single Frame); (Left) Convolutional feature pooling, (Right) Late feature pooling

As is consistent with the literature [130] [128], the convolutional pooling model outperforms the late pooling model. This is most likely because of the former's preservation of spatial information in its pooling operation [130]. These figures show that convolutional pooling improves performance of the single frame model, with a new accuracy of 75% achieved by the convolutional pooling network.

Section 2.5 details the development and implementation of a model, *FrameNet*, which can accurately classify (See *Figure 3.11*) images of two-phase flow within a set of flow image classes, which are different to the specified flow regime classes. This distinction between class sets is introduced because similar flow images commonly occur across flow regime classes. While this network is utilized for transfer learning to produce the final CNN+LSTM network described in *Section 2.5*, its successful classification results (See *Table 3.2*, and *Figure 3.11*) warrant an investigation into the use of features extracted by this model for temporal feature pooling, with the aim to achieve accurate flow regime classification.

The late pooling model produced by FrameNet performs slightly worse than that produced by the single frame model, with accuracies of 69% and 71% achieved by those networks respectively. This is because the high-level features produced by the single frame model are directly related to the flow regime classes, whereas FrameNet produces features related to flow image classes, adding an additional level of complexity for the network in its classification of flow regime.

The convolutional pooling network produced by FrameNet improves upon the one produced by the single frame network, with the FrameNet convolutional pooling network achieving an accuracy of 77%. This improvement indicates that the spatial features learnt by FrameNet might be more robust and useful descriptors of the flow images, for tasks of flow regime classification, than those learnt by the single frame model.

This method improves upon the single frame model in its classification of flow regimes by around 9%, yet the performance of the new network on transitional flow classes is still unsatisfactory when compared to its performance on more stable flow regimes, such as mist and annular flow. This suggests that more sophisticated methods of incorporating temporal information are required to effectively classify these transitional regimes.

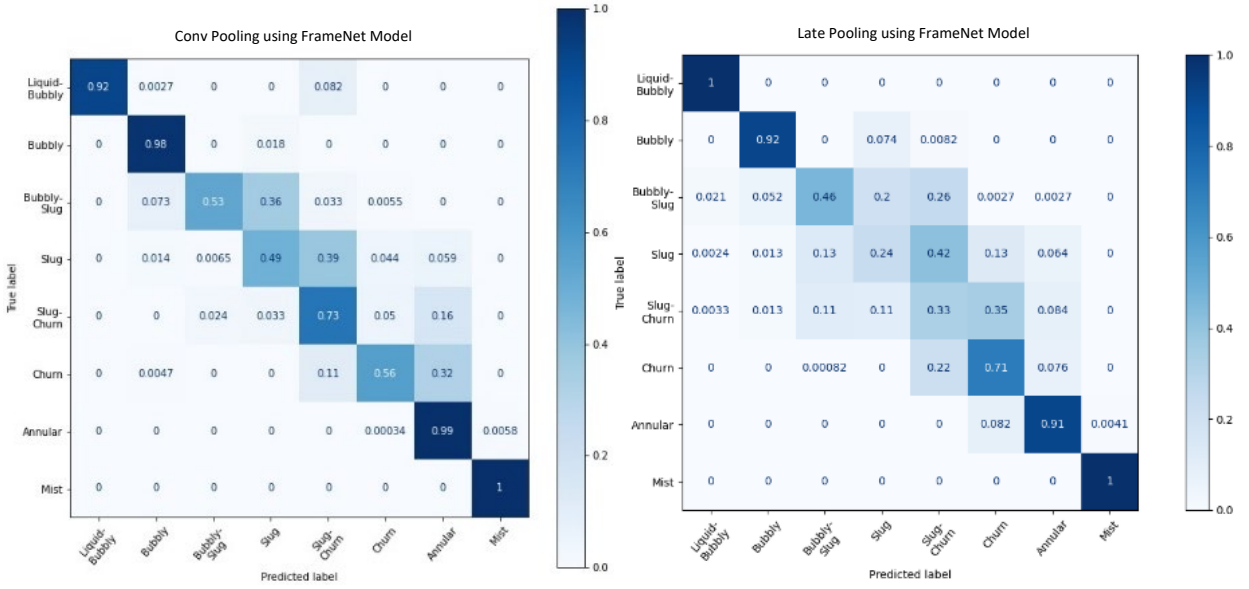

Figure 3.6- Feature pooling normalized confusion matrices (FrameNet); (Left) Convolutional feature pooling, (Right) Late feature pooling

2.4 Two-Stream Model implementation

A two-stream model based on the work of Simonyan [129] was implemented to include temporal information in the classification of a given image. The architecture of the network was largely the same as Simonyans, except for each network stream, ResNet101 [122] was used. This network was used for its excellent performance in classification tasks [122]. The optical flow between two image frames was produced using Farneback's algorithm [138]. The temporal CNN makes use of 10 stacked optical flow images, with an example of an optical flow image shown in *Figure 3.1*. This number of stacked images was found to be optimal by Simonyan [129]. Late fusion between the two networks was performed by averaging the outputs of the networks. Although performing fusion using an SVM was shown to produce better results by Simonyan, an averaging method was used for ease of implementation, and because the performance drop when using fusion by averaging instead of fusion by SVM is small.

31900, 3988 and 3988 images were randomly split for training, validating and testing respectively, with an optical flow image being produced between each consecutive frame from a given video of flow. A cross-entropy loss function was used for training, with this loss function selected because of its excellent performance in classification tasks [155], and a Stochastic Gradient Descent (SGD) optimizer was employed for its good generalization [167]. The optimizer was defined with a momentum of 0.9 and a learning rate of 1e-3. A batch size of 10 was used, with the network being trained over 30 epochs, taking about four hours to train on the workstation described above. All the above hyperparameters were selected by using a grid parameter search.

The Spatial stream CNN trained to achieve an accuracy of 68% on the test dataset. It is unsurprising that it performs identically to the single frame model because they are identical networks. The temporal stream achieves a better accuracy of 75%. This improvement can be attributed to the optical flow stacking in the temporal stream, which allows the network to interpret motion information over a longer time interval. This allows the temporal stream network to classify flow regime based on features related to motion, and therefore time, allowing for better interclass differentiation by this network. This is shown in

Figure 3.7, where the temporal stream network improves upon the spatial stream network for transitional flow regimes of bubbly-slug and slug-churn, while performing worse than the spatial stream for classes of churn and slug flows. This dip in performance could be a result of the ocular flow images being ill-suited to define certain flow regimes. For instance, the ocular flow images for churn and annular flow are visually very similar, resulting in the misclassification of churn flow by the temporal stream network.

The averaging fusion at the output of both networks achieves flow regime classification accuracy of 77%, a similar increase in performance to the temporal stream as that seen by Simonyan [129]. This improved result is shown in *Figure 3.7*, where it can be seen that the confusion matrix produced by averaging fusion benefits from the per-class advantages found in each stream.

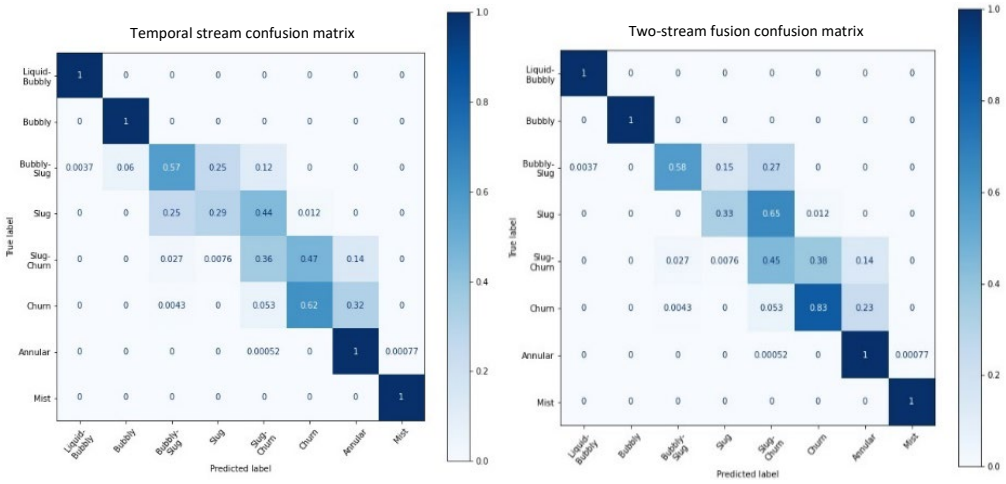

Figure 3.7-Normalized Two-Stream confusion matrices; (Left) Temporal stream, (Right) Output of averaging fusion layer

This improved accuracy of the temporal stream over the spatial stream encourages the idea that temporal features are useful when differentiating between flow regimes, but the results shown in this section make it clear that the use of early fusion, in the form of image stacking, and the use of ocular flow images, to model temporal features, does not preserve all the temporal information required to accurately differentiate between transitional flow classes. A final implementation is developed in the next section, where temporal features are successfully integrated into a classification algorithm.

2.5 CNN + LSTM implementation

The current section presents a solution to the problem of flow-regime classification, by training a network to classify image frames of two-phase flow within a set of *flow-image* classes. The feature sets extracted from each image frame by the frame classifier are used within a sequence of feature sets to classify a set of contiguous video frames within a more complex *flow regime* class. Code for the work presented here can be found by following the Github link in the appendix.

A *Long Short-Term Memory (LSTM)* network [141] is used to classify these sequences of image features. The present work expands this architecture to predict the average vapour quality value for a set of images by using a regression output layer rather than a classification one.

The results of this implementation will confirm that the use of image sequences to account for temporal flow characteristics is useful in image-based two-phase flow studies. Additionally, by utilizing image features extracted by a *CNN* network for three distinct tasks (for frame classification, video classification,

and regression), this method is shown to be a viable alternative to manual image feature extraction in analyzing multi-phase flow.

2.5.1 Architecture

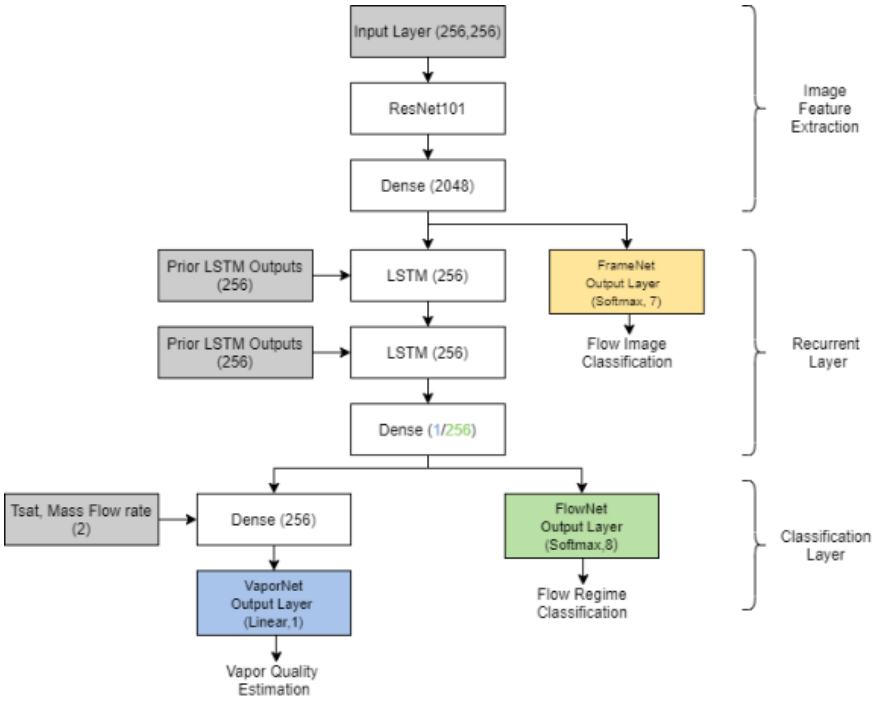

Figure 3.8- High-level network architecture; with paths for FrameNet (yellow), VapourNet (blue), and FlowNet (green). The number of nodes used for each component are shown in brackets.

Proposed Network Architecture

A key aspect of the architecture presented here is its use of separate classes for *image frames* and *frame sequences*. It was observed that images with highly similar features might occur across multiple flow regimes. When training a network that classifies frames based upon flow regime classes, the inter-class confusion is high because of these similarities. The importance of temporal information in flow regime classification has also been observed. Combining these two observations, a novel method to achieve flow regime classification was developed, where a sequence of frame features is used to predict flow regime. Similar network architectures have been implemented in other tasks related to image sequence analysis [145][147] [168].

The network architecture of the flow regime classifier (*FlowNet*) as described in *Figure 3.8* is made up of three components, with the data path for the network flowing downwards. The input to the system is an image from a sequence of frames, and the output is a flow regime class prediction. A vapour quality regression network (*VapourNet*) is implemented by adjusting the classification layers of *FlowNet.*

The first component of the system is the image feature extraction component. This operation is performed by the frame classifier (*FrameNet*). Each image in the sequence is passed through *FrameNet* to extract image features for further processing. This component of the network transforms a sequence of image frames into a sequence of image features, as seen in *Figure 3.9*. This feature sequence is used as input to the recurrent layer, which is made up of *LSTM* units. These *LSTM* units are utilized to capture temporal information across the feature sequence. The data at the output of the recurrent layer is then passed into

a set of dense layers making up the classification layer, which in turn outputs a final classification (for *FlowNet*) or regression value (for *VapourNet*).

Image Feature Extraction

The architecture selected for the frame classifier was that of *ResNet101* [122]. This architecture was chosen for its state-of-the-art classification accuracy. *ResNet101* was modified to have an output layer of seven nodes, to correspond to the frame classes for which it is trained to classify. *FrameNet* is trained to classify frame classes and is used in *FlowNet* and *VapourNet* as a feature extractor. These features are obtained as the 2048 inputs to the dense output layer of *FrameNet*. The features represent a reduction in the image data to a set of sparse features useful for flow-image classification. These features are fed directly into a dense layer for classification within their *FrameNet* classes or passed into the recurrent layer to be analyzed within a sequence of feature sets.

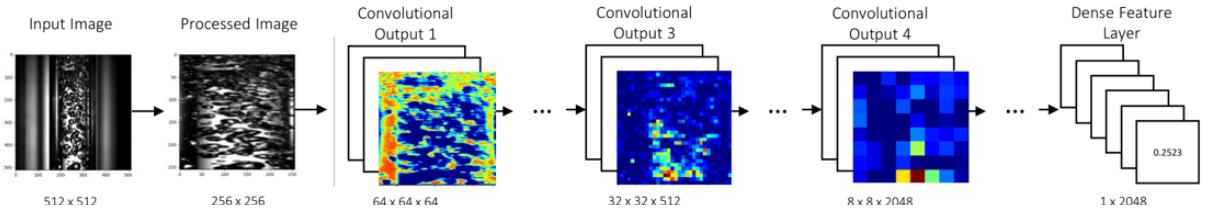

Figure 3.9- Image feature reduction; FrameNet passes an input image through a series of convolutional layers to identify the reduced image features, in the form of a set of single values, useful for frame classification.

Recurrent Layer

The analysis of the feature sequence, with each element of the sequence being made up of 2048 features corresponding to an input frame, is performed by *LSTM* units [141]. *LSTMs* are a form *of Recurrent Neural Network* (*RNN*) [131]. *RNNs* can make predictions based upon past contextual information, allowing them to interpret temporal information present in a sequence. This layer makes use of two stacked *LSTM* networks to form a deep *LSTM*, with this structure allowing for greater levels of abstraction in terms of the temporal input information to the *LSTM* [166]. The deep *LSTM* depth of two was determined through optimization by hyperparameter search. For *FlowNet*, the sequence length is 20 frames, while for *VapourNet*, this sequence is 50 frames long. The *LSTM* units have 256 hidden states, a value which, along with the sequence lengths, was determined through a hyperparameter search. The final *LSTM* unit in the second layer of the deep *LSTM* network is used to output prediction data. This is informed by past information, derived from the outputs of the previous *LSTM* units, which are used to analyse earlier frames. Although bidirectional *LSTMs* have been shown to produce better results than unidirectional *LSTMs* for similar problems [145] [168], a unidirectional design allows for the possibility of real-time sequence analysis.

Classification Layer

The features extracted in the previous layers are processed by a set of dense layers to make a final prediction. *FlowNet* uses one dense layer made up of 256 nodes, which takes the output from the *LSTM* network at its input and outputs to eight nodes with *soft-max* activation functions (one node for each flow regime class). The *soft-max* function was selected because it is an appropriate output function to use with the *Cross-Entropy* loss function used for classification. *VapourNet* makes use of two dense layers, with one linear output node for the final vapour quality prediction. The first of *VapourNet's* dense layers contains one node, which takes in the 256 hidden states from the recurrent layer. The mass flow rate and

saturation temperature (which relate to the pressure of the system) used in generating the sequence of frames are passed into the second dense layer, containing 256 units, along with the output from the single node. The second dense layer connects to a single node, with a linear activation function to output the final vapour quality prediction. The number of units and depth of the classification layer were determined through hyperparameter search.

2.5.2 Class Definitions and Data Preparation

The flow regime classes used to classify a sequence of frames were those found in the literature for vertical up-flow [11], and defined above. The classes used to define individual frames of the flow can be determined through the analysis of an image set which includes frames from video clips of each flow regime. From this analysis, distinct frame classes can be described as follows:

1. *Liquid/Tiny Bubbles:* A small number of tiny, discrete gas bubbles flow in a continuous liquid phase.
2. *Small Bubbles*: A few small, discrete gas bubbles flow in a continuous liquid phase.
3. *Big Bubbles:* A few small, discrete gas bubbles, with some large spherical bubbles and slug like bubbles within the fluid, flow in a continuous liquid phase.
4. *Dense Bubbles/ Taylor Bubbles:* Many small to medium sized discrete bubbles flow in a continuous liquid phase. The bubbles are distributed more consistently and densely across the image, with more than half the viewing section of the tube being taken up by bubbles. Taylor bubbles are also found in this frame class.
5. *Churn:* A mix of gas and liquid flow chaotically, with no visible bubbles.
6. *Annular:* A gas core forms from the center of the pipe. A wavy liquid film flows along the walls of the pipe, and liquid droplets are dispersed within the gas core.
7. *Mist/ Vapour:* No liquid is visible, as a continuous gas phase flows through the channel.

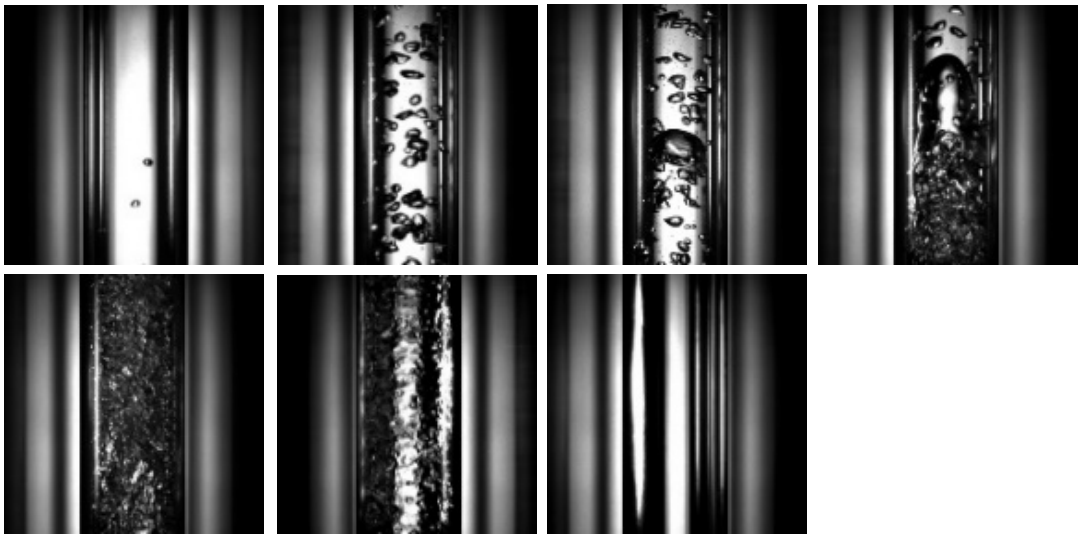

Figure 3.10-Frame classes; (Top Row) Liquid/ Tiny Bubbles, Small Bubbles, Big Bubbles, Dense Bubbles/Taylor Bubbles, (Bottom Row) Churn, Annular, Mist/Vapour

A balanced dataset made up of 39261 frames labelled with the above frame classes was generated by extracting and manually classifying a set of image frames from videos of two-phase flow generated by the test rig and used for training *FrameNet*. These images were standardized and cropped in such a way that

the edges of the flow channel are on the borders of the image. The images are then resized to 256 x 256 pixels as is required for inputs to *FrameNet* [122].

A second dataset used for the training and validating of *FlowNet* was generated by using the frame classifier to extract features for each frame in a 20-frame sequence. These 20-frame clips are extracted from flow videos generated by the test rig. Each datapoint for *FlowNet* is therefore a sequence of 20 feature sets, corresponding to the 20-frame clip which is being classified in terms of its flow regime. The label of each clip is assigned based on the flow regime label of the video from which it is partitioned.

A third dataset is required for *VapourNet*, made up of image features extracted by *FrameNet*. Clips of 50-frames were used, with each clip being labelled for training with an experimentally derived vapour quality value describing the video from which the clip was taken. Additionally*, VapourNet* requires the saturation temperature and mass flow rate of the flow at the time that the clips were taken, as two additional input channels. Both this dataset and *FlowNet's* dataset are balanced and made up of 35232 datapoints.

For the *FlowNet* and *VapourNet*, videos were down sampled from a recorded framerate of 13600 fps to framerates of 30 fps and 10 fps, respectively, with these values arising as the optimal frame rates for their specific tasks from a hyperparameter search. The lower frame rates allow for a longer video clip in the temporal dimension to be processed over a smaller number of frames, with this observation having been explored in the literature [130]. If these rates are too low, temporal aliasing will make it difficult for the *LSTM* network to extract useful temporal features between frames.

2.5.3 Training

Each of the three neural networks was trained using gradient descent methods, with the use of backpropagation over time for the *LSTM* network. Before the training of the recurrent layers, *FrameNet* was pretrained using the image dataset mentioned above. A *Cross-Entropy* loss function was used for training, with this loss function selected because of its excellent performance in classification tasks [155], and a *Stochastic Gradient Descent (SGD)* optimizer was employed for its good generalization [167]. A cross-entropy loss function was used for *FlowNet*, but an *Adam* [158] optimization function was deployed for its faster convergence. For *VapourNet*, a *Mean Square Error (MSE)* loss function was implemented and optimized using *Adam*. The detailed hyperparameters used for training can be seen in *Table 3.1*.

FrameNet was trained first, separately from the other two networks, as it is used to generate the training and testing data for the other two networks. The recurrent and classification layers of *FlowNet* and *VapourNet* were then trained using sets of this feature data generated by *FrameNet*.

Table 3.1- Training Hyper Parameters for FrameNet, FlowNet and VapourNet

Network	***FrameNet***	***FlowNet***	***VapourNet***
Optimizer	SGD	Adam	Adam
Learning-Rate	1E-3	1E-4	1E-4
Batch Size	10	256	256
Training Epochs	30	60	100
Momentum	0.9	Adaptive	adaptive

2.5.4 Results

To evaluate the performance of the three network types on their respective datasets, k-fold cross validation [169] [170] was implemented for each network. This method was implemented in lieu of the option to test the networks on multiple datasets. This method provides a better estimate of model prediction performance, and a model's sensitivity to variations in training data. *k* was chosen to be 5, as 5-fold cross validation has shown to offer an acceptable balance between bias and variance in its results [170]. By performing 5-fold cross validation, five unique sets of model parameters are produced for each of the three network types, with the performance of the networks being evaluated based on the results across their models. Network specific test sets, which were not used for the training of the models, are used to evaluate the performance of each model. Testing the models on unseen data ensures that the results reflect how the model might perform in practical application, and by using the same test set across the models of the same network type, an objective comparison can be made. Each of the test sets for the three respective models was made up of 26200 datapoints.

The results from the 5-fold cross validation of each network, using the test sets, can be seen in *Table 3.2,* with the best results for each network seen in bold. While the columns describing *FlowNet* and *FrameNet* show the accuracy of the classifiers (the percentage of classifications which were correct out of the test batch), the fourth column, describing *VapourNet,* makes use of the *Root Mean Square Error (RMSE)*. This is the square root of the MSE loss produced by *VapourNet* on the test set, and it is a measure of the average error made by the model per vapour quality prediction. The confusion matrices shown in *Figure 3.11* were generated using the highest performing model for each network.

Table 3.2- 5-fold test results for FrameNet, FlowNet and VapourNet

Fold	***FrameNet***	***FlowNet***	***VapourNet***
	Accuracy [%]	Accuracy [%]	RMSE in Vapour Quality prediction
Fold-1	**93.0**	91.5	4.8E-2
Fold-2	92.5	**95.4**	5.2E-2
Fold-3	91.0	92.5	**4.4E-2**
Fold-4	90.6	87.4	6.1E-2
Fold-5	92.3	91.8	6.4E-2
Mean	91.9	91.7	5.5E-2
Standard Deviation	0.9	2.6	0.8E-2

The left and right matrices, found in *Figure 3.11,* show the normalized confusion matrices for *FrameNet* and *FlowNet* respectively. Confusion matrices reveal how a classification model performs across its classes, revealing any prediction biases a model might have. *Figure 3.12* shows a boxplot for the set of predictions made for each vapour quality value being tested. These datapoints were generated across a range of mass flow rates and pressures, to ensure that this representation of VapourNet's performance would not be biased for a limited set of experimental conditions. Because each vapour quality being tested is represented by a video with a respective average vapour quality, there are multiple vapour quality

predictions made for each video, by using clips from the larger video for predictions. Each set of predictions is represented by a boxplot in *Figure 3.12,* with a tighter distribution of predictions representing more confident predictions for that vapour quality.

2.5.5 Discussion

Table 3.2 shows that the accuracy of *FrameNet* is high across its models. Although the *FrameNet* prediction accuracies are mostly consistent across classes, there are certain frame classes that produce a disproportionate number of classification errors, with these misclassifications occurring mainly between neighboring classes (in terms of the vapour qualities which produces them), as is seen in *Figure 3.11*. Most of these incorrect classifications occur within the *Small Bubbles* and *Big Bubbles* classes. This is likely due to the similarity between the image features found in these classes, due to the similarity between the bubbles seen in these classes: both classes have small, circular bubbles with only minor deformations in their shapes. A similar result, and subsequent explanation, can be seen for the *Big Bubbles* class, which incorrectly classifies frames as *Small Bubbles*. This issue might also extend to the classification of a sequence of frames, potentially causing confusion between the *Churn* and *Annular FlowNet* classes. Another possible cause for the above confusion is the introduction of noise to the *FrameNet* data labels, from the manual labelling of this data producing incorrectly labelled training and test data.

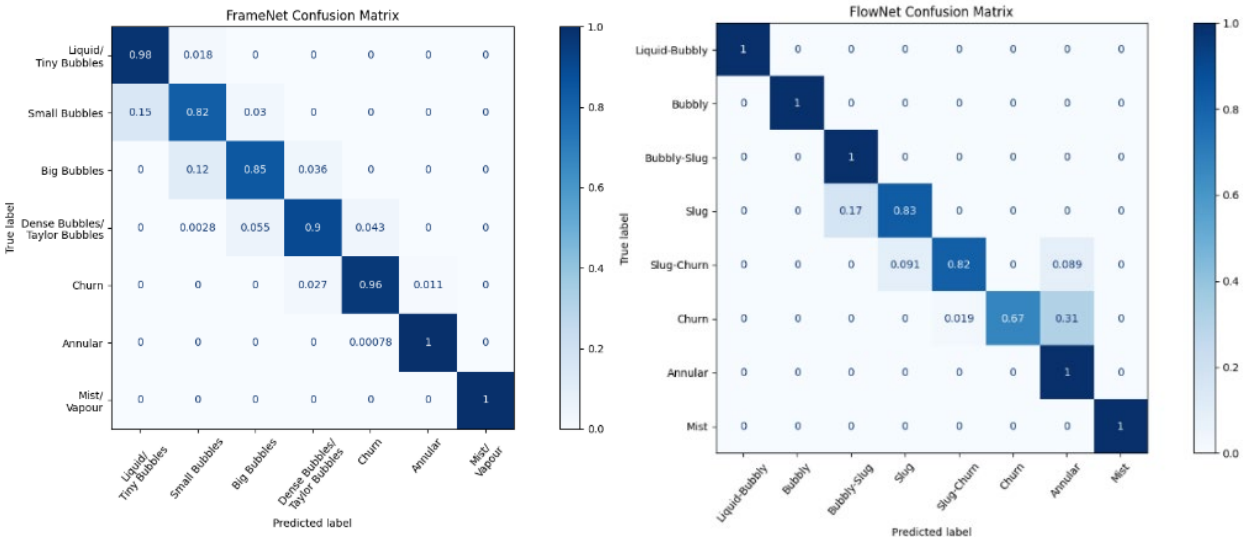

Figure 3.11- Normalized Confusion matrices of FrameNet (Left) and FlowNet(Right)

As seen in *Table 3.2*, both *VapourNet* and *FlowNet* have moderate standard deviation values when compared to *FrameNet* in their cross-validation results, which indicates that the performances of these networks are more sensitive to the data used for training and that the networks may not generalize as well to a real-world test. This could be from the increased model complexity introduced by the recurrent network layers. Another explanation is that the sequence data used to train these networks was comprised of labelled videos which were truncated and partitioned into sets of clips which share a label with their overall video. It was observed that varying flow regimes arise among clips from the same video, yet all clips share the label of the overall video. This results in the incorrect labelling of some clips. This leads to the network overfitting to incorrectly labelled clips, producing the variance in results. This also explains the incorrect *FlowNet* predictions between consecutive flow regimes seen in *Figure 3.11*, because these instances would represent clips of the overall video where the flow regime transitions from the

average regime of the video to a transitional regime in a class preceding or proceeding the average regime class. The way to interpret the clip predictions would then be that when accumulated, they express the flow regime makeup of the overall video. In all test cases, the average flow regime predicted across the set of clips in a given video was correct for that video.

Despite this, *FlowNet* achieves a mean accuracy of 92%, with almost all incorrect classifications falling within consecutively occurring flow regimes to their correct regime. These results give good confidence in predictions for the current application. The model architecture therefore has the capacity to be trained successfully for the current task, but there is a tendency to overfit to the training data, producing the high variance in results as mentioned above.

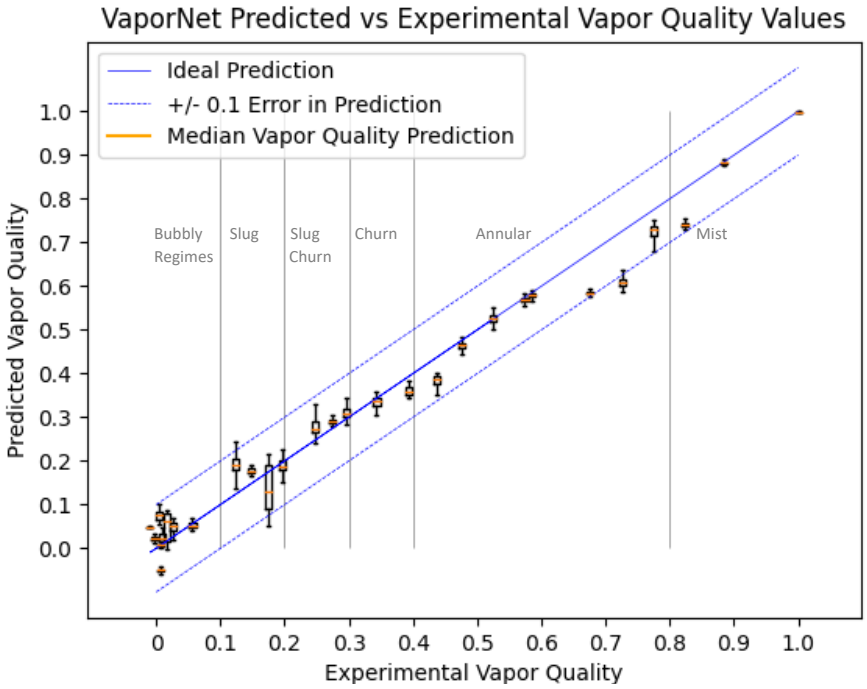

Figure 3.12- VapourNet predicted values vs Experimental Vapour quality values, with rough classifications of the flow regimes found within specific vapour quality ranges

Figure 3.12 shows a clear correlation between vapour quality predictions and true vapour quality labels, with generally accurate, high confidence predictions being made with the following exceptions: Between the label ranges of 0.1 and 0.2, the network shows a wider range of predictions, and therefore a lower confidence per label. This range of vapour qualities produces *slug* flow, with videos of this flow regime transitioning between clips of *liquid-bubbly*, to *bubbly-slug*, to *slug*, to *slug-churn*. Because this vapour quality range produces videos which change significantly over time, it is not surprising that different average vapour qualities would be predicted across the set of clips for a such a video. Nevertheless, the mean vapour quality predictions for these videos are still reasonable. There is a distinct drop in accuracy from the ranges 0.6 to 0.9, with mean prediction accuracy dropping as low as 12% error. This region represents *Annular* and *Mist* flow. The networks' tendency to under-predict vapour quality in this region could be a result of inaccuracies in the vapour quality labels applied to each video. There is a high degree of uncertainty for the average vapour quality label assigned to each video, and as such, it is possible that *VapourNet's* apparent sensitivity to input data could be reflecting a response to receiving training data labels that have a high degree of uncertainty. It is also possible that the image features extracted by

FrameNet for flow image classification do not contain the information content required to accurately predict the vapour quality of high-vapour-quality annular flow. This explains the seemingly uniform classification of these videos, seen in *Figure 3.12*, as the image data required to calculate their vapour quality to a more accurate degree is lost by *FrameNet.*

Table 3.2 reveals that *VapourNet* can predict vapour quality with a mean RMSE of 5% in terms of possible vapour quality values across the partitions of its training data. *VapourNet* can therefore be applied to utilize a sequence of image features and make correlations between a set of input images and their associated average vapour quality for applications where the accuracy required falls within this error

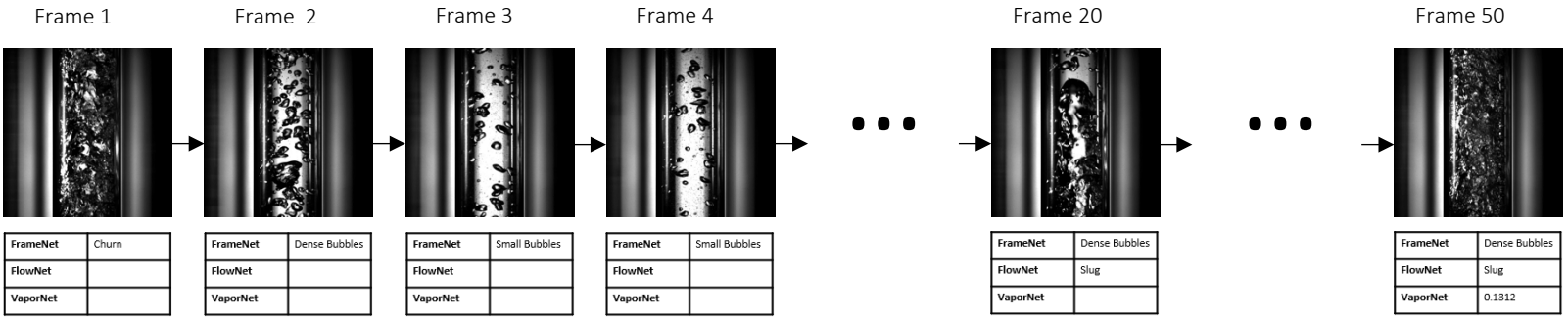

Figure 3.13-Example of classification and regression results

range.

Figure 3.13 highlights that classifying flow regime by sequence is an appropriate design decision: *FrameNet* identifies different image classes within the same flow regime, whereas *FlowNet* analyses the sequence of frames to make a correct classification on the entire set. This shows that temporal information is useful when classifying flow regime from image data. Additionally, when observing this sequence of frames, the deficiencies of *FrameNet* as a flow regime classifier are highlighted; *FrameNet* classifies frames 1,2 and 3 within different classes, yet they all belong to the same flow regime class. This shows that despite *FrameNet* achieving higher accuracy than *FlowNet,* it is fundamentally unsuitable for flow-regime classification because it does not account for temporal information.

2.5.6 Uncertainty in predictions

To measuring the quality of a prediction output from *FrameNet* and *FlowNet*, the probability of miss-classification by class is used. The left and right histograms depicted in *Figure 3.14* show the probabilities of misclassification associated with each possible output prediction produced by FrameNet and FlowNet respectively. While the overall probability of misclassification by each of these networks is 4.6% for FlowNet and 7% for FrameNet, the probabilities shown in the figure below can be used by a user to interpret the quality of an output prediction produced by either network. These values were derived by calculating what percent of predictions within each class are incorrect, with this calculation relating to the vertical columns of the confusion matrices seen in *Figure 3.11*.

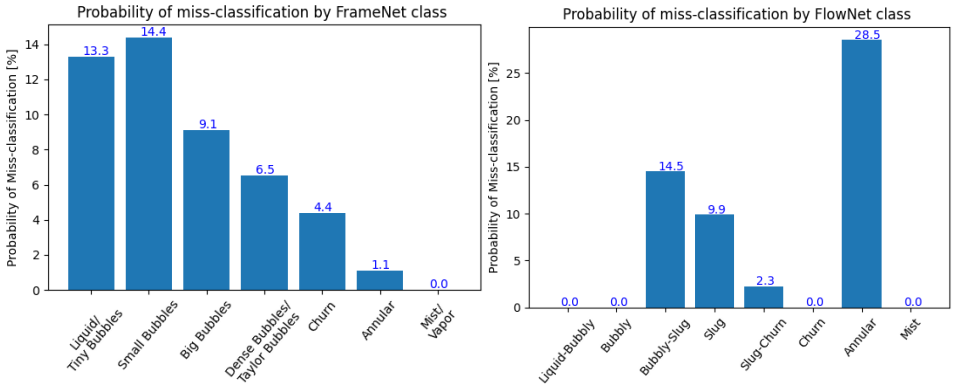

Figure 3.14- Probability of misclassification by class; (Left) FrameNet, (Right) FlowNet

In terms of VapourNet, the lowest RMSE value shown in *Table 3.2* is 4.4E-2, meaning that on average, a predicted output falls within ±4.4E-2 of its true value. From this, a range of possible values of ±4.4E-2 is included in each prediction made by the network. In addition to this, *Figure 3.12* shows that almost all vapour quality predictions are within ±1.0E-1 of their true value. This range represents a conservative worst case set of possible output values for a given prediction, which might be considered for tasks related to safety monitoring.

2.5.7 Practical Applications

The results observed when testing *FrameNet* indicate that *ResNet101* is an appropriate architecture for classifying flow images. *FrameNet* can be used on its own as a tool to identify the types of images inherent in different flow regime videos. Additionally, its good classification accuracy for *Churn* and *Annular* flow allows this network to be used to detect the transition point between those regimes in a sequence of frames, a task which is often impossible for a human observer due to the transient nature of this event.

FlowNet can be used to classify video clips of two-phase flow into the correct flow regime across the entire range of possible vapour qualities and flow regimes. The automation of this task will reduce the need for manual data labelling by a user studying the flow. *FlowNet* can also be used to identify transition points between flow regimes, a useful feature when developing a flow regime map. *FlowNet* can also be used to identify the flow regime composition of a video of two-phase flow. *FlowNet's* performance accuracy can be increased when the average prediction across a set of clips for a given video is taken to classify that video.

VapourNet makes reasonable predictions of the average vapour quality of a given video clip. It is a useful tool for a user who might require an automated estimate of vapour quality, when manual calculation is not feasible. The results produced by *VapourNet* also serve as a reasonable first estimate, which could be used by researchers to target data within specific vapour quality ranges for further study. *VapourNet* could also be implemented as a sensor to provide feedback within a control system, although this would most likely require an increase in the networks prediction accuracy.

2.5.8 Moving towards a real-time implementation

For *VapourNet* to be utilized within a control feedback loop (for tasks such as the control of the refrigerant feed into a system, so as to maintain a constant vapour quality in the return line), this model must be deployed in a way that allows for the interpolation of vapour quality values with as little latency as possible. To this end, the current section details a prototype real-time implementation of *VapourNet*. A

similar prototype is detailed for *FlowNet*, with the real-time implementation of that model being useful for the live, automatic detection of flow-regime transitions.

Hardware

For the output of these models to be utilized in a feedback control loop in the same way as sensor data, the models must be deployed for long periods of time. Along with this requirement, at the level of a laboratory prototype, such an implementation must be cost effective, powerful enough to generate data in real time, and must allow for easy installation and deployment.

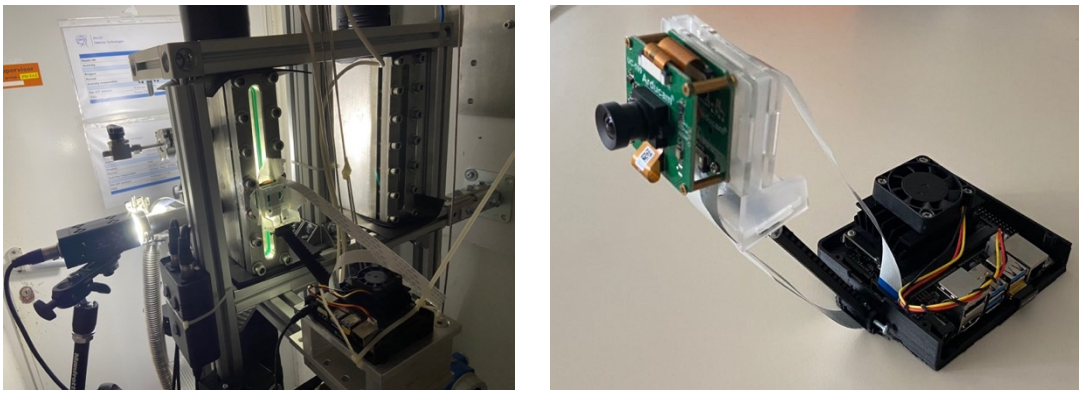

Figure 3.15- Laboratory Prototype; (Left) The prototype deployed to collect data from the test environment, (Right) The prototype.

To evaluate the use of these models in a feedback control setting, a laboratory prototype was developed using a Jetson Nano micro-computer with an Arducam OV9281 camera as shown in *Figure 3.15*. The Jetson Nano has a 128 core GPU and is able achieve acceptable frame rates to demonstrate the principle at low cost.

Model Revisions for prototype

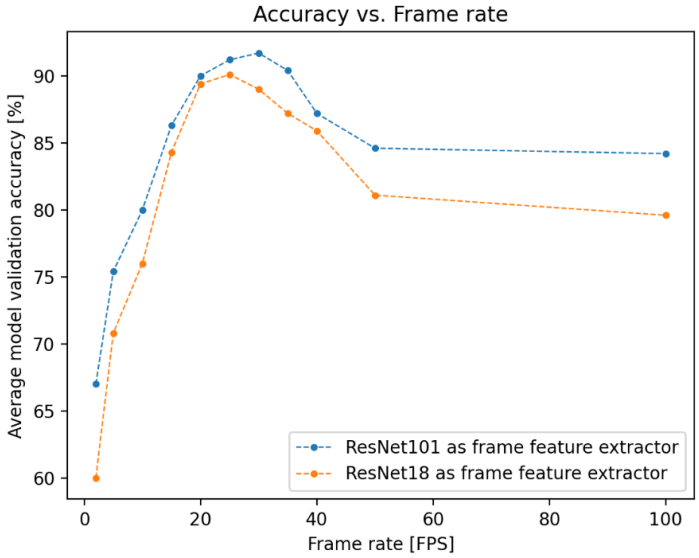

Figure 3.16- Accuracy vs. Frame rate for FlowNet; Depicted here are the results of the hyperparameter searches run during the training of the networks used for offline (using ResNet101) and online (using ResNet18) testing. These results are generated with offline data from the FastCam, and as such, performance is expected to degrade in online testing, where data generated by the ArduCam is used.

Through performance testing, it was found that the Jetson Nano can process individual frames using *FrameNet* (which has ResNet101 architecture) at a stable frame rate of just over 10 FPS. While this interpolation speed is fast enough for *VapourNet*, *FlowNet* was trained using data captured at a higher frame rate (30 FPS).

It was found that the Jetson Nano can process frames using ResNet18 [122] (A version of ResNet with fewer parameters and therefore a lower memory requirement) at 20 FPS. As such, *FlowNet* and *FrameNet* were retrained using this new architecture, with results shown in *Figure 3.16.* From these results, the version of *FlowNet* deployed on the prototype makes use of ResNet18 architecture and data captured at 20 FPS. The improvement of this lighter model at 20 FPS over the results seen when *FlowNet* is retrained with data captured at 10 FPS, while still using ResNet101 architecture, indicates that with fewer model parameters, useful image features for flow regime classification can still be extracted by the network. This improvement also confirms that frame rate is a crucial hyper parameter in designing a network for flow regime classification.

Results and Discussion

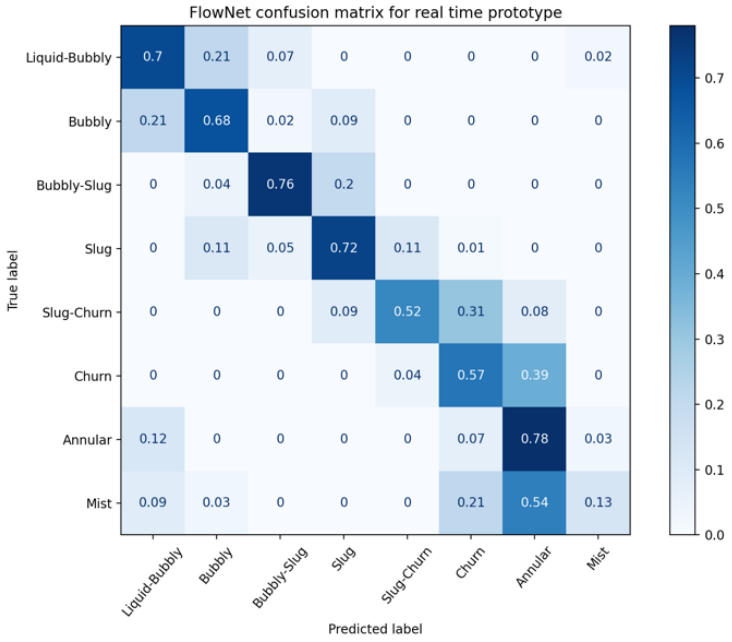

Figure 3.17- Normalized confusion matrix for FlowNet, generated from real-time data, using the laboratory prototype

When testing *FlowNet* on the prototype architecture, with data collected in real time at 20 FPS, a flow regime classification accuracy of 61% was achieved. *Figure 3.17* reveals the confusion matrix obtained in the testing of this model. While a drop in accuracy can be seen for each flow regime class when compared to the off-line results seen in *Figure 3.11*, the most significant decrease in accuracy occurs for the mist flow class. Because images of mist flow are very similar, it is likely that *FlowNet* made use of image features highly specific to the lighting conditions found in the training data when classifying this regime. For the data collected using the real-time prototype, there are inherent differences in the input images, when compared to the training data, due to the altered camera positioning and different camera parameters (such as aperture, white balance, shutter speed, etc.) . This leads to a change in the features found within an image, and thus affects the performance of the classifier. This effect decreases the performance of all

classes, but it affects mist flow, which contains more consistent features across images, the most negatively. Another cause for the decrease in performance across classes is the difference in image resolution between the training data and testing data. While the Arducam can capture images with a resolution of 512 x 512 at the desired frame rates, the lens of the camera does not offer the same zoom capabilities as the Fastcam, and as such, the images of the tube section passed into the network must be rescaled from a lower resolution. This leads to a distortion and potential loss of feature information, effecting the performance of the network across classes.

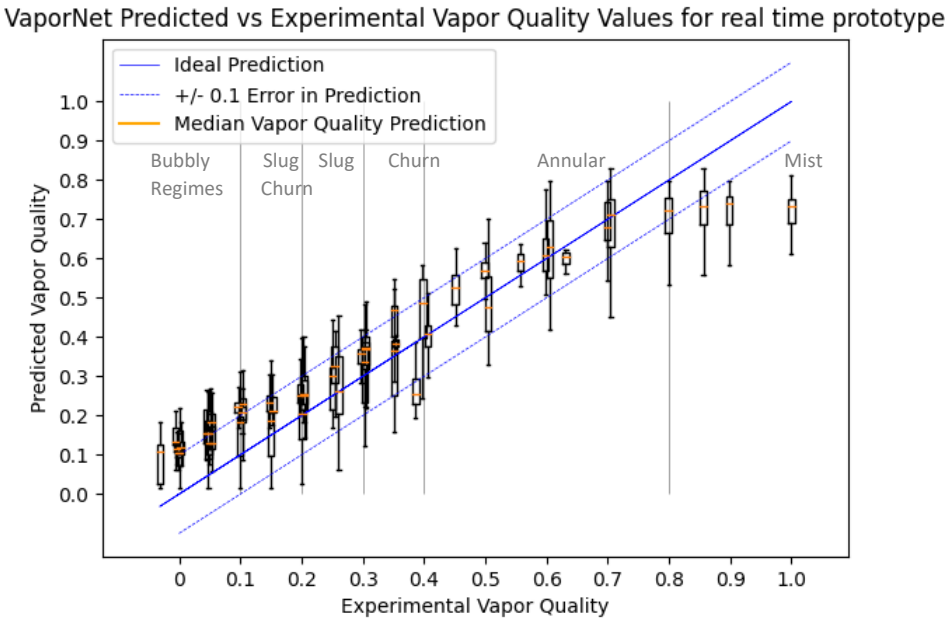

Figure 3.18- VapourNet predicted values vs Experimental Vapour quality values for the real-time prototype, using real-time data, with rough classifications of the flow regimes found within specific vapour quality ranges

VapourNet achieves an RMSE of 0.1 when tested in real-time. The comparison between *Figure 3.12* and *Figure 3.18* reveals that a decrease in performance occurs across vapour qualities for the new implementation, most likely for the same reasons as those mentioned for *FlowNet*. Mist flow stands out once again for suffering the greatest decrease in performance when testing on the new data, most likely for the same reasons as those stated above.

Although both models suffer a decrease in performance from this new implementation, when compared to the results seen in an offline implementation, these decreases are most likely the result of the differences between training and testing data, which were generated using unique camera setups, and not a result of the change to real-time data capturing and processing. The fact that both models still follow a desirable trend for their respective tasks, with *FlowNet* mostly classifying video clips into the correct class, with the majority of incorrect predictions being in nearby classes to the target class, and with *VapourNet* having predictions which increase proportionately with increasing vapour quality, despite the differences of images between datasets, strengthens the idea that these models could be improved through retraining to achieve similar performances to those seen in their off-line implementations. The data shown here therefore represents an initial set of results for the real-time implementations of

FlowNet and *VapourNet*, as these results would surely be improved if the models had been trained on data from the camera environment used at test-time.

2.5.9 Conclusion

FrameNet was successfully trained to classify individual images of two-phase CO_2 flow, but it was found that individual frame classification can be misleading when considering flow with transient properties. *VapourNet* and *FlowNet* address this issue by incorporating temporal information through the addition of *LSTM* networks to their architectures, while making use of spatial information provided by *FrameNet*. The results produced by these networks, and the related qualitative arguments presented in this section encourage the inclusion of temporal information in further image-based studies of multiphase flow. Additionally, the image feature detectors learnt by the convolutional layers of *FrameNet* were shown to be useful for all three network types. Their successful application in three unique tasks confirms that these feature detectors are robust and produce a well generalized reduction of the image information content.

Encouraging results were also seen in the prototype real-time implementations of these models, with the possibility to improve these results through the retraining of the models using data more similar to that seen at test-time.

2.6 Discussion and Conclusion

Table 3.3 allows for the comparison of the various methods which were developed in the attempt to classify flow regime using images of two-phase flow. This table makes it clear that the incorporation of temporal information is highly beneficial in this task, with performance across the models increasing with the complexity by which the models incorporate temporal information.

An accuracy of around 77% is the best that these models achieve without the incorporation of temporal information describing a sequence of frames. This is because three of the eight classes (making up around 40% of all test data), bubbly-slug, slug, and slug-churn, are consistently poorly classified across models. The model which breaks this threshold, *FlowNet*, is the only model to perform well on these three classes. *FlowNet's* ability to accurately classify these highly transient flow regimes leads to the conclusion that the classification of these regimes, using computer vision techniques, requires the successful incorporation of temporal features within a given model, as the qualitative definitions, used to define these classes, account for temporal observations describing the regimes, rather than just spatial ones.

Table 3.3-Flow Regime classification accuracies achieved across architectures

Model	Classification Accuracy [%]
Single frame model	68
Late Feature Pooling (using Single Frame Model)	71
Convolutional Feature Pooling (using FrameNet)	77
Two-Stream (using Averaging Fusion)	77
CNN + LSTM (FlowNet)	95

The successfully developed tools found in this chapter include *FlowNet,* a flow regime classifier for vertical two-phase up flow, *FrameNet*, an image-based classifier which can be used to detect transitions between visually complex flow regimes (such as churn, annular, and mist flows) which are imperceivable to humans on a frame-by-frame basis, and *VapourNet*, which is used to extract the average vapour quality found across a set of sequenced image frames.

Conclusion

This work has documented the successful development and performance of a set of tools for the analysis of vertical up two-phase CO_2 flow. These tools make use of techniques common in the fields of image processing and computer vision, and include an algorithm for the extraction of void fraction from a given input image, an image based detector of flow regime transitions for complex flow regimes, a classifier of flow regimes across the spectrum of possible vapour qualities, and a model which can estimate the average vapour quality from a set of sequenced input images.

The work done in *Part II*, which makes use of traditional image processing techniques, reveals many of the advantages and disadvantages of these methods. The utilization of manually extracted, low-level image features by these techniques, allows for the transparent analysis of a system which makes use of them, and allows for the user to utilize these insights in the further development of the system. This gives a level of confidence in the results of such a system, which deep learning-based computer vision techniques cannot provide.

As stated, in the development of these techniques, the transparency of the system allowed for effective fine tuning and debugging. In contrast, the problems encountered when testing different network architectures for flow regime classification, seen in *Part III*, did not have obvious solutions for improving the performance of these architectures. This led to the implementation, and subsequent abandonment, of a set of network architectures, without the realistic possibility of fine tuning one of these architectures to achieve the desired results. This is due to the black box nature of deep learning techniques, which leads to systems which are simple to implement, but challenging to understand.

This simplicity of implementation allows for a far lower level of expertise required to solve a given problem with deep learning-based computer vision techniques. The research and work done in *Part II* is more extensive than that found in *Part III*, yet the work done in the former is only applicable to a narrow range of vapour qualities, while the work done in the latter is relevant for the entire range of vapour qualities. This contrasts the understanding required to solve problems using traditional image processing techniques with the relative ease with which problems can be solved using deep learning-based computer vision techniques. This is a testament to the performance and flexibility of these deep learning architectures.

With techniques from both these fields having unique advantages, any selection between them should be task specific. With that said, in recent years, more and more image-based problems are being solved using the deep learning approach [120] [121], due to its ease of application, the constantly decreasing barrier to access hardware powerful enough to run such systems, and the constantly improving results produced by these methods [122] [123] [124].

Appendix

Github link: https://github.com/shaikadish/verticalFlowStudy

References

[1] EP-DT Detector Technologies, "Detector Cooling Services," 2021. [Online]. Available: https://ep-dep-dt.web.cern.ch/detector-cooling-service. [Accessed 7 July 2021].

[2] CERN, "The Atlas Experiment," 2021. [Online]. Available: https://atlas.cern/about. [Accessed 16 July 2021].

[3] P. Sassi, J. Pallarès and Y. Stiriba, "Visualization and measurement of two-phase flows in horizontal pipelines," *Experimental and Computational Multiphase Flow,* vol. 2, no. 1, pp. 41-51, 2020.

[4] B. A. Shannak, "Frictional pressure drop of gas liquid two-phase flow in pipes," *Nuclear Engineering and Design,* vol. 328, no. 12, pp. 3277-3284, 2008.

[5] R. Lockhart and R. Martinelli, "Proposed correlation of data for isothermal," *Chem. Eng. Prog,* vol. 45, no. 1, pp. 39-45, 1949.

[6] D. Chisholm, "A theoretical basis for the Lockhart–Martinelli correlation for two-phase flow," *International Journal of Heat Mass Transfer,* vol. 10, pp. 1767-1778, 1967.

[7] A. E. Dukler, M. Wicks and R. G. Cleveland, "Frictional pressure drop in two-phase flow: B. An approach through similarity analysis.," *A.I.Ch.E. Journal,* vol. 10, no. 1, pp. 44-51, 1964.

[8] J. Thome and A. Cioncolini, "Two-Phase Pressure Drop," in *Encyclopedia of Two-Phase Heat Transfer and Flow*, World Scientific Publishing, 2015, pp. 143-176.

[9] F. Holland and R. Bragg, "Gas–liquid two-phase flow," in *Fluid Flow for Chemical Engineers (Second Edition)*, Butterworth-Heinemann, 1995, pp. 219-267.

[10] A. J. Ghajar and S. M. Bhagwat, "Flow Patterns, Void Fraction and Pressure Drop in Gas-Liquid Two Phase Flow at Different Pipe Orientations," in *Frontiers and Progress in Multiphase Flow 1*, Springer International Publishing, 2014, pp. 157-212.

[11] Y. Taitel, D. Bornea and A. E. Dukler, "Modelling flow pattern transitions for steady upward gas-liquid flow in vertical tubes," *AIChE Journal,* vol. 26, no. 3, pp. 345-354, 1980.

[12] J. Mandhane, G. Gregory and K. Aziz, "A flow pattern map for gas—liquid flow in horizontal pipes," *International Journal of Multiphase Flow,* vol. 1, no. 4, pp. 537-553, 1974.

[13] L. Cheng, G. Ribatski, J. M. Quibén and John R. Thome, "New prediction methods for CO2 evapouration inside tubes: Part I – A two-phase flow pattern map and a flow pattern based

phenomenological model for two-phase flow frictional pressure drops," *International Journal of Heat and Mass Transfer,* vol. 51, no. 1-2, pp. 111-124, 2008.

[14] D. Schmid, B. Verlaat, P. Petagna, R. Revellin and J. Schiffmann, "Flow pattern observations and flow pattern map for adiabatic two-phase flow of carbon dioxide in vertical upward and downward direction," *Experimental Thermal and Fluid Science,* vol. 131, no. 1, 2022.

[15] R. Clift, J. R. Grace and M. E. Weber, Bubbles, Drops and Particles, New York: Academic Press, Inc., 1978.

[16] M. Kabir and J. Xu, "Experimental Approaches to Measurement of Vapour Quality of Two-Phase Flow Boiling," in *Heat Transfer - Design, Experimentation and Applications*, IntechOpen, 2020.

[17] T. L. Bergman, F. P. Incropera, D. P. DeWitt and A. S. Lavine, "Boiling and Condensation," in *Fundamentals of Heat and Mass Transfer*, John Wiley & Sons, 2011, pp. 654-706.

[18] P. Godbole, C. Tang and A. Ghajar, "Comparison of Void Fraction Correlations for Different Flow Patterns in Upward Vertical Two-Phase Flow," *Heat Transfer Engineering,* vol. 32, pp. 843-860, 2011.

[19] M. Pietrzak and M. Płaczek, "Void fraction predictive methods in two-phase flow across a small diameter channel," *International Journal of Multiphase Flow,* vol. 121, 2019.

[20] Y. Xu and X. Fang, "Correlations of void fraction for two-phase refrigerant flow in pipes," *Applied Thermal Engineering,* vol. 64, no. 1-2, pp. 242-251, 2014.

[21] D. Chishom, "Pressure gradients due to friction during the flow of evapourating two-phase mixtures in smooth tubes and channels," *International Journal of Heat and Mass Transfer,* vol. 16, no. 2, pp. 347-358, 1973.

[22] A. Premoli, D. Francesco and A. Prima, "An Empiri-cal Correlation for Evaluating Two-Phase Mixture Density Under Adiabatic Conditions," in *European Two-Phase Flow Group Meeting*, Milan, 1970.

[23] G. A. Hughmark, "Holdup in gas liquid flow," *Chemical Engineering Progress,* vol. 58, pp. 62-65, 1962.

[24] A. A. Armand, "The Resistance During the Movement of a Two-Phase System in Horizontal Pipes," *Izvestiya Vsesoyuznogo Teplotekhnicheskogo Instituta,* vol. 1, pp. 16-23, 1946.

[25] W. A. Massena, "Steam-Water Pressure Drop and Critical Discharge Flow—A Digital Computer Program," Hanford Atomic Products Operation, Richland, WA, 1960.

[26] S. Rouhani and E. Axelsson, "Calculation of void volume fraction in the subcooled and quality boiling regions," *International Journal of Heat and Mass Transfer,* vol. 13, no. 2, pp. 383-393, 1970.

[27] S. Morooka, T. Ishizuka, M. Iizuka and Y. K. , “Experimental study on void fraction in a simulated BWR fuel assembly (evaluation of cross-sectional averaged void fraction),” *Nuclear Engineering Design,* vol. 114, pp. 91-98, 1989.

[28] S. Bhagwat and A. Ghajar, “Flow Pattern and Pipe Orientation Independent Semi-Empirical Void Fraction Correlation for a Gas-Liquid Two Phase Flow Based on the Concept of Drift Flux Model,” in *ASME 2012 Heat Transfer Summer Conference*, 2012.

[29] D. Butterworth, “A comparison of some void-fraction relationships for co-current gas-liquid flow,” *International Journal of Multiphase Flow,* vol. 1, no. 6, pp. 845-850, 1975.

[30] S. G. Bankoff, “A Variable Density Single-Fluid Model for Two-Phase Flow With Particular Reference to Steam-Water Flow,” *ASME. J. Heat Transfer,* vol. 82, no. 4, pp. 265-272, 1960.

[31] M. A. Woldesemayat and A. J. Ghajar, “Comparison of voidfraction correlations for different flow patterns in horizontal and upward inclined pipes,” *International Journal of Multiphase Flow,* vol. 33, pp. 347-370, 2007.

[32] D. Chisholm and A. Laird, “Two phase flow in rough tubes,” *Trans. ASME,* vol. 80, pp. 276-286, 1958.

[33] D. Graham, D. Yashar, M. Wilson, H. Kopke, J. Chato and T. Newell, “An investigation of refrigerant void fraction in horizontal, micro fin tubes,” *HVAC&R Res.,* vol. 7, pp. 67-82, 2001.

[34] G. Wallis, One Dimensional Two-Phase Flow, New York: McGraw-Hill Inc., 1969.

[35] Y. Fu and Y. Liu, “Development of a robust image processing technique for bubbly flow measurement in a narrow rectangular channel,” *International Journal of Multiphase Flow,* vol. 84, pp. 217-228, 2016.

[36] P. L. Serra, P. H. Masotti, M. S. Rocha, D. A. d. Andrade, W. M. Torres and R. N. d. Mesquita, “Two-phase flow void fraction estimation based on bubble image segmentation using Randomized Hough Transform with Neural Network (RHTN),” *Progress in Nuclear Energy,* vol. 118, 2020.

[37] A. X. Meng, G. A. Hill and A. K. Dalai, “Modified volume expansion method for measuring gas holdup,” *The Canadian Journal of Chemical Engineering,* vol. 80, no. 2, pp. 194-199, 2002.

[38] Y.-P. Ma, N.-M. Chung, B.-S. Pei, W.-K. Lin and Y.-Y. Hsu, “Two Simplified Methods to Determine Void Fractions for Two-Phase Flow,” *Nuclear Technology,* vol. 94, no. 1, pp. 124-133, 1991.

[39] H.-M. Prasser, A. Böttger and J. Zschau, “A new electrode-mesh tomograph for gas-liquid flows,” *Flow Measurement and Instrumentation,* vol. 2, no. 9, pp. 111-119, 1998.

[40] H.-M. Prasser, D. Scholz and C. Zippe, “Bubble size measurement using wire-mesh sensors,” *Flow Measurement and Instrumentation,* vol. 4, no. 12, pp. 299-312, 2001.

[41] B. Reddy Vanga, A. Zaruba, H.-M. Prasser, E. Krepper and M. Lopez de Bertodano, “Wire-mesh tomography measurements of void fraction in rectangular bubble columns,” in *Proceedings of the 2004 International Congress on Advances in Nuclear Power Plants*, 2004.

[42] A. Manera, B. Ozar, S. Paranjape, M. Ishii and H.-M. Prasser, “Comparison between wire-mesh sensors and conductive needle-probes for measurements of two-phase flow parameters,” *Nuclear Engineering and Design,* vol. 239, no. 9, pp. 1718-1724, 2009.

[43] C. Ofuchi, H. Eidt, C. Rodrigues, E. Dos Santos, P. Dos Santos, M. da Silva, F. J. Neves, P. Domingos and R. Morales, “Multiple wire-mesh sensors applied to the characterization of two-phase flow inside a cyclonic flow distribution system,” *Sensors ,* vol. 19, no. 1, 2019.

[44] P. Niu, D. Wang, P. Wei, Y. Yang, Y. Pan, S. Wang and X. Yu, “Liquid flow measurement using phase isolation and an imaging method in horizontal gas-liquid two-phase flow,” *Measurement Science and Technology,* vol. 31, no. 9, 2020.

[45] M. Rahman, T. Heidrick and B. Fleck, “A Critical Review of Advanced Experimental Techniques to Measure Two-Phase Gas/Liquid Flow,” *The Open Fuels & Energy Science Journal,* vol. 2, pp. 54-70, 2009.

[46] E. Bjork, “A simple technique for refrigerant mass measurement,” *Applied Thermal Engineering,* vol. 25, pp. 1115-1125, 2005.

[47] C. Corazza, K. Rosseel, W. Leysen, K. Gladinez, A. Marino, J. Lim and A. Aerts, “Optical fibre void fraction detection for liquid metal fast neutron reactors,” *Experimental Thermal and Fluid Science,* vol. 113, 2020.

[48] H. De Lasa, S. L. P. Lee and M. A. Bergougnou, “Bubble measurement in three-phase fluidized beds using a u-shaped optical fiber,” *The Canadian Journal of Chemical Engineering,* vol. 62, no. 2, pp. 165-169, 1984.

[49] V. Bertola, “Two-Phase Flow Measurement Techniques,” in *Modelling and Experimentation in Two-Phase Flow*, New York, Springer-Verlag Wien, 2003, pp. 281-325.

[50] W. Leung, S. Revanka, Y. Ishii and M. Ishii, “Axial development of interfacial area and void concentration profiles measured by double-sensor probe method,” *International Journal of Heat and Mass Transfer,* vol. 38, no. 3, pp. 445-453, 1995.

[51] M. Banowski, A. Patmonoaji, D. Lucas and U. Hampel, “A novel fuzzy-logic based method for determination of individual bubble velocity and size from dual-plane ultrafast X-ray tomography data of two-phase flow,” *International Journal of Multiphase Flow,* no. 96, pp. 144-160, 2017.

[52] S. Azizi, A. Yadav, Y. M. Lau, U. Hampel, S. Roy and M. Schubert, “On the experimental investigation of gas-liquid flow in bubble columns using ultrafast X-ray tomography and radioactive particle tracking,” *Chemical Engineering Science,* vol. 170, pp. 320-331, 2017.

[53] M. Banowski, D. Lucas and L. Szalinski, “A new algorithm for segmentation of ultrafast X-ray tomographed gas-liquid flows,” *International Journal of Thermal Sciences,* no. 90, pp. 311-322, 2015.

[54] S. H. Stavland, C. Sætre, B. T. Hjertaker, S. Tjugum, A. Hallanger and R. Maad, “Gas Fraction Measurements using Single and Dual Beam Gamma-Densitometry for Two Phase Gas-Liquid Pipe Flow,” in *2019 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2019.

[55] Q. Duan, J. Peng, L. Wang and Y. Yan, “Measurement of the void fraction of gas-liquid two-phase CO2 flow using laser attenuation techniques,” in *2018 IEEE International Instrumentation and Measurement Technology Conference: Discovering New Horizons in Instrumentation and Measurement*, 2018.

[56] H. Wu and Q. Duan, “Gas void fraction measurement of gas-liquid two-phase CO2 flow using laser attenuation techniqu,” *Sensors (Switzerland),* vol. 19, no. 14, 2019.

[57] T. Takamasa, K. Kondo, M. Kawase, K. S. Rezkallah and N. K. Clarke, “Measurement of interfacial configuration of bubbly flow under normal and microgravity conditions using stereo image-processing method,” *Transactions of the Japan Society of Mechanical Engineers, Part B,* vol. 63, no. 606, pp. 396-403, 1997.

[58] T. Takamasa and A. Tomiyama, “Three-dimensional gas-liquid two-phase bubbly flow in a C-shaped tube.,” in *Ninth International Topical Meeting on Nuclear Reactor Thermal Hydraulics*, San Francisco, 1999.

[59] X. Fu, P. Zhang, H. H. C. J. Huang, Y. Huang and R. Z. Wang, “3D visualization of two-phase flow in the micro-tube by a simple but effective method,” *Journal of Micromechanics and Microengineering,* vol. 19, no. 8, 2009.

[60] R. Kong, A. Rau, S. Kim, S. Bajorek, K. Tien and C. Hoxie, “A robust image analysis technique for the study of horizontal air-water plug flow,” *Experimental Thermal and Fluid Science,* vol. 102, pp. 245-260, 2019.

[61] e. a. Xiaoran Yu, “Measurement technique for solid-liquid two-phase flow using a Normal-line Hough Transform method,” in *The 6th International Symposium on Measurement Techniques for Multihphse Flows*, Okinawa, 2008.

[62] M. Honkanen, P. Saarenrinne, T. Stoor and J. Niinimäki, “Recognition of highly overlapping ellipse-like bubble images,” *Measurement Science and Technology,* vol. 16, no. 9, 2005.

[63] W.-H. Zhang, X. Jiang and Y.-M. Liu, “A method for recognizing overlapping elliptical bubbles in bubble image,” *Pattern Recognition Letters,* vol. 33, no. 12, pp. 1543-1548, 2012.

[64] A. Karn, C. Ellis, R. Arndt and J. Hong, “An integrative image measurement technique for dense bubbly flows with a wide size distribution,” *Chemical Engineering Science,* vol. 122, pp. 240-249, 2015.

[65] Y. Lau, N. Deen and J. Kuipers, “Development of an image measurement technique for size distribution in dense bubbly flows,” *Chemical Engineering Science,* vol. 94, pp. 20-29, 2013.

[66] S. Hosokawa, K. Tanaka, A. Tomiyama, Y. Maeda, S. Yamaguchi and Y. Ito, “Measurement of micro Bubbles generated by a pressurized dissolution method,” in *The 6th International Symposium of Measurement Techniques for Multiphase Flows*, Okinawa, 2008.

[67] T. Bui-Dinh and T.-S. Choi, “Bubble velocity measurements using binary image processing techniques,” in *Proc. SPIE 4115, Applications of Digital Image Processing XXIII*, 2000.

[68] F. Pla, “Recognition of Partial Circular Shapes from Segmented Contours,” *Computer Vision and Image Understanding,* vol. 63, no. 2, pp. 334-343, 1996.

[69] L. Shen, X. Song, M. Iguchi and F. Yamamoto, “A method for recognizing particles in overlapped particle images,” *Pattern Recognition Letters,* vol. 21, no. 1, pp. 21-30, 2000.

[70] Y. Fu and Y. Liu, “Development of a robust image processing technique for bubbly flow measurement in a narrow rectangular channel,” *International Journal of Multiphase Flow,* vol. 84, pp. 217-228, 2016.

[71] G. Taubin, “Estimation of Planar Curves, Surfaces, and Nonplanar Space Curves Defined by Implicit Equations with Applications to Edge and Range Image Segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 13, pp. 1115-1138, 1991.

[72] A. Fitzgibbon, M. Pilu and R. B. Fisher, “Direct least square fitting of ellipses,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 21, no. 5, pp. 476-480, 1999.

[73] F. Meyer and S. Beucher, “Morphological segmentation,” *Journal of Visual Communication and Image Representation,* vol. 1, no. 1, pp. 21-46, 1990.

[74] P. L. Serra, P. H. Masotti, M. S. Rocha, D. A. d. Andrade, W. M. Torres and R. N. d. Mesquita, “Two-phase flow void fraction estimation based on bubble image segmentation using Randomized Hough Transform with Neural Network (RHTN),” *Progress in Nuclear Energy,* vol. 118, 2020.

[75] J. Ilonen, R. Juránek, T. Eerola, L. Lensu, M. Dubská, P. Zemčík and H. Kälviäinen, “Comparison of bubble detectors and size distribution estimators,” *Pattern Recognition Letters,* vol. 101, pp. 60-66, 2018.

[76] T. Haas, C. Schubert, M. Eickhoff and H. Pfeifer, “BubCNN: Bubble detection using Faster RCNN and shape regression network,” *Chemical Engineering Science,* p. 2020, 216.

[77] I. E. Poletaev, K. S. Pervunin and M. P. Tokarev, “Artificial neural network for bubbles pattern recognition on the images,” *Journal of Physics: Conference Series,* vol. 754, no. 7, 2016.

[78] Z. Gao, L. Hou, W. Dang, X. Wang, X. Hong, X. Yang and G. Chen, “Multitask-based Temporal-Channelwise CNN for Parameter Prediction of Two-phase Flows,” in *IEEE Transactions on Industrial Informatics*, 2020.

[79] Y. Fu and Y. Liu, “BubGAN: Bubble generative adversarial networks for synthesizing realistic bubbly flow images,” *Chemical Engineering Science,* vol. 204, pp. 35-47, 2019.

[80] C. Sunde, S. Avdic and I. Pázsit, “Classification of two-phase flow regimes via image analysis and a neuro-wavelet approach,” *Progress in Nuclear Energy,* vol. 46, no. 3-4, pp. 348-358, 2005.

[81] C. Shanthi, N. Pappa and J. A. Suganya, “Digital Image Processing Based Flow Regime Identification of Gas/Liquid Two - Phase Flow,” *IFAC Proceedings Volumes,* vol. 46, no. 32, pp. 409-414, 2013.

[82] M. Du, H. Yin, X. Chen and X. Wang, “Oil-in-Water Two-Phase Flow Pattern Identification From Experimental Snapshots Using Convolutional Neural Network,” *IEEE Access,* vol. 7, pp. 6219-6225, 2019.

[83] H. Zhou and X. Niu, “An image processing algorithm for the measurement of multiphase bubbly flow using predictor-corrector method,” *International Journal of Multiphase Flow,* vol. 128, 2020.

[84] V. Serdyukov, I. Malakhov and A. Surtaev, “High-speed visualization and image processing of sub-atmospheric water boiling on a transparent heater,” *Journal of Visualization,* vol. 23, no. 5, pp. 873-884, 2020.

[85] D. Tian, “A review on image feature extraction and representation techniques,” *International Journal of Multimedia and Ubiquitous Engineering,* vol. 8, pp. 385-395, 2013.

[86] A. Sethi, “INTERACTION BETWEEN MODULES IN LEARNING SYSTEMS FOR VISION APPLICATIONS,” July 2021.

[87] MATLAB, 9.8.0.1417392 (R2020a) Update 4, Natick, Massachusetts: The MathWorks Inc., 2020.

[88] G. Van Rossum and F. L. Drake Jr, Python tutorial, Amsterdam, The Netherlands: Centrum voor Wiskunde en Informatica , 1995.

[89] MATLAB, MATLAB and Image Processing Toolbox Release 2020a, Natick, Massachusetts: The MathWorks, Inc., 2020.

[90] Itseez, *Open Source Computer Vision Library,* \url{https://github.com/itseez/opencv}, 2015.

[91] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang and Z. DeVito, “PyTorch: An Imperative Style, High-

Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024-8035.

[92] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 9, no. 1, pp. 62-66, 1979.

[93] D. Bradley and G. Roth, "Adaptive Thresholding using the Integral Image. J. Graphics Tools," *J. Graphics Tools,* vol. 12, pp. 13-21, 2007.

[94] A. C. Bovik, Binary Image Morphology, San Diego: Academic Press, 2009, pp. 79-89.

[95] P. Patidar and S. Srivastava, "Image De-noising by Various Filters for Different Noise," *International Journal of Computer Applications,* vol. 9, no. 4, 2010.

[96] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 22, no. 11, pp. 1330-1334, 2000.

[97] J. Heikkila and O. Silven, "A four-step camera calibration procedure with implicit image correction," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Juan, 1997.

[98] F. Meyer, "Topographic distance and watershed lines," *Signal Processing,* vol. 38, pp. 113-125, 1994.

[99] S. Beucher, "The watershed transformation applied to image segmentation," *Scanning Microscopy Supplement,* pp. 299-314, 1992.

[100] P. Pratim Acharjya and D. Ghoshal, "Watershed Segmentation based on Distance Transform and Edge Detection Techniques," *International Journal of Computer Applications,* vol. 52, pp. 6-10, 2012.

[101] C. Aslan, A. Erdem, E. Erdem and S. Tari, "Disconnected Skeleton: Shape at Its Absolute Scale," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 30, no. 12, pp. 2188-2203, 2008.

[102] P. Maragos and R. Schafer, "Morphological skeleton representation and coding of binary images," *IEEE Transactions on Acoustics, Speech, and Signal Processing,* vol. 34, no. 5, pp. 1228-1244, 1986.

[103] I. Sobel, "An Isotropic 3x3 Image Gradient Operator," in *Stanford Artificial Intelligence Project*, 1968.

[104] J. Prewit, "Object Enhancement and Extraction," in *Picture Process and Psychopictoric*, New York, Academic Press, 1970, pp. 75-149.

[105] D. Marr and E.Hildreth, "Theory of Edge Detection," *Proceedings of the Royal Society of London. Series B, Biological Sciences,* vol. 207, no. 1167, 1980.

[106] J. Canny, “A Computational Approach To Edge Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 8, no. 6, pp. 679-698, 1986.

[107] P. R. Reddy, V. Amarnadh and M. Bhaskar, “Evaluation of Stopping Criterion in Contour,” *International Journal of Computer Science and Information Technologies,* vol. 3, no. 3, 2012.

[108] J. Illingworth and J. Kittler, “The Adaptive Hough Transform,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vols. PAMI-9, no. 5, pp. 690-698, 1987.

[109] D. Ballard, “Generalizing the Hough Transform to Detect Arbitrary Shapes,” *Pattern Recognition,* vol. 13, no. 2, pp. 111-122, 1981.

[110] Y. X. Qiang and Q. Ji, “A New Efficient Ellipse Detection Method,” *International Conference on Pattern Recognition,* vol. 2, 2002.

[111] C. A. Basca, M. Talos and R. Brad, “Randomized Hough Transform for Ellipse Detection with Result Clustering,” *The International Conference on "Computer as a Tool",* vol. 2, pp. 1397-1400, 2005.

[112] M. Stojmenovic and A. Nayak, “Direct Ellipse Fitting and Measuring Based on Shape Boundaries,” in *Advances in Image and Video Technology*, Santiago, 2007.

[113] A. Carmona-Poyato, F. Madrid-Cuevas, R. Medina-Carnicer and R. Muñoz-Salinas, “Polygonal approximation of digital planar curves through break point suppression,” *Pattern Recognition,* vol. 43, no. 1, pp. 14-25, 2010.

[114] J.-M. Salotti, “An efficient algorithm for the optimal polygonal approximation of digitized curves,” *Pattern Recognition Letters,* vol. 22, pp. 215-221, 2001.

[115] J. Perez and E. Vida, “Optimum polygonal approximation of digitized curves,” *Pattern Recognition Letters,* vol. 15, pp. 743-750, 1994.

[116] A. Masood and S. Haq, “A novel approach to polygonal approximation of digital curves,” *Journal of Visual Communication and Image Representation,* vol. 18, pp. 264-274, 2007.

[117] C. Teh and R. T. Chin, “On the detection of dominant points on digital curves,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 11, no. 8, pp. 859-827, 1989.

[118] A. Papoulis and S. U. Pillai, Probability, random varilables, and stochastic processes, McGraw-Hill, 1991.

[119] Z. Kalal, K. Mikolajczyk and J. Matas, “Forward-Backward Error: Automatic Detection of Tracking Failures,” in *2010 20th International Conference on Pattern Recognition*, 2010.

[120] A. Voulodimos, N. Doulamis, A. Doulamis and E. Protopapadakis, “Deep Learning for Computer Vision: A Brief Review,” *Computational Intelligence and Neuroscience,* vol. 2018, 2018.

[121] N. O. Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. A. Velasco-Hernandez, L. Krpalkova, D. Riordan and J. Walsh, "Deep Learning vs. Traditional Computer Vision," *CoRR,* vol. abs/1910.13796, 2019.

[122] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[123] Y. LeCun, K. Kavukcuoglu and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE International Symposium on Circuits and System*, 2010.

[124] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012.

[125] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556,* 2014.

[126] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, no. 6, pp. 1137-1149, 2017.

[127] K. Weiss, T. Khoshgoftaar and D. Wang, "A survey of transfer learning," *Journal of Big Data,* vol. 3, no. 9, 2016.

[128] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei., "Large-scale video classification with convolutional neural networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[129] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Proc. NIPS*, Montreal, Canada, 2014.

[130] J. Y.-H. Ng, M. J. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga and G. Toderici, "Beyond Short Snippets: Deep Networks for Video Classification," *CoRR,* vol. abs/1503.08909, 2015.

[131] R. M. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," *arXiv preprint arXiv:1912.05911,* 2019.

[132] L. Pigou, A. Oord, S. Dieleman, M. Van Herreweghe and J. Dambre, "Beyond Temporal Pooling: Recurrence and Temporal Convolutions for Gesture Recognition in Video," *International Journal of Computer Vision,* vol. 126, no. 2, 2018.

[133] D. Tran, L. Bourdev, R. Fergus, L. Torresani and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015.

[134] S. Ji, W. Xu, M. Yang and K. Yu, "3D Convolutional Neural Networks for Human Action Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 35, no. 1, pp. 221-231, 2013.

[135] K. Hara, H. Kataoka and Y. Satoh, “Learning spatio-temporal features with 3d residual networks for action recognition,” in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2017.

[136] T. Brox, A. Bruhn, N. Papenberg and J. Weickert, “High Accuracy Optical Flow Estimation Based on a Theory for Warping,” in *Proc. ECCV*, 2004.

[137] H. Wang, A. Klaser, C. Schmid and C.-L. Li, “Action Recognition by dense trajectories,” in *Proc. CVPR*, 2011.

[138] G. Farnebäck, “Two-Frame Motion Estimation Based on Polynomial Expansion,” *In: Image analysis,* vol. 2749, pp. 363-370, June 2003.

[139] B. Zhang, L. Wang, Z. Wang, Y. Qiao and H. Wang, “Real-time action recognition with enhanced motion vector CNNs,” in *CVPR*, 2016.

[140] Y. Bohra, “The Challenge of Vanishing/Exploding Gradients in Deep Neural Networks,” 18 June 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networks/.

[141] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[142] H. Hewamalage, C. Bergmeir and K. Bandara, “Recurrent Neural Networks for Time Series Forecasting: Current status and future directions,” *International Journal of Forecasting,* vol. 37, no. 1, pp. 388-427, 2021.

[143] W. D. Mulder, S. Bethard and M.-F. Moens, “A survey on the application of recurrent neural networks to statistical language modeling,” *Computer Speech and Language,* vol. 30, no. 1, pp. 61-98, 2015.

[144] L. Zhang, S. Wang and B. Liu, “Deep learning for sentiment analysis: A survey,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* vol. 8, no. 4, p. e1253, 2018.

[145] B. Shi, X. Bai and C. Yao, “An End-to-End Trainable Neural Network for Image-based Sequence Recognition,” *CoRR,* vol. abs/1507.05717, 2015.

[146] O. Vinyals, A. Toshev, S. Bengio and D. Erhan, “Show and Tell: A Neural Image Caption Generator,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014.

[147] P. Sahu, D. Yu, K. Yager, M. Dasari and H. Qin, “In-Operando Tracking and Prediction of Transition in Material System using LSTM,” in *AI-Science*, Tempe, 2018.

[148] R. Carrasco-Davis, G. Cabrera-Vives, F. Förster, P. A. Estévez, P. Huijse, P. Protopapas, I. Reyes, J. Martínez-Palomera and C. Donoso, “Deep Learning for Image Sequence Classification of

Astronomical Events," *Publications of the Astronomical Society of the Pacific,* vol. 131, no. 1004, 2019.

[149] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia and A. Baskurt, "Action classification in soccer videos with long short-term memory recurrent neural networks.," in *Proc. ICANN*, Thessaloniki, 2010.

[150] M. Liu and M. Zhu, "Mobile video object detection with temporally-aware feature maps," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

[151] A. Broad, M. Jones and T.-Y. Lee, "Recurrent Multi-frame Single Shot Detector for Video Object Detection," in *BMVC*, 2018.

[152] V. Campos, B. Jou, X. Giro-i-Nieto, J. Torres and S.-F. Chang, "Skip rnn: Learning to skip state updates in recurrent neural networks," *arXiv preprint arXiv:1708.06834},* 2007.

[153] F. Nie, H. Zhanxuan and X. Li, "An investigation for loss functions widely used in machine learning," *Communications in Information and Systems,* vol. 18, pp. 37-52, 2018.

[154] S. Kornblith, H. Lee, T. Chen and M. Norouzi, "What's in a Loss Function for Image Classification?," *CoRR,* vol. abs/2010.16402, 2020.

[155] P. Simard, D. Steinkraus and J. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis," 2003, pp. 958-962.

[156] K. E. Koech, "Cross-Entropy Loss Function," 2 October 2020. [Online]. Available: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e. [Accessed 11 November 2021].

[157] J. Du, "The Frontier of SGD and Its Variants in Machine Learning," *Journal of Physics: Conference Series,* vol. 1229, 2019.

[158] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.

[159] T. Evgeniou and M. Pontil, "Support Vector Machines: Theory and Applications," in *Advanced Course on Artificial Intelligence*, Chania, Greece, 2001.

[160] G. Foody and A. Mathur, "IEEE Transactions on Geoscience and Remote Sensing," *IEEE Transactions on Geoscience and Remote Sensing,* vol. 42, no. 6, pp. 1335-1343, 2004.

[161] P. Wang, E. Fan and P. Wang, "Comparative analysis of image classification algorithms based on traditional machine learning and deep learning," *Pattern Recognition Letters,* vol. 141, pp. 61-67, 2021.

[162] H. Azizpour, J. Sullivan and S. Carlsson, “Cnn features off-the-shelf: An astounding baseline for recognition.,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014.

[163] S. Na, L. Xumin and G. Yong, “Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm,” in *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, 2010.

[164] T. Kodinariya and P. Makwana, “Review on Determining of Cluster in K-means Clustering,” *International Journal of Advance Research in Computer Science and Management Studies,* vol. 1, pp. 90-95, 2013.

[165] N. Dhanachandra, K. Manglem and Y. J. Chanu, “Image Segmentation Using K -means Clustering Algorithm and Subtractive Clustering Algorithm,” *Procedia Computer Science,* vol. 54, pp. 764-771, 2015.

[166] A. G. Hinton and M. G. E. Abdel-rahman, “Speech Recognition with Deep Recurrent Neural Networks,” *CoRR,* vol. CoRR, 2013.

[167] A. Wilson, R. Roelofs, M. Stern, N. Srebro and B. Recht, “The Marginal Value of Adaptive Gradient Methods in Machine Learning,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017*, 2017.

[168] R. Carrasco-Davis, G. Cabrera-Vives, F. Förster, P. A. Estévez, P. Huijse, P. Protopapas, I. Reyes, J. Martínez-Palomera and C. Donoso, “Deep Learning for Image Sequence Classification of Astronomical Events,” *Publications of the Astronomical Society of the Pacific,* vol. 131, no. 1004, 2019.

[169] H. Salehinejad, S. Sankar, J. Barfett, E. Colak and S. Valaee, “Recent advances in recurrent neural networks,” *arXiv preprint arXiv:1801.01078,* 2017.

[170] G. Vanwinckelen and H. Blockeel, “On Estimating Model Accuracy with Repeated Cross-Validation,” in *BeneLearn 2012: Proceedings of the 21st Belgian-Dutch Conference on Machine Learning*, 2012.